Un framework para la simulación de sistemas pervasivos*

Javier Muñoz, Idoia Ruiz†, Vicente Pelechano, Carlos Cetina

Dept. de Sistemas Informáticos y Computadores

†Instituto Tecnológico de Informática

Universidad Politécnica de Valencia

Campus de Vera

46022 Valencia

{jmunoz, pele, ccetina}@dsic.upv.es, †miruifue@iti.upv.es

Resumen

Los sistemas pervasivos están formados por distintos dispositivos encargados de proporcionar funcionalidad a los usuarios. Durante el desarrollo de sistemas pervasivos es necesario prototipar y probar el sistema final. En muchos casos no se dispone de todos los dispositivos en el momento de la prueba, por lo que proporcionar dispositivos virtuales permitirá ejecutar el sistema en un entorno de simulación. Este trabajo propone un framework para el desarrollo de dispositivos virtuales y la simulación de sistemas pervasivos. El framework, desarrollado utilizando la tecnología OSGi, proporciona: (1) mecanismos para el desarrollo de dispositivos virtuales (que simulen dispositivos reales) y (2) una interfaz gráfica para monitorizar el estado de los dispositivos y modificar la información sobre el entorno que éstos transmiten al sistema. Este framework nos está permitiendo desarrollar y probar aplicaciones pervasivas en el contexto de nuestro método de modelado y generación automática de sistemas pervasivos.

1. Introducción

Los sistemas pervasivos tratan de construir entornos donde los elementos de computación desaparezcan a ojos de los usuarios pero su funcionalidad continúe proporcionándose. Esta visión fue descrita inicialmente por Weiser [9] en los inicios de los años 90 y está basada en la construcción de entornos saturados de computación que se integran adecuadamente con los usuarios humanos. Actualmente podemos encontrar sistemas pervasivos en entornos como los automóviles, oficinas, edificios públicos y, por supuesto, en nuestros hogares. Los requisitos que deben satisfacer este tipo de sistemas abarcan una gran cantidad servicios [8] en áreas como la multimedia, las comunicaciones o la automatización, por lo que estos sistemas utilizan todo tipo de dispositivos hardware para proporcionar estos servicios.

El hecho de utilizar múltiples dispositivos hardware hace que la prueba del software del sistema pervasivo antes de su implantación en el entorno definitivo sea una tarea compleja, porque en muchas situaciones no siempre es posible disponer en el momento adecuado de todos los dispositivos que integran el sistema. Esta limitación se ha hecho patente en el ámbito de nuestra investigación. Nosotros estamos desarrollando un método de producción automática de sistemas pervasivos. Este método promueve el modelado de sistemas pervasivos siguiendo una notación que extiende UML, a partir de especificaciones a alto nivel de abstracción de sistemas pervasivos, se proporcionan mecanismos de transformación que permiten generar automáticamente código Ja-

*Este trabajo ha sido desarrollado con el apoyo del MEC bajo el proyecto DESTINO TIN2004-03534 y ha sido cofinanciado por el programa FEDER y el Programa de Incentivo a la Investigación de la UPV.

Actas del Simposio de Computación Ubicua e Inteligencia Ambiental, UCAmI2005, pp.181-189
ISBN: 84-9732-442-0

va para el entorno OSGi siguiendo una arquitectura predeterminada.

Actualmente se ha desarrollado software para simular dispositivos en redes de comunicaciones, como el AdventNet Simulation Toolkit 5[1], o para simular elementos básicos para la programación de FPGAs (dispositivos electrónicos digitales programables), como VTSim [3]. Por otra parte, UBIWISE [1], desarrollado en HP Labs, es un simulador para computación ubicua centrado en dispositivos de computación y comunicaciones. Este sistema representa los dispositivos en un entorno 3D que se utiliza para simular un entorno real. Se pueden definir regiones de la figura que son enlazadas con funcionalidad programada en Java. Esto implica que debe realizarse un esfuerzo considerable para crear la figura tridimensional del dispositivo, lo cual es un inconveniente serio. Además, los dispositivos para la automatización (sensores y actuadores) no se contemplan. En este ámbito se puede afirmar que actualmente no existe una solución para prototipar el software de un sistema pervasivo sin disponer de los dispositivos que utiliza.

