Universidad Autónoma de Madrid

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

TESIS DOCTORAL

TRATAMIENTO DE INFORMACIÓN CONTEXTUAL EN ENTORNOS INTELIGENTES

Pablo A. Haya Coll

Madrid, Marzo 2006

Escuela Politécnica Superior

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

TESIS DOCTORAL

TRATAMIENTO DE INFORMACIÓN CONTEXTUAL EN ENTORNOS INTELIGENTES

Autor:
Pablo A. Haya Coll

Director:
D. Xavier Alamán Roldán

Madrid, Marzo 2006

TÍTULO:	Tratamiento de información contextual en entornos inteligentes
AUTOR:	Pablo A. Haya Coll
DIRECTOR:	D. Xavier Alamán Roldán
DEPARTAMENTO:	Ingeniería Informática

MIEMBROS DEL TRIBUNAL:

PRESIDENTE:

SECRETARIO:

VOCAL 1:

VOCAL 2:

VOCAL 3:

FECHA DE LECTURA:

CALIFICACIÓN:

RESUMEN:

A finales de los años 80 surge la *Computación Ubicua* cuando Mark Weiser propone trasladar la capacidad de computación de los rígidos y voluminosos ordenadores personales a miles de dispositivos diseminados por el entorno, de forma que las computadoras se fundan con el entorno hasta volverse invisibles al usuario. En la *Computación Ubicua* la interacción persona-ordenador se expande a todo el espacio, dando lugar a entornos capaces de adquirir información de forma autónoma y de emplearla para adaptarse a las necesidades de sus ocupantes. Esta interacción entre el entorno y el usuario se ve beneficiada al considerar información contextual como puede ser la localización, la tarea que está realizando el usuario, otros recursos que se encuentren cerca, las condiciones ambientales del entorno, etc. Este conocimiento implícito de la situación hace posible que el entorno responda con cierto grado de proactividad, de forma que se libera la atención del usuario cuando no sea imprescindible.

La combinación de la información contextual con los *entornos inteligentes* conlleva una serie de complicaciones que dificultan el desarrollo de *aplicaciones sensibles al contexto*. Se han de integrar y gestionar una mezcla heterogénea de tecnologías que permitan capturar, distribuir y modificar el contexto teniendo en cuenta que esta información contextual proviene de fuentes de diversa naturaleza, y que se puede describir en diferentes niveles de abstracción. A esto hay que añadir que la configuración del entorno es dinámica, de manera que pueden aparecer y desaparecer nuevos componentes del entorno. El resultado es que el desarrollador de *aplicaciones sensibles al contexto* se encuentra ante una casuística difícil de manejar debido al elevado número de posibles configuraciones.

La propuesta de esta tesis radica en un capa de contexto que sirve como pegamento para conseguir la sinergia necesaria entre los elementos ubicuos que constituyen el *entorno inteligente*. Ésta aporta: (a) una representación estándar del contexto que es independiente de su naturaleza y nivel de abstracción. Se provee de un modelo unificado del mundo reduciendo la complejidad en el acceso a la información contextual. (b) Un mecanismo flexible para poder almacenar y distribuir esa información que facilita la configuración y reutilización de los distintos componentes del entorno.

La implementación de la capa de contexto reside en una estructura de datos global denominada pizarra. Esta pizarra constituye un modelo del mundo, donde se encuentra almacenado todo el contexto generado por los componentes del entorno y por el mismo entorno. La pizarra almacena la información en forma de grafo, donde cada vértice representa una entidad, como puede ser un usuario, una aplicación, un dispositivo, una habitación, etc. El contexto se representa mediante un lenguaje común, de tal forma que conviven en la pizarra información procedente de los sensores —como el estado de los dispositivos físicos— con contexto deducido a partir de la información sensorial, como por ejemplo el número de personas que hay en el entorno, o la tarea que está realizando una persona en cada momento.

Gracias a esta arquitectura se consigue un acoplamiento débil entre los diversos componentes en tres niveles: temporal, espacial y funcional. Este anonimato

permite que la conexión o desconexión de un componente en el entorno se realice de forma transparente al resto.

La capa de contexto que se propone ha sido probada en un *entorno inteligente* real equipado como un salón de una casa convencional y, en parte, como una oficina. Este prototipo incluye dispositivos de distintas tecnologías, varias *aplicaciones sensibles al contexto* y dos interfaces de usuario de distinta de naturaleza que se generan automáticamente a partir del modelo del entorno almacenado en la pizarra.

ABSTRACT:

Ubiquitous Computing arised during the late eighties with Mark Weiser's proposal of bringing computing capabilities from the rigid and voluminous personal computer to thousand of devices scattered throughout the physical environment. Computers would seamlessly integrate with the environment, becoming invisible to users. In the field of *Ubiquitous Computing*, human-computer interactions is expanded to the space around the user. The environment is able to acquire information by itself and use it to adapt to its occupant's requirements. This interaction between the environment and the user is enhanced when contextual information —such as location, user's activity, nearby resources, environmental variables, etc.— is taken into account. This implicit information about the current situation allows the environment to respond with certain degree of proactivity, so that the users' attention need not be constantly dedicated to interaction.

The combination of contextual information and *intelligent environments* entails several issues that complicate the development of *context-aware applications.* A heterogeneous mixture of technologies that capture, distribute and modify the context must be integrated and managed. Context information can be obtained from different sources and can be described at multiple levels of abstraction. Besides, the environment configuration can change dynamically, when environment components are attached or detached at run time. The result is that the developer of context-aware applications is confronted with a huge space of possible configurations.

This thesis presents a proposal for a context-based architecture that achieves the required synergy among the ubiquitous component of an intelligent environment. The proposal contribution is summarized as follows: (a) a model of context information that is independent of the nature of the source, and capable of combining different degrees of abstraction. (b) a flexible communication mechanism that allows storage and retrieval of context information. This mechanism facilitates reusability and configuration of the different components of the environment.

The context layer implementation relies on a global data structure, the blackboard. This blackboard stores a model of the world, which includes all the prominent information related to the environment and its components. The model is maintained as a graph, where each vertix is an entity that can represent a user, a device, a room, etc. The context is defined by means of a common language, so that information from different sources coexists in the blackboard. Data sources include sensors —such as physical device status— and context derived from these sensors, such as the number of persons to be found in the environment or inference on their current activities.

This architecture achieves a loose coupling between the different components, in three levels: temporal, spatial and functional. This anonymity is necessary to allow transparent insertion and removal of components.

The proposed context layer has been tested in a real-world *intelligent environment*, outfitted as a conventional living-room and, partly, as an office. This prototype includes components of various technologies, multiple context-aware applications, and two user interfaces with different modalities, generated automa-

tically from the model of the environment stored in the blackboard.

PALABRAS CLAVE:

Contexto, *entornos inteligentes, entornos activos, Computación Ubicua, aplicaciones sensibles al contexto, computación basada en la localización*

A mis padres y a mi hermano.

A Javier Martínez.

Índice general

Índice de figuras

Índice de tablas

PARTE I

Introducción

Capítulo 1

Introducción

1.1. Introducción

Como no podía ser de otra manera, una tesis que versa sobre el contexto tiene que introducirse refiriéndose al contexto en el que fue realizada. Esta tesis nace en el año 2000 como parte del proyecto Odisea [1] [13]. Este proyecto surge dentro del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid con vocación multidisciplinar. En él han participado, en mayor o menor medida, profesores, doctorandos y becarios del Departamento pertenecientes a distintas áreas de investigación. Así, han confluido expertos en redes de ordenadores, interacción hombre-máquina, visión artificial y dispositivos electrónicos. El objetivo del proyecto es la exploración de las tecnologías necesarias para la implementación de *entornos activos*, en los cuales la interacción persona-máquina se realiza de manera natural y sensible al contexto dentro de cual el usuario está ejecutando una tarea. Un ejemplo de *entorno activo* puede ser una vivienda en la cual se pueda interaccionar en lenguaje natural con los diferentes electrodomésticos y otros dispositivos (luces, teléfonos, cadenas de música, etc.), y donde el sistema de interacción es multimodal y sensible al contexto: a las personas que ocupan cada habitación, a sus gustos y preferencias y a la tarea que están realizando en cada momento. El proyecto Odisea evolucionó al proyecto *Interact*[2] [15] financiado por el Ministerio de Ciencia y Tecnología TIC2000-0464, y ha dado como resultado un laboratorio de pruebas donde se ha implementado una habitación inteligente sensible al contexto capaz de interactuar con sus inquilinos ya sea mediante lenguaje natural o a través de Internet, mediante cualquier navegador convencional.

Tal como se deduce del título[3] el planteamiento de esta tesis se centra en estudiar la manera más eficaz de gestionar el contexto dentro de un entorno inteligente. Ahora bien, el término información contextual tiene múltiples acepciones que dependen del dominio de aplicación. Incluso si se restringe éste al campo de las ciencias de la computación, todavía el espectro de posibles definiciones es muy amplio. En la presente tesis se considera el contexto en el mismo sentido que le

[1] Ofimática y Domótica Inteligentes Soportadas mediante Entornos Activos
[2] Interfaces de Usuario Multimodales: Entornos Activos
[3] *Tratamiento de información contextual en entornos inteligentes*

dan los trabajos realizados en *aplicaciones sensibles al contexto*. Esta área engloba aplicaciones y dispositivos que consideran como una entrada más del sistema la información sobre las circunstancias bajo las cuales operan. Estas circunstancias pueden ser la localización, la tarea que está realizando el usuario, otros recursos que se encuentren cerca, las condiciones ambientales del entorno, etc. La definición y representación del contexto es un punto central de esta tesis que se tratará en profundidad en el capítulo 4.

