Diseño e implementación de espacios inteligentes

Erwin Aitenbichler, Fernando Lyardet, Max Mühlhäuser
Telecooperation Group, Department of Computer Science, Darmstadt University of Technology, Alemania

<{erwin,fernando,max}@tk informatik tu-darmstadt.de>

Traducción: Mercedes Montes Rubio (socia de ATI)

Resumen: *el proyecto Mundo del Telecooperation Group está dedicado a modelos generales y arquitecturas para computación ubicua. El sistema Mundo Smart Environments (Entornos Inteligentes Mundo) proporciona las herramientas y los servicios esenciales necesarios para construir aplicaciones para dichos entornos. Mientras que el desarrollo de servicios sencillos es una práctica común, el como coordinarlos y como hacer que los ensamblados de servicios se comporten de forma inteligente es todavía un tema de investigación. Presentamos un proceso de desarrollo de software para el desarrollo sistemático de aplicaciones de espacios inteligentes. Este proceso es ayudado por un conjunto de herramientas y servicios comunes para el modelado, inspección, depuración, pruebas y una rápida creación de prototipos. Describimos además como estas herramientas se aplican en determinadas fases para ayudar al proceso.*

Palabras clave: *computación ubicua, creación rápida de prototipos, entorno inteligente, espacio inteligente, sensibilidad al contexto.*

Autores

Erwin Aitenbichler se licenció en Informática por la Universidad Johannes Kepler de Linz, Austria y se doctoró en Informática por la Universidad Tecnológica de Darmstadt, Alemania. Es en la actualidad investigador postdoctorado en el grupo Telecooperation del Departamento de Informática de la citada Universidad. Sus temas de investigación son los entornos inteligentes y la computación ubicua. Erwin es miembro de la ACM.

Fernando Lyardet se licenció en Informática por la Universidad Nacional de La Plata (UNLP), Argentina, y en la actualidad es investigador adjunto en el grupo Telecooperation de la Universidad Tecnológica de Darmstadt, Alemania. Sus temas de investigación incluyen entornos inteligentes, productos inteligentes y computación ubicua. Es también miembro de la ACM y la AAAI.

Max Mühlhäuser es profesor de Informática en la Universidad Tecnológica de Darmstadt, Alemania. Se doctoró en informática por la Universidad de Karlsruhe y fundó un centro de investigación para Equipamiento Digital. Desde 1989 ha trabajado tanto como profesor como profesor invitado en distintas Universidades de Alemania, Austria, Francia, Canada y Estados Unidos. Max ha publicado más de 200 artículos y ha sido coautor, así como editor de libros, sobre e-learning, ingeniería de software distribuido y multimedia así como de computación ubicua.

1. Introducción

Los espacios o entornos inteligentes son áreas para vivir o trabajar donde la tecnología informática se integra dentro de la infraestructura del edificio. El objetivo típico es facilitar las interacciones con el sistema informático o hacer los procesos de trabajo más eficientes. Los espacios inteligentes son intrísicamente sistemas distribuidos con muchos nodos informáticos hetereogéneos. Las aplicaciones de tecnología puntera se basan en arquitecturas orientadas a servicio (*Service-Oriented Architectures*, SOA). Mientras que es de sobra conocido como implementar servicios para una aplicación sencilla, todavía queda por resolver como podemos coordinar servicios eficientemente y como hacer ensamblados de servicios "inteligentes".

En este artículo nos centramos en la coordinación de servicios basada en contexto desde el lado de la infraestructura.

Para realizar esto, describimos una capa de *middleware* que contiene un conjunto de servicios útiles para cualquier aplicación de espacio inteligente, lo que ayuda en las importantes áreas de comunicación, contexto y coordinación.

Comunicación: un espacio inteligente contiene multitud de dispositivos heterogéneos. La integración de todos estos sistemas se alcanza siempre a nivel de software. De esta forma, en el nivel básico, un sistema distribuido en tiempo de ejecución es el pegamento que une todos los servicios de software que se ejecutan en distintas plataformas, sistemas operativos o lenguajes de programación. Incluso debe admitir conexiones de redes espontáneas con dispositivos que los usuarios introducen en el entorno.

Contexto: en los espacios inteligentes, las interacciones ya no se realizarán de forma estática, ya que tienen lugar en ubicaciones que cambian o en contextos de libre movimiento. Por esta razón, las ubicaciones de la gente y los objetos en el espacio juegan un papel importante. Las ubicaciones de objetos estáticos pueden definirse en un modelo, mientras que las ubicaciones de los objetos móviles pueden determinarse mediante sensores. Además de los contextos de ubicación, diferentes sensores (de inclinación, aceleración o peso, micrófonos, etc.) o servicios (calendarios, programaciones, sistema de reserva de habitación, etc.) pueden proporcionar valiosa información sobre los usuarios y sus tareas. Los datos de entrada obtenidos de los sensores deben procesarse y transformarse en información de contexto, para que a las aplicaciones les resulten útiles.