Este artículo presenta un framework para facilitar la simulación de sistemas pervasivos mediante el uso de dispositivos virtuales. El framework ha sido desarrollado sobre la tecnología OSGi y proporciona mecanismos para el desarrollo de dispositivos virtuales, y su monitorización y control mediante una interfaz gráfica de usuario. La estructura del artículo es la siguiente: la sección 2 presenta brevemente el método de producción automática de sistemas pervasivos con el objetivo de proporcionar el contexto adecuado a este trabajo. A continuación, en la sección 3 se describen los objetivos y requisitos del framework que han marcado su diseño e implementación. En la sección 4 se presenta la arquitectura del framework, los distintos elementos que lo componen, sus funciones y cómo se relacionan entre ellos. La sección 5 describe cómo se utiliza el framework para la creación de nuevos tipos de dispositivos e instancias de estos tipos, así como se presentan pautas para el desarrollo de software que haga uso de los dispositivos. Finalmente, la sección 6 presenta las conclusiones y posibles trabajos futuros.

[1] http://www.adventnet.com/products/simulator/

2. Contexto del trabajo

Este trabajo se enmarca dentro de una línea de investigación encaminada a la creación de un método para el desarrollo de sistemas pervasivos basado en las últimas tendencias de la ingeniería del software, como son el estándar Model Driven Architecture (MDA) [6] y la propuesta de las Factorías de Software [2]. La aproximación utilizada, presentada en [5], se basa en la especificación del sistema pervasivo utilizando un lenguaje de modelado de alto nivel de abstracción. Concretamente, se utiliza un lenguaje basado basado en la sintaxis de UML. El lenguaje se basa en el concepto de *servicio* para representar la funcionalidad requerida por los usuarios del sistema pervasivo. Una vez descritos los servicios que debe proporcionar el sistema, se detallan los dispositivos o sistemas software externos que implementarán esos servicios. Entonces esta especificación es automáticamente transformada en el código fuente que, una vez compilado, produce el sistema pervasivo.

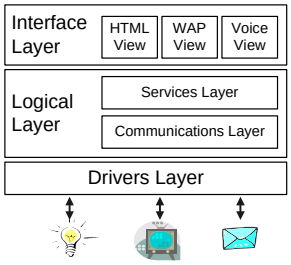

Figura 1: Arquitectura de los sistemas pervasivos generados por nuestro método.

Siguiendo las guías propuestas por las Factorías de Software, se ha definido una arquitectura que compartirán todos los sistemas generados, así como una infraestructura software de soporte a esta arquitectura. La arquitectura, que se muestra en la Figura 1, está estructurada en capas. La capa superior (*Interfaz Layer*) contiene la interfaz hacia los usuarios

que, en este tipo de sistemas, debe proporcionarse mediante diversos mecanismos (dispositivos móviles, reconocimiento de voz, acceso remoto desde Internet, etc.). La capa intermedia (*Logical Layer*) contiene la funcionalidad del sistema, que en nuestros sistemas se trata de una representación de los servicios requeridos, los cuales harán uso de dispositivos o sistemas software externos para proporcionar la funcionalidad del sistema. Finalmente, en la capa inferior (*Drivers Layer*) se encuentran los elementos software encargados de comunicarse con los dispositivos que utilizará el sistema. Estos elementos deben gestionar los mecanismos de programación (librerías de acceso, etc.) de cada tecnología y/o frabricante, por lo que deben desarrollarse específicamente para cada dispositivo.