El alcance del estudio queda acotado a las *aplicaciones sensibles al contexto* que operan dentro de un *entorno inteligente*. Éste consiste en una infraestructura restringida por unas barreras físicas y compartida por un conjunto de aplicaciones, dispositivos y personas. El adjetivo *inteligente* se emplea para indicar la habilidad de adquirir información de forma autónoma y de emplearla para adaptarse a las necesidades de sus ocupantes. El entorno se convierte en una *aplicación sensible al contexto* más, contribuyendo activamente en la interacción con el usuario. Así, en la literatura también se pueden encontrar referencias a los *entornos inteligentes* como *entornos activos*[4], o espacios activos[5]. En la vida diaria una persona desarrolla sus actividad en múltiples tipos de entornos. Un hogar, una oficina, una clase, un vehículo o un restaurante son ejemplos posibles de *entornos activos*. La propuesta de esta tesis se ha realizado eligiendo como entorno de estudio el salón de una casa convencional, y en cierta medida también sobre una oficina. Gran parte de las conclusiones que se han extraído son extrapolables a otros entornos, aunque se han tomado ciertas consideraciones particulares al hogar, que se irán indicando a medida que vayan surgiendo en el documento.

Para entender mejor la importancia del contexto y su relación con los *entornos inteligentes* es necesario referirse a la *Computación Ubicua*, una tercera área de investigación que engloba a los dos temas centrales de la tesis. La *Computación Ubicua* [6], también conocida como *Computación Pervasiva*[7], nació a finales de la década de los 80 como un nueva rama de la computación móvil y de los sistemas distribuidos. Desde entonces ha ofrecido nuevas oportunidades y desafíos dentro del campo de las ciencias de la computación [233]. El término *Computación Ubicua* fue definido por primera vez por Mark Weiser [287], el cual propone trasladar la capacidad de computación de los rígidos y voluminosos ordenadores personales a miles de dispositivos diseminados por el entorno, de forma que las computadoras se fundan con el entorno hasta volverse invisibles al usuario.

Una de las frases recurrentes que se emplea para describir la *Computación Ubicua* es que ésta proveerá de capacidades de computación en cualquier momento y en cualquier sitio[8]. En cierto sentido el efecto que se persigue se puede equiparar al avance que ha supuesto la telefonía móvil frente a la telefonía fija en la década de los noventa. La telefonía fija provee un servicio que permite al usuario llamar en cualquier momento. Por el contrario, el acceso al servicio queda restringido espacialmente ya que requiere de la disponibilidad de un terminal fijo cercano. Con

[4] *Active Environments*
[5] *Active Spaces*
[6] *Ubiquitous Computing*
[7] *Pervasive Computing*
[8] *anytime, anywhere*

la aparición de la telefonía móvil el servicio telefónico ha dejado de ser dependiente de la localización. La implantación de esta nueva tecnología ha conseguido que la comunicación telefónica sea *ubicua* en nuestra vida diaria, hasta el punto de haber modificado nuestros hábitos y creado nuevas normas sociales.

Este grado de imbricación es el objetivo final de la transformación en el uso de los ordenadores que plantea la *Computación Ubicua*, tal como lo expresa Weiser en [286]:

> La tecnologías más profundas son aquellas que desaparecen. Se imbrican en el tejido de la vida diaria hasta que son indistinguibles de ella.

El reto de la *Computación Ubicua* precisa de una revolución en dos ejes: por un lado se precisa de un proceso de distribución de nuevos dispositivos y un despliegue de infraestructuras que garantice la ubicuidad de las capacidades de computación. Se requieren nuevos dispositivos, más portables, con capacidades inalámbricas y bajo consumo que permitan mayor flexibilidad y movilidad al usuario, así como la implantación de nuevas infraestructuras que soporten los nuevos dispositivos integrados en el entorno (una habitación, una oficina, un edificio...) y que permitan mayor fiabilidad y conectividad cuando sea posible. En este eje se han conseguido grandes avances gracias a la computación móvil.

El otro eje en el cual se espera una importante revolución es en el *software*. Las aplicaciones que se pueden desarrollar con un ordenador son potencialmente infinitas, y es en este sector donde se va a producir la gran revolución. En este sentido el cambio que preconiza la *Computación Ubicua* tiene profundas implicaciones a la hora de concebir los nuevos sistemas informáticos. La nuevas tecnologías que se propone desplegar no deberían suponer una distracción para el usuario. Actualmente, el modelo de interacción que soporta el ordenador personal requiere de toda la atención del usuario. El uso se encuentra restringido tanto por las características físicas del ordenador como por la metáfora de interacción que emplean los entornos de ventana. Para el usuario neófito constituye el principal obstáculo que encuentra a la hora de acercarse al ordenador. Éste se ve obligado a gastar una gran cantidad de esfuerzo en el periodo de adaptación, siendo especialmente crítico este aprendizaje en personas mayores. La creación de nuevos terminales más compactos y portables supera una de las limitaciones que impone el ordenador personal. El usuario puede transportar consigo las aplicaciones tradicionales, o al menos una versión reducida, pudiendo disponer de la aplicación en cualquier momento. Pero esta solución por si sola es insuficiente, ya que estos nuevos dispositivos imponen mayores restricciones en la interacción que suponen para el usuario un coste elevado en el uso y el aprendizaje. Este esfuerzo tampoco se ve reducido en la curva de aprendizaje, que en el caso del ordenador personal es elevada para usuarios que no estén familiarizados con las tecnologías de la información, y que en el caso de la computación móvil sigue teniendo la misma pendiente, incluso podría aventurarse que más pronunciada. En este sentido, el despliegue de la *Computación Ubicua* tiene que venir guiado por una nueva forma de entender la interacción con los ordenadores. Una nueva forma en la que la tecnología inter-

acciona con el usuario de forma transparente [289], sin que suponga un trastorno para su vida diaria.

Dicho según las palabras de Donald Norman [196]:

> Necesitamos movernos hacia la tercera generación de ordenadores personales, la generación donde la máquina desaparece de la vista, donde podemos volver a concentrarnos en las actividades y objetivos de nuestra vida.

La transformación que propone la *Computación Ubicua* en las aplicaciones informáticas se puede resumir en tres objetivos:

- **Reconocer el contexto.** Uno de los puntos débiles de las aplicaciones de escritorio es la insensibilidad ante los cambios del contexto del usuario. Mientras que se ha hecho un considerable avance en la modelización del usuario y la adaptación según diferentes perfiles y preferencias, las aplicaciones tradicionales muestran el mismo comportamiento independientemente de la situación en que se encuentre el usuario. El contexto se ha reconocido como una parte fundamental de la comunicación humana [69], por lo que debe ser incorporado también el diseño de los sistemas informáticos para acercarlos a nuestros códigos de comunicación. La localización, la actividad o el foco de atención del usuario, entre otras variables contextuales, tienen que formar parte de las nuevas aplicaciones ubicuas.

- **Actuar proactivamente.** El diálogo entre la aplicación y el usuario puede ser iniciado o bien por el usuario, o bien por la aplicación, o bien una mezcla de ambos. En general, la mayor parte de las aplicaciones de escritorio son pasivas, ya que es el usuario el que debe tomar la iniciativa. Un campo que ha prestado especial interés por la proactividad son los agentes personales. Éstos se encargan de buscar y mostrar información según la preferencias del usuario pero sin necesitar supervisión explícita. En el marco de la *Computación Ubicua* es preciso introducir en el diseño de las aplicaciones cierto grado de proactividad que permita liberar la atención del usuario cuando no sea imprescindible. Para ello se hace necesario realizar modelos del comportamiento humano [198] que permitan inferir las necesidades del usuario.

- **Aplicaciones ubicuas.** Actualmente para que una aplicación sea *ubicua* es necesario instalarla en cada uno de los dispositivos donde se quiere emplear, teniendo en cuenta que para cada dispositivo se cargará una versión distinta que se ajuste a las restricciones que éste imponga. Esto incrementa la complejidad en el desarrollo, mantenimiento e interoperatibilidad de las diferentes versiones de la aplicación. Una aproximación para solucionar este problema es mantener una funcionalidad única y generar la interfaz dinámicamente adaptándose al dispositivo que requiera el usuario en cada momento [205]. En este sentido se están realizando importantes esfuerzos en estandarizar lenguajes de representación de interfaces de usuario abstractas como

UIML [16] o XIML [214], de modo que se pueda definir una interacción genérica independiente del modo a emplear y ligarla con la funcionalidad dinámicamente.

La consecución de estos objetivos recae sobre tres tecnologías: la computación ubicua, las aplicaciones sensibles al contexto y los entornos inteligentes. Recientemente, a la combinación de estas tres áreas se le ha denominado *Inteligencia Ambiental*[9]. Ésta se refiere a la presencia de un entorno digital que es interactivo, sensible, y adaptativo a la presencia de sus ocupantes [1]. La *Inteligencia Ambiental* pretende fundir los conceptos de ubicuidad y transparencia en un diseño *centrado en el usuario* que dé como resultado un verdadero *ordenador invisible*. La *Inteligencia Ambiental* promueve el diseño centrado en el usuario y el uso de sistemas altamente interactivos. El objetivo final es conseguir que las tecnologías previamente mencionadas ayuden de manera no invasiva en el quehacer diario de las personas.

Actualmente, la *Inteligencia Ambiental* está recibiendo importantes apoyos institucionales, como demuestra su inclusión por parte de la Unión Europea como uno de los temas prioritarios del VI Programa Marco [148]. También en el ámbito industrial se pueden encontrar varias iniciativas por parte de empresas y consorcios de prestigio internacional, como Philips [211], IBM [285] o HP [146].