Coordinación: después de programar la funcionalidad de la aplicación en un conjunto de servicios, la lógica de la aplicación en su totalidad debe programarse, bien en algún lenguaje de programación, o bien diseñarse con herramientas apropiadas en un enfoque basado en modelos. En este área, las herramientas de desarrollo rápido de aplicaciones (*Rapid Application Development*, RAD) son altamente beneficiosas. En primer lugar, permiten realizar un rápido prototipo de las aplicaciones, lo que es importante en la investigación de computación ubicua. En segundo lugar, posibilitan a los técnicos o usuarios finales que no poseen habilidades de programación la personalización de la funcionalidad de aplicaciones de alto nivel.

El presente artículo describe el sistema de Entornos Inteligentes Mundo (*Mundo Smart Environments*). Este trabajo es parte del proyecto Mundo [1], dedicado a los modelos y arquitecturas generales para la computación ubicua. Presentamos una arquitectura de software para entornos inteligentes, y los principales servicios y herramientas de desarrollo necesarios que proporcionan soluciones para las tres áreas descritas. Posteriormente, describimos un proceso de desarrollo de software para aplicaciones de espacios inteligentes y como se emplean nuestras herramientas en este proceso.

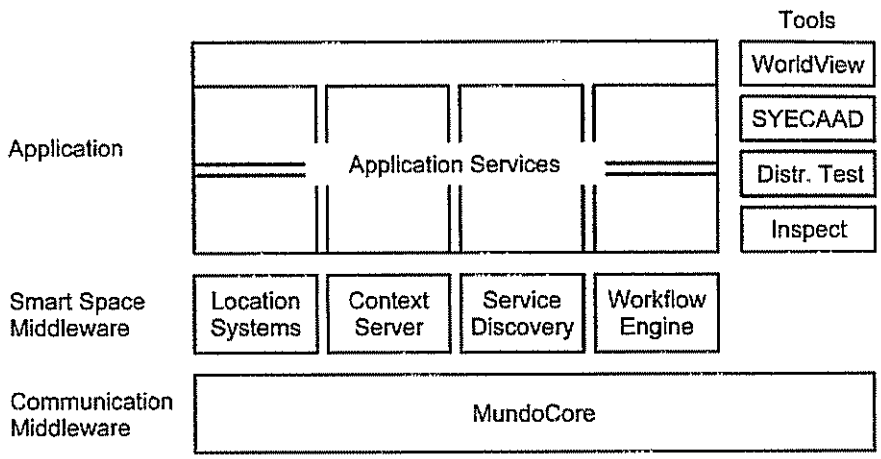

Figura 1. Arquitectura de software *Mundo Smart Environments*

El artículo se estructura de la siguiente forma. En la **sección 2** damos una visión general de nuestra arquitectura de software *Mundo Smart Environments*. Basándose en esta plataforma, las aplicaciones se desarrollan de acuerdo al proceso descrito en la **sección 3**. También mostramos como se emplean nuestras herramientas para dar soporte a ciertas tareas de dicho proceso. La aplicación de ejemplo en la **sección 4** muestra la aplicación de este proceso. En la **sección 5** describimos los trabajos relacionados y finalmente concluimos en la **sección 6**.

2. Plataforma de Entornos Inteligentes

La estructura de nuestras aplicaciones de entorno inteligente, se basa en su totalidad en una arquitectura orientada a servicios, como se muestra en la **figura 1**. La comunicación *middleware MundoCore* [2] proporciona las bases de un software común y permite la comunicación entre los ordenadores en el sistema distribuido. Los clásicos sistemas *middleware* distribuidos admiten aspectos como resolución de nombres, llamadas a procedimientos remotos y computación de objetos distribuidos. En la computación ubicua, existen nuevos tipos de servicio requeridos por casi todas las aplicaciones, por lo que resulta útil trasladar esta funcionalidad a la capa *middleware*. Los servicios comunes para entornos inteligentes incluyen rastreo de ubicación, procesamiento de contexto, descubrimiento de servicios y soporte a flujos de trabajo (*workflow*). El *middleware* y sus servicios se explicarán más ampliamente en las siguientes secciones.

2.1. MundoCore

MundoCore es la capa inferior del *middleware* en nuestra plataforma de entornos inteligentes. Es el encargado de todos los aspectos relacionados con la comunicación y se diseñó específicamente para las necesidades de los servicios en las capas superiores. El objetivo original del *middleware* de computación de objetos distribuidos era permitir la cooperación de los objetos con independencia de los dispositivos, sistemas operativos y lenguajes de programación. Con el tiempo, los ordenadores personales y las plataformas de servidor se hicieron más potentes y no tuvieron problemas en ejecutar software *middleware* de gran tamaño, monolítico y mal optimizado. La computación ubicua introduce un amplio espectro de nuevas plataformas de computación, tremendamente diferente en cuanto a tamaño, movilidad y usabilidad. El nivel inferior de este espectro está marcado por los ordenadores embebidos en los objetos cotidianos y pequeños nódulos de sensor que a menudo, sólo tienen capacidades de procesamiento muy limitadas.