Esta arquitectura está soportada por una infraestructura software implementada utilizando la tecnología OSGi [4]. OSGi es un *middleware* en Java que proporciona una serie de características y servicios útiles para el desarrollo de sistemas pervasivos, como actualización dinámica de componentes, desacople de interfaces e implementación y gestión del ciclo de vida de los servicios instalados. Además, tiene disponibles adaptadores para acceder a otras tecnologías utilizadas en éste ámbito, como EIB, UPnP o LonWorks. Por todo ello, fue elegido como plataforma de implementación de nuestros sistemas.

3. Objetivos y requisitos del framework

El objetivo principal que pretende alcanzar este trabajo es *poder ejecutar sistemas pervasivos sin disponer de los dispositivos que éste utilizará en la implantación definitiva, pero sin disminuir el número de escenarios posibles en los que se puede encontrar el sistema*. Para alcanzar este objetivo se ha considerado como una opción adecuada el desarrollo de una serie de *drivers* de dispositivos virtuales, que puedan ser utilizados por el sistema en lugar de los reales, en las situaciones en la que no se tenga disponibilidad de ellos y/o en etapas de prototipación. Para facilitar esta tarea se ha creado un framework [2] con los siguientes requisitos:

- debe ser posible reemplazar los *drivers* virtuales por los mecanismos utilizados para acceder a los dispositivos reales sin que ello implique modificaciones en el resto del sistema. De este modo se consigue que el sistema en prueba va a ser el mismo que se utilizará en la implantación final.
- debe disponer de una interfaz de usuario para monitorizar el estado de los dispositivos, observar cómo utiliza el sistema los dispositivos (las operaciones que se invocan) y modificar los valores que éstos suministran al sistema como información del entorno (temperatura, luminosidad, presencia, etc.).
- el acceso a los dispositivos debe organizarse atendiendo a su ubicación, ya que este criterio resulta intuitivo en este contexto (todo dispositivo físico está ubicado en algún lugar).
- la creación de nuevos tipos de dispositivos virtuales debe ser sencilla para que el desarrollo de dispositivos virtuales no resulte más costoso que la adquisición de los dispositivos reales.
- debe ser independiente del método de desarrollo utilizado para la implementación de sistemas pervasivos para facilitar su difusión y para continuar siendo útil en caso de evoluciones del método.

Las siguientes secciones describen el framework implementado que satisface estos requisitos, así como el modo en que éste debe utilizase.

4. Estructura del framework

El framework para la simulación de sistemas pervasivos está estructurado en tres partes bien diferenciadas: la infraestructura de clases abstractas para la creación de dispositivos

[2] Un framework es un diseño reutilizable de todo o parte de un sistema que está representado por un conjunto de clases abstractas y por la forma en que interactúan sus instancias.

virtuales, un catálogo de dispositivos virtuales creados utilizando la infraestructura anterior, y una interfaz de usuario para poder interactuar con los dispositivos virtuales cuando estos se encuentran en ejecución

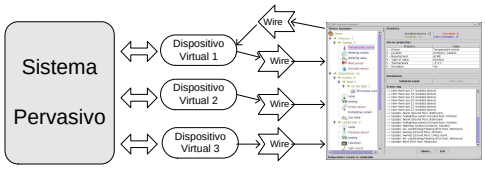

Figura 2: Estructuración del framework

Antes de pasar a describir cada una de estas partes, vamos a presentar el esquema global de funcionamiento que sigue el framework. Tal y como se presenta en la Figura 2, los dispositivos virtuales serán utilizados por el sistema pervasivo, pero no se impone ninguna restricción sobre cómo se realizará esta comunicación. De este modo se consigue que nuestro framework sea independiente del método utilizado para desarrollar el sistema pervasivo. Por otra parte, la comunicación entre los dispositivos y la interfaz de usuario del simulador se realiza utilizando los mecanismos de comunicación que proporciona OSGi llamados *Wires*. A través de este mecanismo se envían paquetes de propiedades (*HashMap*) en ambas direcciones. De este modo se facilita un alto desacoplamiento entre los dispositivos y la interfaz, ya que no necesitan referencias mutuas explícitas. A continuación se describen detalladamente los componentes del framework.