1.2. Motivación

Un entorno inteligente se caracteriza por ser una mezcla heterogénea de numerosos componentes *software* y *hardware* [133]. Esta mezcla implica ciertos desafíos:

- **Se han de integrar y gestionar distintos tipos de tecnología**, con el consecuente aumento de complejidad en el desarrollo. El usuario puede utilizar múltiples modos (habla, gestos, tacto, etc.) para interaccionar con el entorno, con lo que se requieren interfaces de usuario muy variadas. Por otro lado la distribución de la información requiere distintos tipos de redes (véase el apartado 3.2.2), según la naturaleza de ésta.
- **Los componentes se encuentran altamente distribuidos.** Tanto los sensores, que se encargan de recoger la información del entorno, como los actuadores, que transmiten la respuesta del entorno al usuario, tienen localizaciones distribuidas. Nuevamente se añade una dificultad extra a la hora de desarrollar aplicaciones.
- **La configuración del entorno es dinámica.** No se puede prever siempre el momento en que se conectan y desconectan nuevos dispositivos o entran y salen nuevos usuarios.
- **El sistema tiene que estar funcionando siempre.** Por ejemplo, en una vivienda un usuario consideraría inadmisible desconexiones periódicas del

[9] *Ambient Intelligence*

sistema. Esto implica que la gestión de nuevos componentes se tenga que realizar con el sistema ejecutándose.

La combinación de las *aplicaciones sensibles al contexto* y los *entornos activos* conlleva una serie de dificultades adicionales. A la información contextual generada por usuarios, dispositivos y aplicaciones hay que añadir el contexto del entorno. Dentro de un *entorno activo* se pueden encontrar fuentes de información contextual de diversa naturaleza. Las fuentes pueden ser muy cercanas al mundo físico, o por el contrario pueden estar relacionadas con el mundo virtual. El primer conjunto se compone de toda clase de dispositivos ligados al entorno que interaccionan con el mundo físico tales como sensores, conmutadores, electrodomésticos, pantallas, micrófonos, altavoces, etc. El segundo conjunto incluye aquellos componentes puramente computacionales, tales como gestores de diálogos, agentes inteligentes, clientes de correo, etc. Ya se ha mencionado la naturaleza distribuida de estos componentes, lo cual implica mayor complejidad en el desarrollo de aplicaciones.

Un problema importante surge por la diferencia en cuanto al nivel de abstracción en la información contextual aportada por los distintos componentes. Los sensores manejan información de alta resolución que es rica en detalles pero pobre en cuanto a abstracción. En cambio las aplicaciones pueden requerir información contextual más elaborada que la que aportan los sensores. Los desarrolladores tienen que convivir con información que tiene distinta resolución, lo que implica distintos formatos y distintas redes de distribución.

Finalmente, la interfaz con el usuario debe ser lo más flexible posible, para así poder obtener un interacción natural, permitiendo que esta tecnología forme parte de la vida cotidiana del usuario. En este punto el contexto también juega un papel importante. La interacción persona-ordenador no es tan efectiva como la comunicación entre dos personas. Un factor que contribuye a esta ineficiencia es que una parte importante de la comunicación se basa en un conocimiento común que poseen las personas que permite entender una situación. Este conocimiento implícito de la situación facilita la comunicación y permite desambiguar partes del discurso. En la comunicación persona-ordenador ocurren dos fenómenos paralelos e igualmente nocivos, por un lado la imposibilidad por parte del usuario de utilizar los mismos códigos que emplearía con una persona, por otro lado por parte del ordenador de capturar esa información contextual. Como resultado la persona se adecúa a la máquina y no al revés. Un punto importante es determinar cuál es la información contextual que el usuario debe suministrar al ordenador para conseguir una interacción más natural. Una vez determinado el contexto, es preciso capturarlo. Para ello se requiere de múltiples sensores e interfaces de tal forma que se pueda captar tanto la interacción explícita como la implícita del usuario, teniendo en cuenta que este proceso se realice de forma no intrusiva.

1.3. Solución propuesta y contribuciones

Nuestra propuesta parte de un modelo de *entorno inteligente* en el cual el entorno esta compuesto por un conjunto de fuentes contextuales distribuidas y cuyo

número varía dinámicamente. Ligado al entorno se provee una infraestructura que permite distribuir el contexto producido por las fuentes. Esta infraestructura se denominada capa de contexto, y sirve de intermediaria entre las fuentes contextuales y las *aplicaciones sensibles al contexto.*

La capa de contexto aporta:

- Una representación estándar del contexto que es independiente de su naturaleza y nivel de abstracción. Se provee de un modelo unificado del mundo reduciendo la complejidad en el acceso a la información contextual.
- Un mecanismo común para poder almacenar y distribuir esa información. Esto se realiza mediante una única interfaz que abstrae los detalles de comunicación entre los diferentes dispositivos físicos.

En esta tesis se propone la capa de contexto como pegamento para conseguir la sinergia necesaria entre los elementos ubicuos que constituyen el *entorno inteligente.* De acuerdo con otros trabajos tales como [292] [122] [171], creemos que un modelo del mundo centralizado es la mejor opción para obtener interacciones complejas entre dispositivos y aplicaciones.

La implementación de la capa de contexto reside en una estructura de datos global denominada pizarra [93]. Esta pizarra constituye un modelo del mundo, donde se encuentra almacenado todo el contexto generado por los componentes del entorno y por el mismo entorno. La pizarra almacena la información en forma de grafo, donde cada vértice representa una entidad, como puede ser un usuario, una aplicación, un dispositivo, una habitación, etc. Cada entidad se compone de un conjunto de propiedades, que se representan por pares nombre-valor. Las aristas que unen los vértices simbolizan relaciones entre las entidades. De esta manera, la información contextual queda recogida tanto en las propiedades de las entidades (la temperatura de una habitación, el estado de un dispositivo, la identidad de un usuario...) como en las relaciones entre las entidades. Por ejemplo, la localización de una persona queda capturada por una relación entre una entidad usuario y una entidad entorno. Las aristas del grafo también se emplean para representar otros tipos de relaciones como son los flujos de información entre los dispositivos multimedia (véase el apartado 4.7.2). Los dispositivos y aplicaciones utilizan la información de la pizarra para entender el contexto y adaptarse a él.

El contexto se representa mediante un lenguaje común, de tal forma que conviven en la pizarra información procedente de los sensores —como el estado de los dispositivos físicos— con contexto deducido a partir de la información sensorial, como por ejemplo el número de personas que hay en el entorno, o la tarea que está realizando una persona en cada momento.

Para acceder y modificar la información almacenada en la pizarra se propone un mecanismo asíncrono de comunicación. En este mecanismo podemos distinguir dos tipos de componentes: productores y consumidores. El mecanismo proporcionado consiste en una comunicación indirecta, ya que todos los mensajes pasan por la pizarra. Los productores modifican la pizarra ya sea cambiando una propiedad de una entidad, una relación entre dos entidades, o añadiendo o borrando una entidad. Los consumidores se enteran de estos cambios, bien porque están suscritos

a la pizarra y se les notifica cada vez que se produzca un cambio, bien porque acceden directamente a la pizarra donde han quedado registradas las modificaciones. Este tipo de interacción conlleva que los productores y los consumidores se encuentren débilmente acoplados. Los productores no necesitan conocer la identidad y número de consumidores, y éstos no requieren saber quién fue el emisor del mensaje. Hay que añadir que los dos participantes en la interacción no necesitan estar activos al mismo tiempo, ya que uno puede modificar la pizarra y desaparecer, y posteriormente, el otro acceder a la pizarra para recuperar el cambio. Esta relación entre productores y consumidores permite que los participantes puedan aparecer y desaparecer minimizando los costes de configuración, lo cual es una ventaja considerable en la implementación de sistemas donde la relación entre los componentes es dinámica, como es el caso de los *entornos inteligentes.*

En vista a lo anterior, el modelo de programación que se propone es orientado a datos. El foco de atención en el desarrollo se encuentra en el modelo de datos empleado. Los componentes del entorno se comunican mediante modificaciones en este modelo. Este enfoque requiere un análisis cuidadoso del contexto y sus relaciones que tenga como resultado un modelo consistente. A cambio se obtiene una mayor flexibilidad en la funcionalidad, ya que el mismo modelo se puede reutilizar para desarrollar aplicaciones independientes. Para la representación de la información se ha diseñado un dialecto de XML. Se ha elegido XML ya que es eficaz a la hora de describir información estructurada pero irregular, lo cual posibilita un mecanismo flexible de representación de contexto.

Se han encontrado cuatro ventajas que aporta la capa de contexto:

- Facilita la cooperación entre las diversas aplicaciones y dispositivos, ya que la información compartida se encuentra centralizada.
- Presenta a los módulos de la capa de aplicación una interfaz común que abstrae la diversidad de dispositivos del entorno físico. Cualquier tipo de red de dispositivos que implemente una pasarela a nuestra capa de contexto puede integrarse dentro del sistema.
- Se consigue una implementación que es independiente del lenguaje de programación elegido. Los módulos de aplicación no están atados a una plataforma específica, y la implementación del protocolo de comunicación no constituye un costo importante.
- Facilita las operaciones de depuración, ya que en la pizarra se encuentra en un momento dado toda la información sobre el estado del sistema.

1.4. Limitaciones

La propuesta del apartado anterior conlleva una serie de limitaciones que circunscriben la extensión del trabajo. Dentro de los objetivos mencionados se encuentran el estudio de la representación, distribución y almacenamiento del contexto. Ahora bien, existen otros servicios que no son abordados en la capa de

contexto. El modelo planteado parte de un conjunto de fuentes contextuales capaces de generar el contexto del entorno y comunicarlo a los consumidores a través de la pizarra. La tecnología específica que se utilice para obtener el contexto queda encapsulada por las fuentes de contexto. La capa de contexto no proporciona mecanismos para adquisición del contexto, sino que estos tienen que ser desarrollados por terceras personas.