MundoCore se basa en un *diseño microkernel*, admite reconfiguración dinámica y proporciona un conjunto de APIs común para distintos lenguajes de programación (Java, C++, Python) en una amplia gama de dispositivos. Por su pequeño tamaño puede ejecutarse también en muchos sistemas embebidos. La ventaja del C++ es que resulta fácil el acceso al hardware, a APIs del sistema que operan a nivel inferior y a la programación eficiente de servicios de procesamiento de vídeo y audio. Por lo tanto, MundoCore C++ se utiliza principalmente para servicios de nivel inferior y para servicios donde es crítico el rendimiento. La mayoría de los servicios de nivel superior se programan en Java, debido a la alta productividad. Las distintas versiones de MundoCore son compatibles a nivel de protocolo.

La arquitectura interna de MundoCore atiende de la necesidad de diferentes protocolos de invocación y transporte, descubrimiento automático de parejas, superposición entre pares (*peer-to-peer overlay*), diferentes abstracciones de comunicación y las vinculaciones adecuadas del lenguaje. Una API eficaz debe proporcionar funciones adecuadas a los programadores para sus entornos específicos de aplicación. La computación de objetos distribuidos es un modelo de programación ampliamente aceptado y fácil de usar. En sistemas sensibles al contexto y otros sistemas dirigidos por información, la publicación/suscripción es la mejor abstracción para eventos distribuidos porque admite *multicasting* y productores de datos independientes de los consumidores de datos. El obtener del *middleware* las abstracciones adecuadas nos conduce a reducir el tiempo de desarrollo y la cantidad de código de la aplicación.

Además de los eventos, las llamadas en modo remoto pueden implementarse basándose en el sistema de publicación/suscripción. De esta forma, los sistemas se independizan unos de otros y ganan en transparencia de acceso y ejecución. Cuando un cliente envía peticiones a un servicio, la abstracción publicación/suscripción proporciona una funcionalidad similar a un sencillo servicio de identificación. Debido a que los servicios subscriben sus interfaces en el lugar de ejecución, dichos servicios pueden iniciarse en cualquier parte del sistema. Esto nos permite definir los lugares de ejecución de los servicios en tiempo de instalación de forma flexible o migrar dinámicamente los servicios en tiempo de ejecución.

2.2 Servidor de Contexto

El Servidor de Contexto Mundo [3] es el encargado de transformar las lecturas recogidas de los sensores en información entendible para las aplicaciones. El servidor de contexto proporciona la siguiente funcionalidad:

- Interpreta los datos recibidos de los sensores y los transforma en una representación común.
- Mantiene un modelo geométrico del entorno inteligente y admite consultas y operaciones geométricas.
- Deduce contexto de alto nivel desde el contexto de bajo nivel.
- Notifica a las aplicaciones cuando cambian ciertas propiedades del contexto.
- Almacena historiales de contexto detectado e inferido, y admite consultas sobre estos historiales.

El servidor está basado en la idea de controles (*widgets*) para procesar la información de contexto. Los datos producidos por los sensores son ahora muy hetereogéneos. En primer lugar, el servidor transforma los datos recogidos en un reducido número de representaciones de datos comunes. Por ejemplo, los diferentes formatos de los datos y los eventos de los lectores RFID, lectores de transpondedor[1], o sistemas de placas de señales infrarrojas se transforman en un *tipo de datos ID* común y en un conjunto de *eventos de lectura* comunes. Igualmente, las

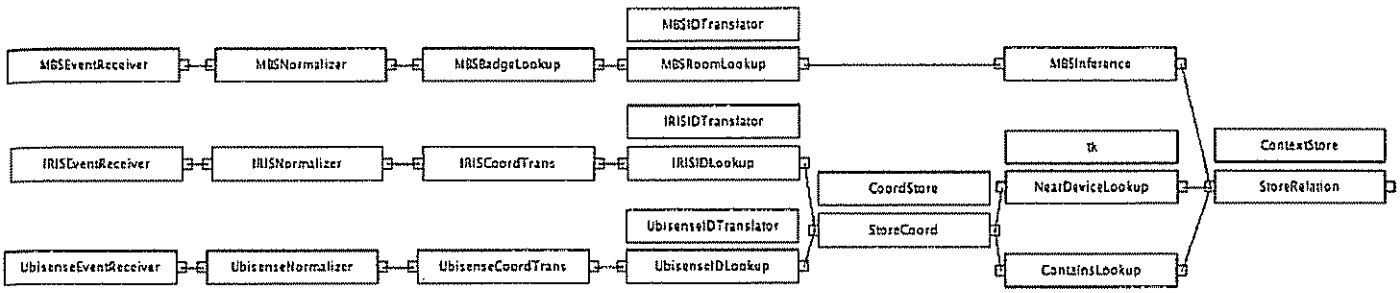

Figura 2. Configuración de controles del Servidor de Contexto para la ubicación de usuarios.

posiciones y orientaciones obtenidas de los sistemas de rastreo en 3D se transforman en un sistema coordinado común. Hasta este punto, las IDs representan etiquetas, transpondedores o placas. El siguiente paso es la realización de un plano uniendo estas identificaciones con las identificaciones de las personas u objetos que llevan dichas etiquetas.