4.1. Infraestructura para el desarrollo de dispositivos virtuales

La infraestructura que proporciona el framework para el desarrollo de dispositivos, que se muestra en la Figura 3 se compone de: clases abstractas para la creación de dispositivos y una clase auxiliar para la gestión de estos dispositivos. El framework proporciona dos clases abstractas llamadas *VirtualDevice* y *VirtualManagedDevice*. Todos los dispositivos virtuales deben extender una de estas dos clases.

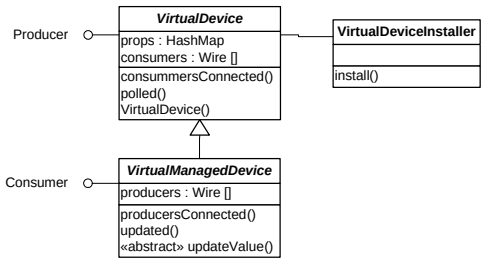

Figura 3: Clases de la infraestructura software del framework.

La clase abstracta *VirtualDevice* se encarga de:

- Definir un atributo llamado `props` de tipo `HashMap` para almacenar las propiedades del dispositivo.
- Definir una serie de constantes que serán utilizadas para referenciar a las propiedades. Por ejemplo, se definen propiedades para especificar el tipo de dispositivo, la ubicación, el fabricante, el tipo de valor que almacenará la propiedad, etc. En la sección 5 se comentarán detalladamente estas propiedades.
- Definir un constructor por defecto, el cual recibe como parámetro la ubicación del dispositivo. Las clases que extiendan de *VirtualDevice* deberán sobrescribir este constructor.
- Implementar la interfaz que le permite participar como emisor de información (*Producer*) en el mecanismo de comunicaciones de OSGi.

Por su parte, la clase *VirtualManagedDevice* es una especialización de la clase anterior. Esta clase está ideada para aquellos dispositivos sensores que reciben información del entorno, de manera que los valores de sus propiedades podrán fijarse desde la interfaz del simulador. Además de lo especificado para la clase *VirtualDevice*, la clase *VirtualManagedDevice* se encarga de:

- Implementar la interfaz que le permite participar como receptor de información

(*Consumer*) en el mecanismo de comunicaciones de OSGi.

- Definir la operación abstracta `updateValue` que deberá ser implementada en la clases concretas para modificar adecuadamente el estado del dispositivo a partir de los valores proporcionados por el usuario.

En la sección 5 se describe cómo deben extenderse estas clases para crear nuevos tipos de dispositivos.

Finalmente, la clase estática *VirtualDeviceInstaller* se encarga de realizar las acciones necesarias para instalar adecuadamente un nuevo dispositivo virtual en el entorno. Esta clase tiene un método llamado *install* que recibe como parámetros una instancia del dispositivo que se desea instalar y un nombre, a partir de los cuales:

- registra al dispositivo virtual como generador (y productor, si es necesario) de información.
- crea canales de comunicación (*Wire*) entre el dispositivo y la interfaz de usuario (un canal para enviar información y otro, si es necesario, para recibir información de la interfaz).

4.2. Interfaz de usuario

La Figura. 4 muestra la interfaz de usuario que permite el acceso a los dispositivos virtuales desarrollados con nuestro framework. Está dividida en cinco paneles que se describen a continuación:

El *Panel 1* está formado por el *árbol de dispositivos*, el cual se utiliza como mecanismo de selección de los dispositivos virtuales. Este árbol se construye a partir de la propiedad *LOCATION* que deben proporcionar todos los dispositivos. Esta propiedad almacena una cadena de texto, donde cada ubicación de la jerarquía está separada por el símbolo "/". Para poder distinguir dos dispositivos del mismo tipo en una misma ubicación, la cadena opcionalmente puede finalizar con un contador de tres dígitos separado por el símbolo ":". Por ejemplo, una posible ubicación sería "/Home/First Floor/Corridor:001". Destacar que la construcción del árbol se realiza de manera dinámica a partir de los valores de los dispositivos registrados en un momento dado.