Los mecanismos de filtrado y extracción de características son también externos a la capa de contexto. En la pizarra convive información contextual de diversos niveles de abstracción. Los sensores envían el contexto directamente obtenido del medio físico. Otros módulos lo recogen de la pizarra para procesarlo, y dejan un contexto más elaborado que es empleado por las aplicaciones finales. La técnicas elegidas para realizar este procesamiento variarán según la aplicación que se pretenda desarrollar. En cualquier caso, no son proporcionadas por la capa de contexto.

Por otro lado, la pizarra impone restricciones en la naturaleza de la información contextual que puede manejar. En particular, se asume que la información contextual no es síncrona. Esto excluye tráfico como audio o vídeo que requiera que cada nuevo cambio llegue a su destino con un retardo mínimo. También se exige que el contexto varíe lentamente, tal como se describirá en el capítulo 7. Esto implica que se tendrán que prever mecanismos externos que filtren la información de aquellas fuentes excesivamente rápidas. Así por ejemplo, si se dispusiera de un sensor que cada milisegundo aportara la temperatura del entorno, sería necesario intercalar un componente que se encargara de promediar los valores obtenidos durante un segundo. El resultado de este filtrado sería el que se introduciría en la pizarra.

Otra limitación inherente a este trabajo viene impuesta por las características de los *entornos activos*. Tal como se definió en el apartado 1.1, se asume que un *entorno activo* provee de una infraestructura. Esto supone la existencia de un conjunto de servicios accesible en cualquier momento por las aplicaciones y dispositivos que operan dentro del entorno. En contraposición a esta disposición estarían las redes dispositivos *ad-hoc*. Éstas se crean y se destruyen dinámicamente según las necesidades de los componentes. La infraestructura necesaria incluye como mínimo una pizarra que implemente la capa de contexto. Por otro lado, un entorno delimita espacialmente el estudio a realizar mediante unas barreras físicas claramente definidas.

La escalabilidad es un aspecto a tener en cuenta en el diseño de los sistemas distribuidos. Al restringirnos a un entorno físico (una habitación, un edificio), la escalabilidad deja de ser un tema crítico, aunque sigue siendo importante y merece su consideración. En cualquier caso, una premisa que se establece es que la capa de contexto propuesta tenga validez dentro de un espacio que abarque al menos un edificio.

La seguridad es un tema importante en un área como la computación ubicua [252]. En concreto, en la implementación de un entorno inteligente son necesarios mecanismos fiables de autentificación y encriptación que preserven la intimidad del usuario e impidan añadir componentes no autorizados al entorno. Los mecanismos de seguridad no han sido tratados dentro de esta tesis, sino que se ha presupuesto

su existencia.

1.5. Organización del documento

El presente documento se estructura en tres partes: introducción, propuesta y conclusiones. La primera parte la componen los tres primeros capítulos. La segunda parte se desarrolla en los capítulos cuarto, quinto y sexto. La última parte incluye al séptimo y octavo capítulo. Finalmente se incluyen una serie de apéndices y la bibliografía empleada.

Los capítulos de que consta la tesis se resumen en:

- Capítulo 1. El presente capítulo ha permitido exponer el marco sobre el cual se ha desarrollado la tesis, identificando los problemas que han motivado el trabajo realizado, señalando las contribuciones al estado del arte y las limitaciones de las mismas.

- Capítulo 2. Este capítulo muestra el desarrollo que ha tenido la *Computación Ubicua* desde sus inicios hasta la actualidad. Para ello se han estudiado dos áreas de la *Computación Ubicua* centrales para esta propuesta: las *aplicaciones sensibles al contexto* y los *entornos inteligentes.*

- Capítulo 3. Mientras que el capítulo anterior se centra en las aplicaciones sensibles al contexto, este capítulo presenta el estado del arte de las arquitecturas que se utilizan en su desarrollo. El capítulo se divide en dos partes: primeramente se describen los diversos tipos de componentes que se pueden encontrar en la arquitectura de un *entorno inteligente* y, seguidamente, se detallan las plataformas más importantes que facilitan el despliegue y uso de dichos componentes. Al final de este capítulo se incluyen las conclusiones de toda la parte de introducción.

- Capítulo 4. Este capítulo aborda la definición y representación del contexto. Se estudia el concepto de información contextual desde distintos puntos de vista. A partir de la noción adquirida se realiza un modelo de representación del contexto que se propone como punto de partida para el desarrollo de *aplicaciones sensibles al contexto.*

- Capítulo 5. En este capítulo se presenta la metáfora de pizarra como mecanismo de comunicación propuesto para la distribución del contexto dentro de un *entorno inteligente.* Durante el capítulo se explica en qué consiste este paradigma presentando sus ventajas y sus inconvenientes. Se detalla el funcionamiento de la pizarra, y del resto de los componentes que constituyen la arquitectura propuesta.

- Capítulo 6. Este capítulo se centra en explicar el lenguaje de representación que se ha diseñado para la descripción de la información contextual. Al final del capítulo se hace especial énfasis en cómo se ha empleado este lenguaje para poder implementar interfaces de usuario generadas dinámicamente.

- Capítulo 7. Se presentan los resultados de los experimentos realizados donde se incluyen medidas sobre el rendimiento de la plataforma y de los demostradores realizados empleando la arquitectura propuesta.
- Capítulo 8. Este capítulo resume las conclusiones obtenidas en cuanto a la propuesta presentada en los capítulos 4, 5 y 6 y detalla la producción científica que ha dado como resultado la tesis.
- Bibliografía. Finalmente se incluye las referencias bibliográficas citadas durante el texto numeradas por orden de aparición.
- Apéndices. Se incluyen dos apéndices donde se especifican, por un lado, la definición de las clases que constituyen el modelo del contexto descrito en el capítulo 4, y por otro, las clases particulares y las instancias que —unidas al modelo anterior— se emplean para describir los componentes del entorno experimental.

Capítulo 2

Computación ubicua

2.1. Introducción

Tal como se ha explicado en el capítulo 1, la *Computación Ubicua* propone dotar a los objetos cotidianos de capacidad de computación y de comunicación. Se apuesta por una amplia red de dispositivos interconectados entre sí. El fin de esta red de dispositivos es facilitar en lo posible la interacción persona-ordenador, de tal forma que el usuario pueda comunicarse con el sistema del mismo modo que se comunica con otros usuarios.

Un primer paso para conseguir una interacción así de fluida es incorporar el contexto como entrada del sistema. El contexto aporta información implícita sobre la situación de la persona y los objetos implicados. Cuando una persona establece un diálogo con otra, ya sea oral o escrito, el contexto es parte fundamental para determinar el significado del mensaje que intenta transmitir. En cambio, cuando la misma persona se comunica con un ordenador, se encuentra con graves carencias en la interacción. Uno de los problemas más importantes son las interfaces de que dispone la persona para interaccionar con el ordenador. Los dispositivos de entrada y salida convencionales (ratón, teclado y pantalla) no son un medio natural de comunicación para las personas. La inclusión de nuevos modos de interacción más cercanos a los que los usuarios están acostumbrados a utilizar en su vida cotidiana es una condición necesaria para superar las limitaciones actuales. Pero esta búsqueda de una interacción natural no quedará completa si no se tiene en cuenta el contexto. Más aún, cuando en la *Computación Ubicua* la interacción persona-ordenador se extiende a todo el espacio. Las situaciones en las que se puede encontrar el usuario pasan de estar frente a un ordenador a interaccionar libremente en un espacio físico: el contexto se multiplica y adquiere mayor relevancia. Aparece entonces un nuevo tipo de aplicaciones que tratan el contexto como una variable de entrada más, y que se denominan *aplicaciones sensibles al contexto.*

Es habitual en las ciencias de la computación referirse al término *entorno*. Éste adopta diferentes significados; así se puede hablar de *entorno* de desarrollo para designar al conjunto de herramientas que facilitan al desarrollador la implementación de aplicaciones. También está el *entorno* de ejecución, que es el conjunto de condiciones extrínsecas que necesita un sistema informático para funcionar, y

que engloba el hardware y el sistema operativo. Estrechamente relacionadas con la anterior acepción de *entorno* se encuentran las variables de *entorno* que almacenan información accesible por todas las aplicaciones que se ejecutan en un mismo *entorno* de ejecución. La mayoría de estos *entornos* comparten una naturaleza virtual, ya que su existencia está condicionada a los propios programas.

La relación más elemental entre una aplicación informática y el entorno físico se obtiene a través del dispositivo donde se ejecuta (ya sea un ordenador personal, un portátil, una PDA, un microcontrolador...), ya que toda aplicación precisa de al menos un dispositivo físico. La interacción del usuario con la aplicación se realiza a través de interfaces que sirven de puente entre los dos entornos, el físico y el virtual. Esta relación también se encuentra en sentido contrario, ya que una aplicación puede afectar al mundo físico. Por ejemplo, el producto de una computación puede ser algo tangible como una hoja impresa, o un efecto en el entorno, como un cambio en la temperatura.

Las *aplicaciones sensibles al contexto* son un tipo de aplicación que tiene una estrecha relación con el entorno físico. Una *aplicación sensible al contexto* consta de sensores y dispositivos que capturan el contexto operando dentro de un espacio físico. La extensión de este espacio puede abarcar desde todo el planeta hasta un pequeño recinto. Para el ámbito de esta tesis, cuando el espacio se encuentra limitado por barreras físicas entonces se denomina entorno.