Para obtener información de contexto del nivel superior, el servidor de contexto utiliza un contexto modelado, por ejemplo un espacio modelo geométrico detallado (2D o 3D). Un modelo es una copia virtual de un entorno real, es el objetivo de las aplicaciones y básicamente es un modelo geométrico detallado que contiene paredes, muebles y otros objetos. El modelo se aumenta con una capa de metadatos que describe los objetos y áreas de interés. Con la ayuda de este contexto modelado, el servidor es ahora capaz de deducir información de contexto del nivel superior desde los sistemas de ubicación que proporcionan coordenadas 3D y orientaciones tales como, en que habitación se encuentra un usuario, que objetos se encuentran cerca del usuario o a que objeto está mirando el usuario en ese momento. En consecuencia, la información entregada a las aplicaciones está ya sintetizada por los sensores subyacentes y describe los cambios en el contexto de nivel superior.

Por ejemplo, consideremos la configuración de controles mostrada en la **figura 2**. Es habitual obtener ciertas relaciones entre los usuarios y los objetos, basándose en tres sistemas distintos de rastreo de ubicación: el sistema de placas Mundo, un sistema de rastreo óptico por infrarrojos y el sistema de rastreo *Ubisense*. El sistema de placas proporciona directamente información de ubicación simbólica. De esta manera, sólo es necesario realizar una correspondencia entre placas específicas del sistema e identificadores de habitación, e identificadores globales. Los datos de los sistemas de rastreo 3D deben interpretarse adicionalmente utilizando el modelo global. Todas las relaciones derivadas se escriben en un espacio de tuplas. Con la configuración de equipo mostrada en la **figura 2**, la situación de la **figura 4** y dado que los usuarios pueden ser rastreados por al menos uno de los tres sistemas de ubicación, el servidor de contexto obtendría las relaciones para la habitación A112 que se muestran en la **tabla 1**.

Las aplicaciones pueden consultar el espacio de tuplas o subscribir cambios. Una operación de consulta devuelve todas las relaciones que encajan con un patrón específico. Cuando una aplicación se subscribe al espacio de tuplas, se le notificará cada vez que se escriba en el espacio una tupla nueva o actualizada que encaje con el patrón especificado.

Existen dos razones principales para implementar el procesamiento del contexto como parte del *middleware*, a parte de las aplicaciones. En primer lugar, este servicio es requerido por muchas aplicaciones de computación ubicua y como parte de un *middleware* común se puede reutilizar fácilmente.

En segundo lugar, el procesamiento del contexto es una tarea que lleva mucho tiempo y puede necesitar grabar información incluso durante mucho más tiempo. Puede que las aplicaciones funcionen por poco tiempo pero pueden necesitar acceso a información del pasado, recogida durante un largo periodo. El acceso a la información del pasado requiere que esta información haya sido grabada con antelación. En consecuencia, el servidor de contexto también debe configurarse de manera que grabe toda la información que se necesitará con posterioridad.

3. Proceso de Desarrollo

La **figura 3** muestra las diferentes fases del proceso de desarrollo del software, basado en el modelo de cascada y que ha sido refinado para ciertas tareas. En muchas de las fases las herramientas existentes son suficientes, por ejemplo una IDE (*Integrated Development Environment*) para programación Java o una JUnit para realizar pruebas unitarias. En las fases restantes introducimos herramientas adicionales para ayudar en el proceso de desarrollo. A continuación, describimos como estas herramientas ayudan en estas diferentes fases.

3.1. Fase de análisis

En la fase de análisis, se debe identificar en primer lugar desde que contexto de información se puede beneficiar la aplicación. Debido a que la utilización de la información de contexto es altamente específica de cada aplicación, es casi imposible crear un modelo de contexto universal que abarque cualquier posible requerimiento. En cambio, nuestro objetivo es definir procesos que describan como seleccionar sensores, configurar el procesamiento y como hacer accesible este contexto de información para la aplicaciones.

Se pueden seleccionar los sensores necesarios basándose en los requerimientos de información. La resolución y precisión requeridas juegan un papel importante en este proceso de selección, en especial para los sistemas de rastreo de ubicación.

A continuación, se define la interfaz para la aplicación, que incluye el formato de mensaje y las clases de subscripciones y consultas que necesita. Por último, el servidor de contexto se configura de tal manera que transforma las lecturas de los sensores en lo que la aplicación necesita. Debido a que la información de contexto sólo será accesible para lecturas posteriores cuando se grabe a tiempo, los controles de la base de datos deben ser configurados adecuadamente. Asimismo, se debe decidir que información tiene que almacenarse y por cuanto tiempo.