La ruta de los iconos mostrados en el árbol se almacena en la propiedad *IMAGE* de los dispositivos. El nombre de los dispositivos puede ser de color rojo o azul. En caso de que el nombre sea de color rojo alguna propiedad de ese dispositivo se podrá modificar desde la interfaz. Si el nombre es de color azul, todos los valores serán de lectura.

El *Panel 2* muestra una serie de estadísticas sobre el sistema. En concreto muestra el número de dispositivos instalados, la cantidad de éstos que son modificables y los que no, y el número de ubicaciones definidas en nuestro sistema.

El *Panel 3* muestra al usuario las propiedades que tiene el dispositivo, y los valores de las mismas. Cada tipo de dispositivo puede tener propiedades distintas, pero algunas de ellas son obligatorias. Se ofrecerán más detalles sobre este aspecto en la sección 5 al describir la construcción nuevos tipos de dispositivo.

El *Panel 4* contiene los botones que permiten editar los valores de aquellos dispositivos que tienen propiedades modificables. Según el tipo de datos que almacena el valor modificable se podrán realizar dos acciones: (1) simular la entrada, para lo cual se introducirá el valor directamente en una caja de texto, o (2) conmutar el valor actual, en caso de que sólo pueda almacenar dos valores.

Finalmente, el *Panel 5* contiene una lista con los eventos que los dispositivos han notificado a la interfaz de usuario. Es responsabilidad de los dispositivos notificar de aquellos eventos que consideren relevantes para el usuario del simulador.

4.3. Catálogo de dispositivos

Haciendo uso de la infraestructura descrita en la subsección 4.1 se han implementado una serie de dispositivos básicos a modo de ejemplo. Básicamente se ha intentado implementar al menos un elemento de cada tipo posible, teniendo en cuenta que los dispositivos pueden

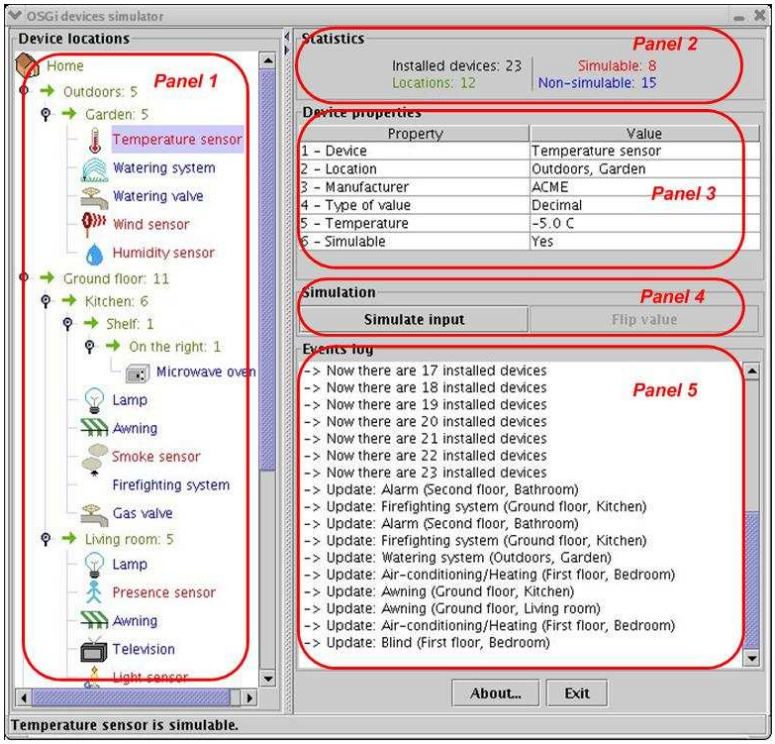

Figura 4: Interfaz de usuario del framework de simulación

	Booleano	**Natural**	**Multiple**
Modif.	Sens. Presencia	Sens. Luminosidad	Estación Metereológica
No Modif.	Bombilla	Toldo	Televisor

Tabla 1: Dispositivos básicos

ser modificables (sensores) o no modificables (efectores) y que las propiedades que trata cada uno de ellos puede ser de varios tipos (booleano, número natural, una mezcla de los anteriores, etc.).