En el análisis de la información contextual hay que tener en cuenta el entorno. Por un lado, limita el alcance del análisis al espacio que engloba dicho entorno. Por otro lado, el propio entorno aporta información contextual a tener en cuenta en el análisis. Siguiendo en esta línea, los *entornos activos* [1] o *Entornos Inteligentes* [2] son espacios que, además de servir como contenedores de *aplicaciones sensibles al contexto* y contribuir al mismo contexto, constituyen una parte de la aplicación. Estos entornos disponen de una infraestructura accesible por los dispositivos y aplicaciones que se ejecutan dentro de él. Esta infraestructura simplifica el diseño de las *aplicaciones sensibles al contexto*, ya que parte de la funcionalidad se confía al *entorno activo*. Así, distintas aplicaciones pueden compartir los servicios ofrecidos por el *entorno activo* potenciando la reutilización en el diseño.

A continuación se va a realizar un introducción a la *Computación Ubicua* resaltando su importancia como nuevo paradigma emergente. Seguidamente se va a plasmar su estrecha relación con las *aplicaciones sensibles al contexto* y los *entornos activos*. En este capítulo se realizará un repaso a las *aplicaciones sensibles al contexto* más relevantes, y se introducirán los *entornos activos* que han sido desarrollados tanto en el ámbito académico como en el comercial. En el capítulo 3 se estudiarán las tecnologías necesarias para llevar a cabo las aplicaciones descritas en el presente capítulo.

[1] Active Environments
[2] Intelligent Environments, Smart Environments

2.2. ¿Qué es la *Computación Ubicua*?

De acuerdo con Mark Weiser[287], las dos características más importantes de la *Computación Ubicua* son la **ubicuidad y la transparencia del sistema.** Por ubicuidad se entiende que las capacidades de procesamiento del sistema (dispositivos, microprocesadores, memoria, interfaces...) se encuentran integradas y distribuidas por el entorno. Esto permite que el usuario pueda disponer de las aplicaciones en cualquier momento y en cualquier situación (en la literatura esta propiedad se denomina con los términos *anywhere* y *anytime*). Por otro lado, la transparencia garantiza que el usuario interaccione con el sistema de forma natural. Los sistemas tradicionales se basan en una interfaz centrada en un ordenador personal, donde la interacción se encuentra limitada por los dispositivos de entrada (teclado y ratón) y salida (pantalla) del ordenador. Esto obliga a recurrir a metáforas que no son naturales para el usuario. De ahí que la computación ubicua propicie nuevos métodos de interacción que complementen, y suplan cuando sea posible, las metáforas clásicas.

Para alcanzar estos dos objetivos, Weiser establece tres líneas de actuación:

- La proliferación de dispositivos más allá de los ordenadores personales.
- La integración del entorno físico como parte del sistema.
- Una interacción sin trabas entre el usuario y los dispositivos.

La primera línea implica el desarrollo de una miríada de dispositivos que sirvan de interfaz al usuario. Así se facilita el intercambio de información, ya que el usuario puede elegir el dispositivo que más le conviene en cada momento. Esto implica la propuesta de nuevos modelos en el desarrollo de aplicaciones; según Guruduth Banavar *et al.* [27], los dispositivos no hay que tratarlos como almacenes de aplicaciones, ni éstas se encuentran sujetas a un único dispositivo: por el contrario, las aplicaciones son un medio para completar una tarea, y para ello emplearán los dispositivos que sean necesarios.

El objetivo de la segunda línea de actuación es salvar la barrera entre el entorno físico y el entorno virtual. Las actividades que realiza una persona suelen circunscribirse dentro de un entorno concreto. La persona puede hacer uso de los recursos que tenga dicho entorno para completar la tarea. Por ejemplo, el empleo del escritorio como metáfora de interacción ha reducido las actividades computacionales al ordenador personal [288]. Esta forma de trabajar se encuentra muy separada de la que el usuario está acostumbrado en su quehacer diario, y no explota todas las posibilidades que le podría ofrecer el entorno.

Para superar esta limitación se han desarrollado dos áreas de investigación que se colocan en paralelo con la *Computación Ubicua*, y que se conocen como *entornos inteligentes* y como *Realidad Aumentada*. Los *entornos inteligentes* se tratarán en profundidad en el apartado 2.5. Respecto a la segunda, uno de los trabajos de referencia que fusiona la *Computación Ubicua* con la *Realidad Aumentada* dio como resultado el concepto de *Interfaces de Usuario Tangibles* (IUT) [3] [151].

[3] Tangible User Interfaces (TUI))

Los IUTs permiten interaccionar con el entorno virtual empleando objetos físicos, tales como libros, tarjetas, tazas..., o partes del entorno, como paredes, puertas, ventanas... Más recientemente, los IUTs se han pasado a denominar *Interfaces Ambientales* [4] y siguen siendo objeto de estudio y desarrollo [121].

Finalmente, la tercera línea de actuación se centra en potenciar nuevas metáforas de interacción. En este sentido, y al hilo del párrafo anterior, la *Computación Ubicua* promueve una interacción multimodal. El usuario dispone de varios modos de comunicación pudiendo elegir el que más le convenga. Las interfaces usuario gráficas pasan a ser uno más de los posibles modos disponibles por el usuario. Otros posibles modos que se pueden encontrar en un entorno ubicuo serían el lenguaje natural, los gestos, la mirada... En general, se espera que el usuario utilice una mezcla de varios modos, cambiando entre ellos dinámicamente.

Dentro de los modos de interacción se podría establecer una clasificación entre comunicación explícita e implícita. En un modo de comunicación explícita la mayor parte de la información se facilita directamente en el mensaje, mientras que en un modo implícito gran parte de la información se sobreentiende al ser compartida por los participantes. Los sistemas informáticos actuales se basan en comunicación fundamentalmente explícita. El usuario indica explícitamente a través de una interfaz la acción que quiere realizar, y éste responde en consecuencia. Por ejemplo, el ratón proporciona una comunicación explícita. En la interacción que propone la *Computación Ubicua* la comunicación implícita adquiere una mayor importancia. El usuario realiza acciones que no están dirigidas a la interacción pero que el sistema las interpreta como tal. Éste sería el caso de detectar hacia dónde mira el usuario para desambiguar el significado de una determinada orden. Un aspecto clave dentro de la comunicación implícita es la información contextual [239]. Así, un entorno ubicuo y multimodal no aprovechará sus posibilidades si no considera el contexto como parte de la interacción.

2.3. La era de la *Computación Ubicua*

La evolución de las tecnologías de la información y comunicación presenta varios hitos importantes que han dado lugar a la aparición de nuevos paradigmas de interacción. Dicha evolución se puede resumir en cuatro eras [290]:

- **La era de las supercomputadoras.** Esta etapa marca los principios de la historia de la informática. Abarca desde la aparición de los primeros ordenadores en la segunda mitad del siglo XX (Mark, Eniac, Univac) hasta la actualidad (BigBlue, Marenostrum), pasando por el desarrollo de los supercomputadores de los años 80 (Cray I,II, III). Aunque la tecnología ha avanzado considerablemente durante estos años, el modelo de interacción se ha mantenido constante: un ordenador que es compartido por varios usuarios.

- **La era del ordenador personal.** La aparición de los circuitos integrados propicia la posibilidad de construir ordenadores cada vez más pequeños y

[4] Ambient Interfaces

más baratos. Al principio de la década de los 80 se comercializan los primeros ordenadores personales [5] que democratizan el empleo de la informática. Aparece un nuevo modelo de uso en el que cada usuario dispone de su propio ordenador.

- **La era de Internet y de los sistemas distribuidos.** La creación de la red Internet y el protocolo TCP/IP se remonta a los años 70. No obstante, no es hasta la década de los 90 cuando el fenómeno Internet, con la aparición de los primeros navegadores gráficos y el lenguaje HTML, entra en la vida cotidiana. En esta nueva era aparece un modelo de uso híbrido entre los dos anteriores. Por un lado, se mantiene la relación en el uso de uno a uno entre usuario y ordenador, por otro, se ofrece la posibilidad de usar a través de la red los recursos de otros ordenadores.

- **La era de la *Computación Ubicua*.** Surge como evolución de la era anterior. La potencia de computación y las capacidades de comunicación se distribuyen en infinidad de dispositivos, donde los ordenadores personales son uno más. Se deriva en un nuevo modelo de uso en el que muchos ordenadores son compartidos por muchos usuarios.

En los últimos años, los avances en tecnología electrónica [277] han propiciado que se pueda dar el salto a esta nueva era. Uno de los fenómenos que ha marcado este impulso es la progresiva descentralización de la capacidad de computación [130]. Tal como explica Bill Schilit [236], esta descentralización se produce en tres ejes principalmente:

- **Megahercios.** La tecnología de circuitos integrados produce microprocesadores cada vez más potentes y más pequeños. Esto propicia que la potencia de computación deje de ser monopolio de las grandes estaciones de trabajo, o de los ordenadores personales, distribuyéndose entre multitud de dispositivos emergentes. Electrodomésticos, teléfonos, automóviles, agendas y relojes, entre otros, forman una nueva generación de aparatos que incorporan capacidades de computación, y que ya están disponibles en el mercado.

- **Megabits por segundo.** En paralelo, el ancho de banda disponible por metro cúbico aumenta año a año. Esto permite expandir el concepto de red de ordenadores tradicionales, al de red de dispositivos. En particular, el desarrollo de la tecnología inalámbrica en los últimos años ha producido la proliferación de dispositivos móviles con capacidad de comunicación.