3.2. Fase de diseño

WorldView es una herramienta versatil con

Usuario:Erwin	Dentro	Habitación:A112
Usuario:Erwin	Cerca	Dispositivo:PolyvisionBoard
Usuario:Andreas	Dentro	Habitación:A112

Tabla 1. Ejemplo de las tuplas obtenidas para describir las relaciones en la habitación 112

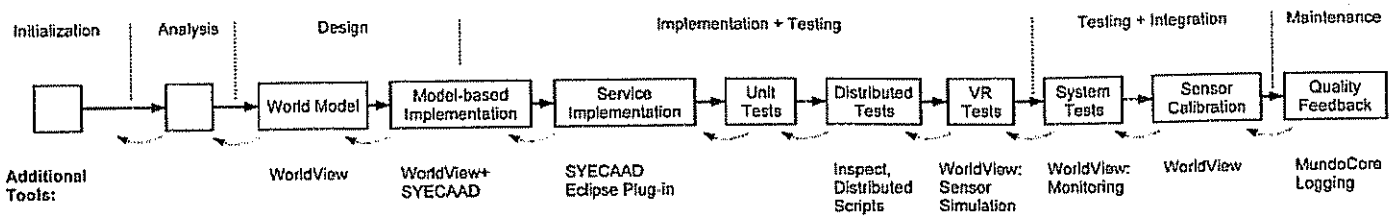

Figura 3. Herramienta de ayuda para las diferentes fases del desarrollo de software.

funciones para ayudar en las fases de modelado, implementación y pruebas. En la fase de modelado se emplea WorldView para crear un modelo espacial del espacio inteligente. Admite modelos en 2D así como modelos geométricos detallados en 3D. En la ventana del mapa, la aplicación muestra el trazado del suelo y proporciona una visión de los recursos disponibles y sus ubicaciones. Los recursos incluyen etiquetas y sensores de los distintos sistemas de ubicación, visores de pared y placas inteligentes identificadoras de puertas.

WorldView ofrece una forma sencilla de definir "regiones de interés" con formas arbitrarias, que pueden ser anotadas con metadatos. Las regiones pueden cargarse en el servidor de contexto como parte del modelo y pueden accederse a través de los controles del contexto para derivar información de contexto de nivel superior. Las aplicaciones sensibles a la ubicación pueden especificar suscripciones basadas en ubicaciones semánticas en lugar de tener que trabajar con coordenadas del sistema de rastreo de ubicación subyacente. De esta forma, WorldView proporciona un método sencillo para definir las regiones de interés que deberían desencadenar eventos espaciales

3.3. Fase de Implementación

Muchas de las investigaciones sobre computación ubicua se llevan a cabo sobre posibles escenarios de aplicación de uso diario. Muchos de estos escenarios son bastante sencillos y fáciles de implementar. Sin embargo, muchas de estas aplicaciones nunca llegan a ver la luz ya que todo el proceso desde el desarrollo hasta su implantación es aún muy complejo. Para desarrollar una aplicación nueva el programador tiene que empezar normalmente por su IDE, instalar las bibliotecas requeridas, escribir el código, probarlo y después instalar la aplicación en el servidor. Por esta razón quisimos hacer este proceso de desarrollo tan fácil y rápido como fuera posible. Uno de los objetivos fue el permitir a una amplia variedad de personas, que tuvieran alguna formación técnica básica, pero no necesariamente conocimientos de un lenguaje de programación, crear y personalizar aplicaciones. Las aplicaciones de estructura sencilla pueden desarrollarse directamente con las herramientas proporcionadas, mientras que de las aplicaciones más complejas pueden realizarse prototipos.

La herramienta *SYstem for Easy Context Aware Application Development* (SYECAAD) [4] facilita el rápido desarrollo de aplicaciones sensibles al contexto permitiendo desarrollar por completo aplicaciones sencillas y realizar prototipos de los sistemas más complejos. Las aplicaciones se construyen utilizando un modelo de bloque orientado a gráficos. Los bloques de construcción básicos se llaman *unidades funcionales*, tienen conectores de entrada-salida y están interconectados para formar *ensamblados funcionales*.

Las unidades funcionales se presentan de tres formas distintas: *sensores*, *operaciones* y *actores*. Una unidad *sensor* recibe tanto datos directamente del sensor como datos preprocesados desde el servidor del contexto. Las *operaciones* efectúan cálculos lógicos o aritméticos, implementan diccionarios, presentan páginas HTML, etc. En la salida, los *actores* pueden controlar el entorno inteligente o enviar información a los usuarios publicando eventos MundoCore o invocando métodos en servicios remotos arbitrarios. De esta forma una aplicación puede por ejemplo controlar clavijas de control inteligentes, proyectores de datos de control, enviar correos electrónicos, SMS a móviles, mensajes de mensajería instantánea, o visualizar información en las placas electrónicas de identificación de puertas