La Tabla 1 muestra los dispositivos básicos que se han desarrollado. A partir de éstos pueden crearse de manera sencilla muchos dispositivos similares. Por ejemplo un sensor de humo será similar a un sensor de presencia, o un actuador para un sistema de riego será similar a una bombilla, ya que ambos tendrán funciones sencillas para encender y apagar.

5. Uso del framework

Esta sección describe como realizar dos tareas importantes para hacer uso del framework: la creación de nuevos tipos de dispositivos y la creación de instancias de esos tipos en el sistema.

5.1. Creación de nuevos tipos de dispositivos

Para la creación un nuevo tipo de dispositivo virtual se debe crear una nueva clase Java, siguiendo el siguiente procedimiento:

1. En primer lugar hay que decidir si el nuevo tipo de dispositivo tendrá alguna propiedad que ha de ser controlada desde la interfaz de usuario. Si es así la nueva clase heredará de la clase abstracta *VirtualManagedDevice*. En caso contrario deberá heredar de la clase *VirtualDevice*. Independientemente de esto, la clase Java podrá implementar tantas interfaces como sean necesarias, ya que esto no entrará en conflicto con su funcionamiento como dispositivo simulado

2. El *constructor de la clase* deberá: (1) llamar al constructor de la clase padre (`this.super()`) utilizando como argumento la cadena de texto con la ubicación del dispositivo virtual, (2) insertar en la variable `props` (que es de tipo `HashMap`) las propiedades necesarias, y (3) realizar otras taréas no relacionadas con la simulación del dispositivo. Respecto a las propiedades a insertar en la variable `props` cabe destacar que algunas de ellas son obligatorias para el correcto funcionamiento del mecanismo de simulación de dispositivos:

   - *DEVICE*: almacena el nombre del dispositivo.
   - *LOCATION*: almacena la ubicación del dispositivo. Esta propiedad es añadida por el constructor de la clase abstracta.
   - *VALUETYPE*: almacena el tipo del valor que gestiona el dispositivo. Los posibles valores de esta propiedad son *NUMERIC*, *BOOLEAN*, *STRING* o *MULTIPLE*. La interfaz del simulador utiliza esta propiedad para proporcionar el mecanismo más adecuado para editar el valor.
   - *EDITABLE*: indica si el valor que gestiona el dispositivo puede ser modificado por el usuario (si se trata de algún tipo de sensor).
   - *IMAGE*: contiene la ruta del icono que se debe utilizar en el árbol de dispositivos para representar a este dispositivo

   El resto de propiedades son mostradas al usuario sin que tengan ninguna implicación en el funcionamiento del sistema.

3. En cualquier momento de la ejecución de la clase que representa el dispositivo simulado es posible realizar la *comunicación con el simulador* para informar de alguna incidencia. Para ello será necesario seguir los siguientes pasos: (1) crear una variable de tipo `HashMap`, (2) insertar las propiedades cuyo valor deba ser modificado y (3) transmitir el diccionario (para lo cual será necesario invocar el método `update` de los canales de comunicación: `consumers[i].update(h)`). Además, el simulador proporciona la capacidad de enviar mensajes al usuario. Para ello será necesario incluir en el diccionario una propiedad llamada *MESSAGE*. El contenido de esta variable será mostrado al usuario en una ventaja emergente.

4. En el caso de que exista alguna propiedad que pueda ser modificada (en cuyo caso se habrá extendido la clase `VirtualManagedDevice`), será necesario implementar el método `updateValue`. Este método recibe como parámetro una cadena de texto que deberá ser convertida al tipo de datos adecuado antes de almacenar su valor y/o realizar las acciones oportunas.