- **Megabytes.** Este tercer eje se refiere al aumento en la capacidad de almacenamiento. De esta forma se posibilita también una mayor descentralización de la información. Los futuros dispositivos de almacenamiento [279] permitirán que toda la información personal pueda viajar junto con su propietario, facilitando un acceso ubicuo a la misma.

[5] Personal Computer (PC)

2.4. Sistemas ubicuos

La descentralización que se está produciendo en los tres ejes anteriores viene acompañada por el desarrollo de múltiples sistemas ubicuos. Éstos comenzaron siendo parte de proyectos científicos y ahora están empezando a migrar del laboratorio al mercado.

Uno de los primeros esfuerzos dentro de la *Computación Ubicua* se realizó a finales de los ochenta en el *Electronics and Imaging Laboratory* perteneciente al *Xerox Palo Alto Research Centre*. En 1988, en este laboratorio se desarrolló el programa de *Ubiquitous Computing* cuyo nombre posteriormente se ha tomado prestado para referirse a todo un área de investigación. Este programa incluía los proyectos *LiveBoard* [91], *ParcPad* [157] y *ParcTab* [280]. Durante los siguientes años se terminó el conocido sistema *Active Badge System* [278] (propiedad de la empresa Olivetti) y en 1992 se implementó el primer sistema *Ubi-Comp* [291].

En Europa, la iniciativa *Dissapearing Computing* perteneciente al quinto programa Marco[6] ha englobado desde el 2001 a diecisiete proyectos centrados en el desarrollo de tecnología ubicua.

Gran parte de los sistemas ubicuos desarrollados hasta el momento caen dentro del área de las *aplicaciones sensibles al contexto* (véase el apartado 2.4.1), o se encuentran integrados dentro de los *entornos activos* (véase el apartado 2.5.1).

Otras áreas donde ha calado la *Computación Ubicua*, y no se encuentran en los dos apartados anteriores son:

- Asistencia a la tercera edad [253] y [139].
- Navegadores web [33].
- Asistencia a personas discapacitadas [206].
- *Campus* ubicuos como el proyecto ActiveCampus [120].
- Modelado de usuario [137].
- Computación recreativa [268] y [57] (que son dos proyectos de investigación asociados al consorcio [94]).
- Seguridad y privacidad [252].

En el ámbito nacional se han realizado importantes progresos, tanto en entornos educativos [201] [270], como en museos virtuales y espacios naturales [248], domótica [184], plataformas de agentes móviles [56] y tecnologías de la rehabilitación [2].

La difusión de los avances en *Computación Ubicua* se encuentra cubierta con la celebración en la actualidad de tres conferencias anuales con dedicación exclusiva: *Ubicomp*, *Pervasive* y *Percom*, y la edición de dos revistas especializadas *Personal and Ubiquitous Computing* y *IEEE Pervasive Computing*. En la red se puede encontrar una extensa recopilación de proyectos, personas e información relacionada

[6] 5th Research Framework Programme (FP5)

con la *Computación Ubicua* en el portal *Ubiquitous Computing Research Page* en la dirección <http:\\www.ucrp.org>.

2.4.1. La *Computación Ubicua* y las aplicaciones sensibles al contexto

Según Anind Dey [84] una aplicación sensible al contexto es aquella que emplea el contexto para ofrecer al usuario información y/o servicios relevantes, donde la relevancia depende de la tarea del usuario. En este mismo artículo, Dey propone una clasificación de las aplicaciones sensibles al contexto atendiendo a la siguientes características:

- *Presentación de información.* Son aquellas aplicaciones que muestran información personalizada según el contexto del usuario.
- *Ejecución automática* de un servicio dependiendo del contexto.
- *Etiquetado* del contexto. Estas aplicaciones etiquetan el contexto recopilado para que posteriormente el usuario pueda consultar los cambios que se han producido.

Por otro parte, Bill Schilit [234] establece cuatro categorías de *aplicaciones sensibles al contexto* que surgen de aplicar dos criterios ortogonales. El primero divide las aplicaciones según si la tarea que realizan provee algún tipo de información o si permite enviar comandos. El segundo criterio establece si la tarea precisa de la intervención del usuario (*manual*) o no (*automática*). Las cuatro categorías que distingue Schilit se puede ver enmarcadas en estos dos ejes en la tabla 2.1.

	manual	automático
información	Selección próxima	Reconfiguración contextual automática
comandos	Comandos contextuales	Acciones disparadas por contexto

Tabla 2.1: Clasificación de las aplicaciones sensibles al contexto según [234]

Estas cuatro categorías se resumen de la siguiente forma:

- **Selección próxima.** Engloba aquellas aplicaciones que presentan una interfaz de usuario capaz de resaltar aquellos objetos o recursos que se encuentran más próximos al usuario.
- **Reconfiguración automática contextual.** Dentro de esta categoría entran aquellas aplicaciones que usan la información contextual para cambiar la configuración de los componentes del sistema, por ejemplo, para cambiar los canales de comunicación.

- **Comandos contextuales.** Esta categoría la componen aquellas aplicaciones que cuando el usuario activa un comando, producen diferentes resultados dependiendo del contexto.

- **Acciones disparadas por contexto.** Incluye aquellas que mediante simples reglas del tipo *if-then-else* realizan diferentes acciones dependiendo de los diferentes eventos producidos por cambios en el contexto.

La clasificación anterior la resumen Guanling Chen y David Kotz [61] en dos categorías:

- **Aplicaciones sensibles al contexto de forma activa.** Son aquellas que automáticamente se adaptan a las actualizaciones del contexto modificando su comportamiento.

- **Aplicaciones sensibles al contexto de forma pasiva.** Reaccionan a los cambios del contexto a petición del usuario, además guardan estos cambios para poder recuperarlos en un futuro.

En pro de visualizar cómo se aplican los criterios expuestos, en la tabla 2.2 se muestra una clasificación de un conjunto de aplicaciones sensibles al contexto que se encuentran en la literatura. Para cada una se especifica: (a) el tipo de contexto que emplea —activo o pasivo— según la definición de Guanling Chen y David Kotz; (b) las características que presenta la aplicación según el criterio de Anind Dey : Presentación, Ejecución automática y/o Etiquetado; y (c) las variables contextuales que más habitualmente se emplean: la actividad que está realizando el usuario (A), la identidad del mismo y de las personas que le acompañan (I), la localización del usuario (L) y el tiempo (T). Esta última variable se considera en dos sentidos: o bien que la aplicación guarda un histórico de los cambios contextuales en los que incluye la fecha en la que se produjo el cambio como una variable más del contexto, o bien que la tarea que lleva a cabo la aplicación también se ve afectada por el instante de tiempo en que se está realizando.

Existen en la literatura buenas recopilaciones de *aplicaciones sensibles al contexto* [46, 84, 162, 61]. A continuación se van a presentar las más destacadas, agrupadas según su funcionalidad.

Guías Turísticos Contextuales

Los guías turísticos contextuales en general se emplean para guiar al usuario a través de un espacio, ya sea abierto o cerrado, mostrándole cierta información cada vez que el usuario se acerca a un sitio relevante del recorrido. El usuario porta consigo algún tipo de dispositivo inalámbrico de forma que se le muestra en cada momento en qué parte del recorrido se encuentra. La primera aplicación que surgió en este campo fue Cyberguide [6] desarrollada en el grupo *Future Computing Environments* (FCE) de la Universidad de Georgia Tech. Siguiendo un enfoque similar han surgido distintos proyectos como el proyecto GUIDE [64] de la Universidad de Lancaster, o los proyectos *Metronaut* [250] y *Smart Sight Tourist Assistant* [300] de la Universidad Carnegie Mellon. Dentro de los guías

Nombre de la Aplicación	Descripción	Tipo de Contexto								Propiedades		
		Activo				Pasivo						
		A	I	L	T	A	I	L	T	P	E	T
ActiveBadge [278]	Recepcionista telefónico		X	X						X	X	
ParcTab [280]	Localización de personas							X		X		
Teleporting [294]	Interfaz ubicua		X	X						X		
Cyberguide [6]	Guía turístico			X	X			X		X		
Fieldwork [203]	Recolección de datos							X	X	X		X
Stick-e document [44]	Notas contextuales	X	X	X						X		X
Conference Assistant [87]	Gestor de reuniones		X	X	X	X					X	X
Office assistant [299]	Gestor de visitas	X	X		X						X	
ComMotion [192]	Mensajes contextuales			X	X					X	X	
Jimminy [221]	Memoria Proactiva		X	X						X		X

Tabla 2.2: Clasificación de las aplicaciones sensibles al contexto según el tipo de contexto que manejan y las características que presentan. Esta tabla ha sido confeccionada a partir de la clasificación elaborada en [84], incluyendo los criterios definidos por [61], y expandiéndola con nuevas aplicaciones contextuales.

turísticos contextuales, las guías aplicadas a museos [199, 259, 66] constituyen un área propia.

Memoria Contextual

Este conjunto de aplicaciones presenta dos enfoques. Por un lado están las aplicaciones cuya misión es anotar cuándo se ha producido una situación relevante para el usuario, anotando el contexto en que fue producida. Posteriormente, el usuario podrá consultar las distintas situaciones almacenadas. Estas aplicaciones hacen un uso pasivo del contexto. Por otro lado están las que pretenden potenciar la memoria del usuario recordando sucesos pasados cuando éste se encuentra en un contexto determinado. En este caso el uso es activo, ya que al rememorar la situación se desencadenan una serie de acciones asociadas.