Los ensamblados funcionales son un concepto de modularización. Pueden instalarse, eliminarse, rearrancarse y editarse de forma independiente. En la práctica, la lógica para cada habitación de un edificio sería modelada en un ensamblado separado. Una característica especial del SYECAAD es que puede calcular los estados de salida de las unidades funcionales basándose tanto en un modelo de eventos como en uno basado en estados. Cada vez que el valor de un sensor cambia, todas las operaciones que dependen de este valor deben evaluar de nuevo su estado. En un modelo basado en eventos, los cambios se propagan como mensajes a través del sistema. Sin embargo, cuando una unidad funcional se pone en marcha, los valores de salida no estarán disposibles antes de que todas las entradas hayan propagado los mensajes de cambio, lo que no es muy práctico durante el desarrollo del sistema. En un modelo basado en estados, el estado de todas las unidades se evalúa en un intervalo fijo. Aunque esta solución no es eficaz, se dispone de todos los estados de las salidas de forma inmediata. El modelo híbrido utilizado en SYECAAD permite una ejecución eficaz e incluso realizar cambios en el sistema mientras está funcionando, casi sin perder el estado.

La herramienta SYECAAD utiliza una arquitectura cliente/servidor. El servidor alberga las aplicaciones, mientras que los clientes se conectan al servidor y permiten controlar, editar y probar las aplicaciones en ejecución. El editor de *WorldView (Application Logic Editor)* es uno de estos clientes, encargado de editar aplicaciones (ver **figura 4**). Este sistema simplifica significativamente el desarrollo de la aplicación puesto que ya no es necesario establecer un entorno de desarrollo, porque toda la lógica de la aplicación se almacena de forma centralizada en el servidor de aplicación. El entorno de desarrollo es una aplicación cliente que se conecta a este servidor y permite de esta forma descargar la aplicación desde el servidor, visualizarla, editarla e instalarla con sólo pulsar un botón.

Si los bloques estándar de sensor, operación y actor no son suficientes, se puede ampliar el sistema programando nuevos bloques en Java. Un *plug-in* para la IDE Eclipse ayuda al programador con plantillas de código y documentos de ayuda.

3.4. Fase de Pruebas

Las implementaciones de tipos abstractos de datos y unidades más pequeñas de los entornos de trabajo pueden probarse con éxito mediante *pruebas unitarias*. Sin embargo, para comprobar el correcto funcionamiento de un sistema distribuido es también esencial ejecutar pruebas de integración a través de múltiples ordenadores.

Para llevar a cabo dichas pruebas hemos implementado un Intérprete de *Scripts* Distribuido (*Distributed Script Interpreter*). Los procesos de servidor de *scripts* se arrancan en múltiples ordenadores *host* en la red. Los servidores de *scripts* y el intérprete maestro de *scripts* se basan en MundoCore para la comunicación, lo que les permite identificar-

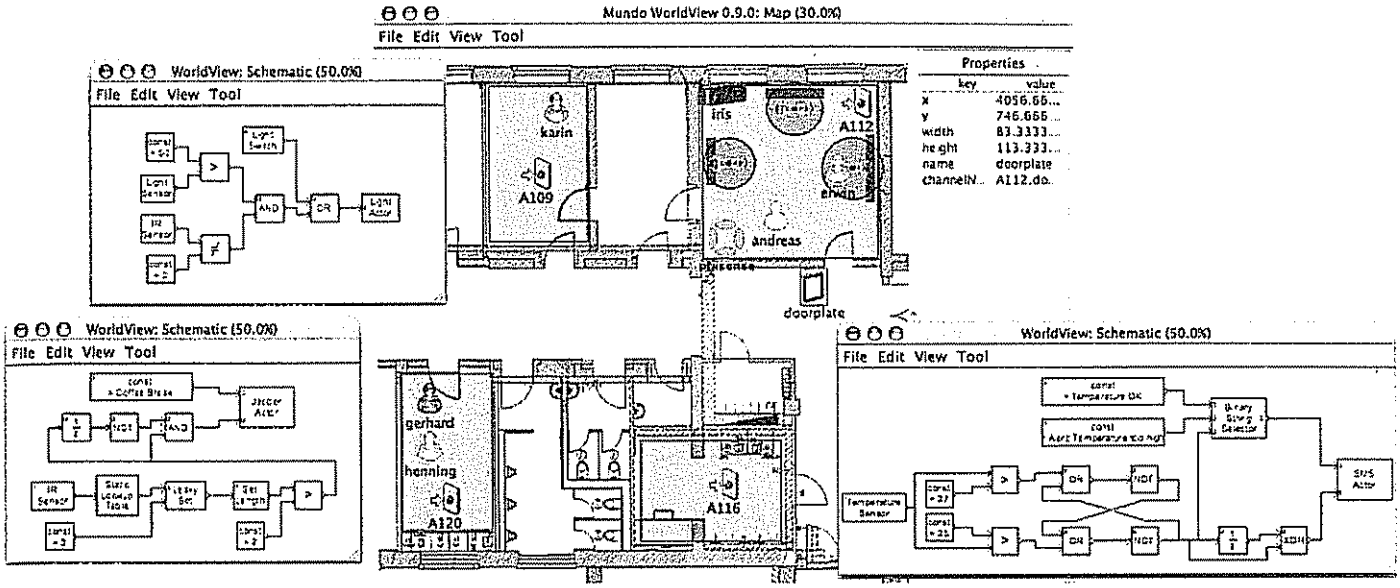

Figura 4. Modelo espacial en *WorldView* y lógica de aplicación para tres habitaciones

se mutuamente de forma automática en el arranque. Para ejecutar una prueba, se pasa el nombre de un fichero de código XML al intérprete maestro que distribuye las subtareas a los procesos de servidor en la red y por último recoge todos los resultados de la prueba.