5. Finalmente, se implementará el resto de la clase siguiendo las directrices que dicte el sistema pervasivo con el que interactuará el dispositivo virtual.

5.2. Creación de una nueva instancia de un tipo de dispositivo

La creación de una nueva instancia de dispositivo se debe realizar desde una clase Java.

Un modo habitual de proceder, es utilizar una clase para instalar todos los dispositivos que necesitará utilizar un sistema. Para instalar un dispositivo se deben realizar los siguientes pasos:

1. Crear una nueva instancia de la clase Java que implementa el dispositivo virtual que queremos instalar. Será necesario crear tantas instancias como dispositivos virtuales deseen instalarse en el sistema.

2. Registrar adecuadamente el dispositivo en el entorno. Para ello se utilizará el método `install` de la clase `VirtualDeviceInstaller`. Este método recibe como parámetros la instancia creada en el paso anterior, un nombre y un identificador único en el entorno.

El siguiente trozo de código muestra un ejemplo de los pasos descritos:

```
VirtualDeviceInstaller devInst =
       new VirtualDeviceInstaller();
LampDevice lamp001 =
       new LampDevice("/Home/Corridor:001");
devInst.install(lamp001,
                "Corridor Lamp",
                "org.oomethod.lamp001");
```

6. Conclusiones y trabajos futuros

En este artículo hemos presentado un framework para la simulación de sistemas pervasivos. El framework, implementado sobre el entorno OSGi, se basa en la construcción de dispositivos virtuales que permitan la ejecución de un sistema pervasivo sin contar con los dispositivos físicos que éste utilizará una vez implantado. Este framework está siendo utilizado en el contexto de un método de generación automática de sistemas pervasivos a partir de modelos conceptuales. El método sigue la filosofía de OO-Method [7], un método orientado al desarrollo de sistemas de información que ha sido implementado industrialmente en la herramienta OlivaNoval Model Execution System[3]. El framework de simulación se ha desarrollado de forma independiente al método de desarrollo y a la arquitectura utilizada para implementar los sistemas pervasivos.

Actualmente estamos trabajando en la ampliación del catálogo de dispositivos, que ahora cuenta con 13 tipos de dispositivos. También se está extendiendo la interfaz del sistema de simulación para permitir la creación dinámica de instancias de los dispositivos.

[3] http://www.care-t.com/

Referencias

[1] John J. Barton and Vikram Vijayaraghavan. UBIWISE, A Ubiquitous Wireless Infrastructure Simulation Environment. Technical report, HP Labs, 2003.

[2] Jack Greenfield, Keith Short, Steve Cook, and Stuart Kent. *Software Factories.* Wiley Publising Inc., 2004.

[3] Jesse Hunter, Peter Athanas, and Cameron Patterson. VTsim: A Virtex-II Device Simulator. In *2004 International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA 2004*, Las Vegas, Jun 2004.

[4] Dave Marples and Peter Kriens. The Open Services Gateway Initiative: An Introductory Overview. *IEEE Communications Magazine*, 39(12):110–114, 2001.

[5] Javier Muñoz and Vicente Pelechano. Building a Software Factory for Pervasive Systems Development. In João Falcão e Cunha Oscar Pastor, editor, *Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17*, volume 3520 of *Lecture Notes in Computer Science*, pages 329–343. Springer-Verlag GmbH, May 2005.

[6] Object Management Group. Model Driven Architecture Guide, 2003.

[7] Oscar Pastor, Jaime Gómez, Emilio Insfrán, and Vicente Pelechano. The OO-Method Approach for Information Systems

Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. *Information Systems*, 26(7):507–534, 2001.

[8] Roy Want, Trevor Pering, Gaetano Borriello, and Keith I. Farkas. Disapearing Hardware. *Pervasive Computing*, 1(1), 2002.

[9] Mark Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, Sept. 1991.