Una de los primeras herramientas que se desarrolló fue *Forget-Me-Not* [163] que utilizaba el sistema *ParcTab* [280]. Cada vez que el usuario se encontraba con otra persona o interaccionaba con un dispositivo quedaba grabado un suceso que incluía el lugar, la hora y el tipo de actividad. De esta forma se generaba una biografía con los eventos acontecidos que luego el usuario podía consultar.

Como ejemplo de contexto proactivo cabe destacar el proyecto Jimminy [221], que es la evolución para dispositivos móviles del proyecto *Remembrance Agent* [222]. Este agente integrado en el editor *Emacs* muestra automáticamente información relevante al texto que se está escribiendo haciendo uso del contexto físico del usuario y en función de otras notas o textos previamente escritos.

El formato en que se guarda el suceso a recordar no siempre tiene que ser textual. En el proyecto *StartleCam* [136] se utiliza una videocámara que graba automáticamente según el usuario muestre interés en determinada situación. Este interés se estima midiendo la conductividad de la piel. Para ello el sistema se entrena con patrones de conductividad que se han obtenido cuando el usuario ha recibido un estímulo que le ha llamado la atención. En tiempo de ejecución el sistema es capaz de reconocer la aparición de alguno de los patrones de entrenamiento, lo cual produce la activación de la videocámara.

Otros proyectos han utilizado la metáfora de las notas *Post-it* para recordar un cierto evento. Éste es el caso del proyecto *Stick-e document* [44] que permite asociar notas a localizaciones concretas. En la misma línea, pero permitiendo que los recordatorios aparezcan proactivamente cuando se cumplen ciertas condiciones del contexto, se encuentran el proyecto *Cybreminder* [85] y el proyecto *ComMotion* [192].

Un tipo de aplicaciones singulares dentro de este apartado son las aplicaciones para trabajo de campo. Éstas son sistemas que permiten añadir anotaciones contextuales a la información capturada en una actividad al aire libre. Éste es el caso del sistema *Fieldwork* [204] desarrollado en la Universidad de Kent y que se ha aplicado en dominios tan variados como excavaciones arqueológicas y proyectos de diversidad biológica [203]. La información contextual que se anota suele ser la localización y la fecha, de tal forma que se puede posteriormente recuperar para realizar un análisis espacio-temporal de cómo fueron recogidos los datos. Otro proyecto en la misma línea, pero en un campo de aplicación totalmente distinto, es

el *Ambient Wood Journal* [284] perteneciente a la iniciativa *Equator* de la Universidad de Nottingham. Este sistema permite anotar la interacción de varios niños en un espacio abierto (como un bosque) para realizar en el futuro una recreación de los sucesos más destacados.

Aplicaciones de comunicación contextual

Las aplicaciones de comunicación contextual constituyen una de las áreas más prolíficas. De hecho, las primeras *aplicaciones sensibles al contexto* que se desarrollaron pertenecen a este campo, como es el sistema ParcTab [280] de los laboratorios Xerox. Estos sistemas establecen algún tipo de comunicación entre dos o más personas modificando la presentación y/o el contenido del mensaje según el contexto de los participantes. Así actúa el sistema de filtrado de mensajes CLUES [176], que prioriza los mensajes dependiendo del contexto del usuario. Otro ejemplo son las listas de correos contextuales [86] que sólo envían un mensaje a los miembros de la lista que estén dentro de un edificio.

Estas aplicaciones contextuales también se emplean con dispositivos auditivos. Así, el sistema *Family Intercom* [189] permite mantener una conversación entre dos personas mediante una infraestructura de micrófonos y altavoces fijos mientras las personas se desplazan por las habitaciones, o el proyecto *Audio Aura* [187] de los laboratorios Xerox, donde se estudia cómo interrumpir la actividad de una persona para notificarle algún evento con diferentes sonidos que dependen del contexto del usuario.

También se han desarrollado aplicaciones de mensajería instantánea como el *Sun's AwareNex* [264] que provee al usuario de una mayor conciencia de la actividad y disponibilidad de los posibles receptores, así como *chats* contextuales como el *ConChat* [217] que enriquece la comunicación textual mediante información contextual y resuelve posibles conflictos semánticos.

Organizadores contextuales de citas

Este último apartado agrupa un conjunto de *aplicaciones sensibles al contexto* que ayudan al usuario a organizar su agenda teniendo en cuenta el contexto del usuario y de los asistentes a la reunión. Ésta es la motivación principal de la aplicación *Conference Assistant* desarrollada con la plataforma *Context Toolkit*. Esta aplicación realiza el seguimiento de la convocatoria de una reunión y recopila información contextual sobre la misma. Posteriormente, esta información se emplea para asegurar la asistencia de los usuarios. De la misma manera, Yasuyuki Sumi [260] propone un sistema que apoya el intercambio de información entre los participantes de un congreso, tanto durante como después de éste. El sistema emplea dispositivos móviles para mostrar a los asistentes información contextual sobre su entorno físico o sobre las personas cercanas. Esta información se filtra según el perfil del usuario.

Dentro de esta categoría también se encuentra el provecto *Context-Aware Office Assistant* [299] del *MIT Media Lab* que consiste en un agente software que permite el paso a un despacho dependiendo de la actividad y agenda del ocupante.

2.5. La *Computación Ubicua* y los *entornos activos*

El otro gran bloque de sistemas ubicuos lo constituyen los *entornos activos*. Un *entorno activo* [7] consiste en una infraestructura, delimitada por unas barreras físicas, capaz de interaccionar y adaptarse a sus ocupantes. Los *entornos activos* son un tipo de *aplicación sensible al contexto* que además pueden contener a otras *aplicaciones sensibles al contexto*. La infraestructura ofrecida se comparte por el conjunto de aplicaciones, dispositivos y personas que operan dentro del entorno. Tal como apunta Mahadev Satyanarayanan [233], los *entornos inteligentes* se muestran como un área clave de investigación dentro de la *Computación Ubicua*. Los *entornos activos* constituyen el marco que permite exportar la computación al mundo físico [73]: son espacios donde la interacción entre dispositivos inteligentes adquiere sentido [249]. El objetivo de estos entornos es conseguir sistemas interactivos y adaptables, que formen parten de la vida cotidiana de la personas [195].

Los *entornos activos* interactúan de forma natural con el individuo y le ayudan de manera no intrusiva en la realización de las tareas cotidianas. Esto se consigue mediante la adquisición de información contextual que permite adelantarse a los deseos del usuario. De este modo se consigue que las habitaciones u oficinas tengan una entidad propia y tomen la iniciativa en la interacción para mejorar la calidad de vida de las personas que las ocupan. Un *entorno activo* se podría ver como una *aplicación sensible al contexto* donde parte de los elementos computacionales se encuentran integrados con el entorno y quedan ocultos para el usuario.

Desde la década de los 90 se ha investigado en el desarrollo de *entornos inteligentes*. En el apartado 2.5.1 se va a realizar un repaso de los entornos más destacados. Tal como se expuso en el apartado 1.2, la realización de un *entorno inteligente* implica el despliegue de redes de comunicación heterogéneas, debido a la gran variedad de dispositivos que se pueden conectar y al tipo distinto de información que intercambian. En el apartado 3.2.2 se describirán las tecnologías más empleadas.

2.5.1. Evolución de los *entornos inteligentes*

Primeros entornos

Los primeros *entornos inteligentes* surgieron de proyectos relacionados con trabajo colaborativo. Estos proyectos se plantearon diversos objetivos, ya fuera capturar los resultados de una reunión de trabajo, como el proyecto *Colab* [254], o realizar tareas de videoconferencia. Éste fue el caso del proyecto *Telepresence* [223] en la Universidad de Ontario y los proyectos *Media Spaces* [37] y *IIIF (Integrated Interactive Intermedia Facility)* [50] de la compañía Xerox Parc.

[7] También denominado *entorno inteligente*

Oficinas

No es de extrañar pues que los primeros *entornos inteligentes* se desarrollaran empleando oficinas. Tal es el caso del proyecto *Resposive environment* [92] que fue uno de las primeros en el campo que aplicó los principios de la *Computación Ubicua* a los *entornos inteligentes*. También en esta línea se encuentra la *Reactive Room* [76] de la Universidad de Toronto. Más recientemente, se ha desarrollado en el laboratorio francés GRAVIR-IMAG la oficina inteligente *Smart Office* [106]. Actualmente, dos de los *entornos inteligentes* en funcionamiento más destacados son salas de reuniones activas, por un lado, el proyecto *Interactive Workspaces* [155] de la Universidad de Standford, que se centra en proveer de un entorno aumentado al usuario mediante pantallas multimedia y dispositivos móviles. Por otro lado, la sala de reunión de proyecto *Active Space* de la Universidad de Illinois, que ha servido como banco de pruebas para el sistema operativo distribuido Gaia [226].

Hogar

Un campo dónde los *entornos inteligentes* han tenido una fuerte repercusión ha sido en la vivienda, ya que esta tiene un gran atractivo por su potencial mercado de usuarios. En este campo es donde los *entornos inteligentes* se acercan más a los retos de la *Computación Ubicua*, si se tiene en cuenta que el público potencial lo forman usuarios no técnicos que requieren de una interacción transparente a la tecnología desplegada. Un proyecto líder en este campo es *The Aware Home* [159] desarrollado en la Universidad Georgia Tech por el grupo *Future Computing Environments*. Éste consiste en un casa real equipada con múltiples sensores y dispositivos ocultos, que permiten localizar al usuario y realizar un seguimiento de sus tareas, de forma que el entorno puede ayudarle en su quehacer diario. Otro ejemplo de casa real es el proyecto *The Neural Network House* [185] de la Universidad de Boulder, Colorado, que se centra en el estudio de redes neuronales para reconocer patrones de actividad de los habitantes. En la misma línea de predecir las tareas del usuario, pero utilizando modelos ocultos de Markov [219] se halla el proyecto *MavHome* [75] de la Universidad de Texas. Una de la iniciativas más recientes es el consorcio *Changing Place/House_n* que incluye al departamento de Arquitectura y el laboratorio *Media Lab*, los cuales están investigando conjuntamente cómo sería la casa del futuro en el laboratorio *PlaceLab* [150].