La herramienta *MundoCore Inspect* es una herramienta de bajo nivel que se construye directamente en el *middleware* de comunicación. Se puede conectar a un nodo remoto arbitrario y gestionar los servidos albergados. El programa permite ver las tablas de asignación de rutas, listar las tablas de importaciones y exportaciones de los *brokers* de mensajes, controlar los mensajes entregados por un canal, ver las interfaces de servicio, llamar dinámicamente a métodos remotos con un cliente genérico, ver la información de configuración de servicio y reconfigurar servicios.

Con el *VR-Tests* es posible probar aplicaciones desde el escritorio, mientras que *WorldView* puede emplearse para simular ciertos sistemas de rastreo. En este caso, el usuario se puede mover en el mapa alrededor de los símbolos y WorldView genera el mismo tipo de eventos que generaría el sistema real de rastreo. Esto reduce significativamente el tiempo de desarrollo de la aplicación, porque los usuarios no tienen que levantarse de su puesto de trabajo y mover objetos físicos cada vez que quieren probar sus aplicaciones.

3.5. Instalación y Mantenimiento

Cuando los resultados de *VR-Tests* son satisfactorios, entonces se puede probar la aplicación con los sensores reales. En esta fase, la aplicación *WorldView* se puede utilizar para inspeccionar el sistema en funcionamiento mediante la visualización de los eventos de ciertas fuentes de eventos como los sistemas de rastreo. Si una etiqueta se mueve físicamente, la posición del símbolo correspondiente se actualiza en tiempo real en la vista del mapa.

El *middleware* de MundoCore implementa depuración de memoria dinámica, rastreo de recursos del sistema, detección de abrazos mortales (*deadlocks*), control del progreso y registro de eventos (*logging*). Algunos errores no se detectan hasta que el sistema se prueba con los sensores reales. En este caso, las anotaciones detalladas proporcionan información relevante para que los desarrolladores puedan arreglar el problema.

4. Ejemplo de aplicación

A continuación describiremos tres sencillas aplicaciones creadas con nuestras herramientas. La configuración para tres habitaciones se muestra en la figura 4.

Office: para la habitación *Office* A109 definimos que si el nivel de luz se encuentra por debajo de un cierto límite y existe al menos una persona en la habitación, entonces las luces se encienden, aunque de forma alternativa se puede utilizar el interruptor manual de la luz. Esta aplicación se basa en un sensor de luz y en un sistema de rastreo de ubicación como entradas, y como salidas lámparas controladas por LON[2].

Coffee kitchen: la configuración de la *Coffee kitchen* A120 describe que si al menos dos personas se encuentran en la cocina durante un tiempo, entonces se envía un mensaje instantáneo a toda la gente de las oficinas cercanas invitándolas a "hacer vidal social". Esta aplicación se basa en un sistema de rastreo de ubicación como entrada y en un servicio que implementa el protocolo Jabber para enviar mensajes instantáneos.

Server room: en nuestro nuevo edificio informático el sistema de aire acondicionado falla de vez en cuando. La configuración para A116 define que si la temperatura supera los 27 grados entonces se notifica al administrador del sistema mediante un SMS. Cuando la temperatura baja por debajo de los 25 grados, se envía otro SMS. Esta aplicación se basa en un termómetro SNMP como entrada y en un servicio para envio de mensajes cortos a través de un sitio web de Internet.

5. Trabajos relacionados

Los entornos inteligentes se examinan en numerosos proyectos tales como Gaia [5], Aura [6], Nexus [7] e iWork [8], y se han construido muchos kits de herramientas e infraestructuras para el desarrollo y ayuda a dichas aplicaciones. Estas tecnologías proporcionan ideas novedosas para abordar las distintos temas de diseño tales como la interface [9][10][11], colaboración entre dispositivos hetereogéneos [12] y sensibilidad al contexto [13]. Algunas herramientas para desarrollar modelos de entorno se encuentran también disponibles en los productos comerciales LBS como Ubiense [14] y Elpas [15], pero el público al que se dirigen es diferente. Mientras que Ubiense proporciona una interfaz C++ orientada a desarrolladores, la solución basada en reglas de Elpas permite al usuario final desarrollar a través de su editor de reglas.

La capacidad de desarrollar del usuario final ha sido el objeto de diferentes soluciones. Por ejemplo, CAMP [16] proporciona un languaje natural simplificado utilizando la

idea de "Poesía Magnética", y Hull describe un sistema que proporciona "paisajes mediáticos" basados en modelos de entorno y *scripting* [17]. En el entorno del proyecto Nexus también se desarrolló una herramienta para programación de entornos inteligentes basada en diagramas de flujo [7]. Sin embargo, estos proyectos cubren solo aplicaciones y temas de diseño muy específicos.

Al contrario que en los ejemplos anteriores, nuestro sistema abarca todas las fases del proceso de desarrollo con una plataforma flexible, servicios reutilizables y herramientas. La descripción del proceso que se presenta en este artículo es una importante contribución para estandarizar el desarrollo de esta clase de aplicaciones.

6. Resumen

Para la creación de aplicaciones sensibles al contexto para entornos inteligentes se hacen necesarias plataformas novedosas y herramientas que ayuden en el desarrollo de la aplicación. En el nivel básico, nuestro *middleware* MundoCore proporciona la plataforma de integración necesaria para habilitar la comunicación entre los dispositivos hetereogéneos en el entorno. Nuestros sistemas de entornos inteligentes admiten un conjunto de servicios comunes en la capa *middleware*. Las interfaces del servidor de contexto con sensores transforman los datos del sensor en información entendible para las aplicaciones, y proporcionan aplicaciones con interfaces de consulta y de suscripción de eventos. La herramienta *WorldView* admite el modelado y control de entornos inteligentes. La herramienta *SYECAAD* facilita el rápido desarrollo de aplicaciones sensibles al contexto basándose en programación visual. Hemos descrito aquí un proceso de desarrollo de software y como estas herramientas se embeben dentro del proceso, contribuyendo de este modo al desarrollo sistemático de aplicaciones de espacios inteligentes.

Referencias

[1] Andreas Hartl, Erwin Aitenbichler, Gerhard Austaller, Andreas Heinemann, Tobias Limberger, Elmar Braun, Max Mühlhäuser. Engineering Multimedia-Aware Personalized Ubiquitous Services In *IEEE Fourth International Symposium on Multimedia Software Engineering (MSE'02)*, pages 344–351, December 2002.

[2] Erwin Aitenbichler, Jussi Kangasharju, Max Mühlhäuser. MundoCore: A Light-weight Infrastructure for Pervasive Computing *Pervasive and Mobile Computing*, 3(4):332–361, August 2007. doi:10.1016/j.pmcj.2007.04 002.

[3] Marek Meyer. *Context Server: Location Context Support for Ubiquitous Computing*. Master's thesis, Darmstadt University of Technology, January 2005

[4] Jean Schütz. SYECAAD: *Ein System zur einfachen Erzeugung kontextsensitiver Applikationen* Master's thesis, Technische Universität Darmstadt, 2005

[5] Manuel Roman, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, Klara Nahrstedt. A Middleware Infrastructure for Active Spaces. *Pervasive*, 1(4):74–83, October 2002.

[6] David Garlan, Daniel P. Siewiorek, Asim Smailagic, Peter Steenkiste. Project Aura: Toward Distraction-Free Pervasive Computing *Pervasive*, 1(2):22–31, April 2002.

[7] Torben Weis, Marcus Handte, Mirko Knoll, Christian Becker. Customizable Pervasive Applications. In *PerCom 2006*, 2006

[8] Brad Johanson, Armando Fox, Terry Winograd. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. *Pervasive*, 1(2):71–78, April 2002.

[9] Rafael Ballagas, Meredith Ringel, Maureen Stone, Jan Borchers. istuff: a physical user ce toolkit for ubiquitous computing environments In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 537–544, New York, NY, USA, 2003. ACM Press.

[10] Saul Greenberg, Chester Fitchett. Phidgets: easy development of physical interfaces through physical widgets. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 209–218, New York, NY, USA, 2001. ACM Press.

[11] Scott R. Klemmer, Jack Li, James Lin, James A. Landay. Papier-mache: toolkit support for tangible input. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 399–406, New York, NY, USA, 2004. ACM Press.

[12] Peter Tandler. Software infrastructure for ubiquitous computing environments: Supporting synchronous collaboration with heterogeneous devices. In *Ubicomp*, pages 96–115, 2001

[13] Daniel Salber, Anind K. Dey, Gregory D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *CHI*, pages 434–441, 1999.

[14] Ubisense. The Smart Space Company. <http://www.ubisense net/>, 2007. Ultimo acceso: 30.06.2007.

[15] Visonic Technologies. Elpas. <http://www.visonictech.com>, 2007. Ultimo acceso: 30.06.2007.

[16] Khai N. Truong, Elaine M. Huang, Gregory D. Abowd. CAMP: A Magnetic Poetry Interface for End-User Programming of Capture Applications for the Home. In *Ubicomp 2004*, 2004.

[17] Richard Hull, Ben Clayton, Tom Melamed. Rapid Authoring of Mediascapes. In *Ubicomp 2004*, 2004.

Notas

[1] <http://es wikipedia.org/wiki/transpondedor>.

[2] *Local Operating Network*: un sistema de buses para automatización de edificios.