En Europa también se han abordado los *entornos activos* en el hogar. El proyecto *Accord* [9] es una iniciativa en la que participan el centro sueco *Institute of Computer Science* y la Universidad de Nottingham. El objetivo es desarrollar interfaces tangibles innovadoras que permitan controlar y configurar fácilmente los dispositivos de un hogar [147]. Otro proyecto interesante es *i-Dorm* [128] de la Universidad de Essex: un dormitorio inteligente que emplea tecnología basada en agentes para aprender de las acciones del usuario. Finalmente una iniciativa de un empresa nacional que también tiene un proyecto orientado al hogar es Fagor Electrodomésticos [126].

Finalmente, las tecnologías asistivas aplicadas al hogar es un área que se en-

cuentra en pleno desarrollo. Así, el proyecto Gator [138] de la Universidad de Florida busca desarrollar dispositivos inteligentes que el propio usuario pueda comprar, instalar y revisar sin la ayuda de un técnico. La parte experimental del proyecto consiste en una casa real denominada *Gator Tech Smart House*. A su vez, la Universidad de Bath —con apoyo de empresas privadas— ha desarrollado la casa *Gloucester Smart House* [272]. Este entorno ha sido acomodado especialmente para pacientes que sufren algún tipo de demencia. El objetivo consiste en crear un entorno que permita a estos pacientes vivir en sus propias casas sin necesidad de una asistencia personal. Se han desarrollado aplicaciones que entran dentro de la categoría de memoria contextual, y que permiten a los habitantes recordar qué tienen que hacer en un momento dado, o dónde dejaron determinado objeto.

Objetos ubicuos

El estudio de los *entornos activos* no se limita solamente al entorno en sí, sino que también incluye los componentes del mismo. Esta línea entra de lleno en el objetivo de la *Computación Ubicua* de modificar objetos cotidianos añadiéndoles capacidades de computación. Un proyecto pionero fue *CyberFridge* un frigorífico inteligente que pertenecía al proyecto *Domisilica* [4]. *Smart Desks*, del mismo departamento, es una aplicación para despachos inteligentes. El objetivo de este proyecto es desarrollar un escritorio que actúe como si fuera un buen ayudante de oficina. Así el escritorio debería conocer los hábitos del usuario, preferencias, sensaciones o simplemente recordar dónde pone las cosas. Para ello deberá ser capaz de analizar la expresión facial [98] o de analizar las acciones o gestos que el individuo realiza [55].

Uno de los centros más activos es el instituto de investigación alemán *Fraunhofer-IPSI* que tiene varios proyectos de objetos ubicuos en marcha dentro un proyecto común denominado *AMBIENTE*. Entre los proyectos relacionados con entornos inteligentes destacan *i-LAND* [257] y *Roomware* [258, 263], que incluye distintos objetos de la vida diaria que han sido dotados de capacidades de computación como puertas, paredes [109], sillas [188], mesas... La continuación de este proyecto es la propuesta *AmbientAgoras* [10] que se centra en presentar al usuario servicios e información dependientes del espacio que ocupa.

Interacción persona-entorno

Otro de los campos de investigación donde los *entornos inteligentes* se acercan más a la *Computación Ubicua* y que ha suscitado gran interés es conseguir dotar al entorno de interfaces no intrusivos que permitan al usuario una interacción natural. Los primeros trabajos en este área se llevaron a cabo en el laboratorio *MediaLab* del MIT, donde se desarrollaron desde el año 1991 diversos sistemas de reconocimiento de caras, expresiones y gestos que luego se integrarían en el proyecto *Smart Rooms* [207]. Otro proyecto pionero fue la *Intelligent Room* [43, 73] desarrollado por el grupo de Inteligencia Artificial del MIT. Esta habitación, que recibió el nombre de HAL, consistía en un entorno altamente interactivo provisto de interfaces multimodales que iban desde los convencionales teclado y ratón, hasta

diálogos en lenguaje natural. Actualmente, este proyecto continua en progreso bajo el nombre de *Aire* [11] dentro de la iniciativa *Oxygen*. El nexo de unión de esto dos primeros proyectos es el empleo de videocámaras como dispositivo de adquisición de contexto. Los trabajos de Pentland se basan exclusivamente en la imagen como entrada, mientras que en la *Intelligent Room* se mezclan diversos dispositivos, haciendo especial hincapié en la interfaz oral. Siguiendo el enfoque multimodal, el proyecto *Meeting Room Project* [245] del laboratorio *Interactive Systems* de la Universidad Carnegie-Mellon se centra en el seguimiento de la atención del usuario [255] en una reunión según su cara, cuerpo y discurso. Otros enfoques multimodales han usado como parte de la interfaz elementos como objetos de la vida cotidiana. Éstos han tenido como referente los trabajos en interfaces tangibles (véase el apartado 2.2) que se materializaron en entornos inteligentes en la *Ambient Room* [152] donde el espacio físico es un elemento más que permite la interacción entre el usuario y el mundo digital.

El empleo de interfaces audio-visuales ha despertado el interés de la comunidad científica. En la Universidad de San Diego el laboratorio de *Computer Vision and Robotics Research* ha desarrollado una oficina inteligente dentro del proyecto AVIARY [266]. En este entorno se integran micrófonos y videocámaras que permiten interaccionar y seguir a los usuarios a lo largo de toda la oficina. En la misma línea se encuentra I.A.E. (Intelligent and Aware Environments), también conocido como *Smart Spaces* [97], que consisten en espacios que han sido transformados en áreas de trabajo inteligentes donde cámaras, pantallas, micrófonos y otros sensores se han fusionado con ordenadores. Parte del anterior entorno es el proyecto *Ubiquitous Video and Audio* [99], cuyo objetivo es el desarrollo de entornos cotidianos, inteligentes e interactivos que utilicen sensores de vídeo y audio para percibir gente e interactuar de modo que puedan soportar sus actividades diarias. Finalmente, el NIST (National Institute of Standards and Technology) con los proyectos *SmartSpace* [107] y *Meeting Room* persigue crear una infraestructura para salas de reuniones sensibles al contexto que capture la actividad del usuario y responda en consecuencia. Sus investigaciones se centran en capturar la voz mediante *arrays* de micrófonos y en distribuir eficientemente la información capturada.

Entornos comerciales

La industria también ha mostrado un especial interés por los *entornos activos*. Una de las empresas emprendedoras ha sido Xerox, tal como se ha visto en el apartado 2.4.1. Otra iniciativa importante es el proyecto *EasyLiving* [47] de la empresa Microsoft. Dentro de este proyecto se ha desarrollado tecnología base para seguimiento y localización de personas empleando videocámaras. También se pueden encontrar entornos relacionados con el hogar como son la *Internet Home* [68] con un enfoque cercano a la domótica y el *HomeLab* [8] de Accenture. Este último se enmarca dentro de la iniciativa *Inteligencia Ambiental*. Este término fue acuñado por la empresa Philips [211] para designar una nueva metáfora que surge de la fusión de los logros obtenidos en las áreas de *entornos inteligentes*, *Computación Ubicua* y *Consciencia del Contexto*. Philips participa en la *Inteligencia Ambiental*

con su propio *HomeLab.*

Otros entornos

Otros espacios que también han sido empleados con éxito como *entornos inteligentes* han sido, una guardería *Kids Room* [38] del MIT, un aula, en el proyecto *Classroom 2000* [5] del Georgia Tech, o un laboratorio de biología [22].

2.5.2. Los sistemas ubicuos y los entornos activos en España

En España, existe una amplia tradición de investigación en cada una de las técnicas necesarias para la construcción de sistemas ubicuos y entornos activos (visión artificial, reconocimiento y análisis del habla, arquitecturas de agentes heterogéneos, interfaces de usuario...). En los últimos años se han incrementado los esfuerzos para trasladar estas técnicas al campo de la computación ubicua. El interés suscitado en la *Computación Ubicua* ha quedado reflejado con la celebración de dos eventos nacionales: un taller en *Computacion Ubicua e Inteligencia Ambiental* y el I Simposio sobre Computación Ubicua e Inteligencia Ambiental. El primero se celebró dentro del marco de la X Conferencia de la Asociación española para la Inteligencia Artificial que tuvo lugar en septiembre de 2003. El segundo formó parte del I Congreso Español de Informática (CEDI'2005), y en él se dieron cita la mayor parte de los grupos de investigación españoles que se encuentran investigando en este área.

Los trabajos realizados por el grupo CHICO de la Universidad de Castilla la Mancha en entornos ubicuos con proyectos como AULA funden los entornos educativos colaborativos con la computación ubicua [201, 42]. Un grupo con tradicción en entornos con realidad aumentada [247, 20] es el grupo grupo GRIHO de la Universidad de Lleida.

Dentro de las distintas tecnologías que hacen posible el desarrollo de sistemas ubicuos se pueden encontrar diferentes campos de investigación. Muchos de ellos tienen una amplia tradición en España, y han dado lugar a prestigiosos grupos de investigación. Dentro del marco de este trabajo se van a exponer sólo aquellos grupos que trabajan directamente en la adecuación de dichas tecnologías a la *Computación Ubicua.*

En tecnologías relacionadas con la capa física, un lugar relevante lo ocupa el estudio y fabricación de sensores. En este área se encuentran inmersos diversos grupos de investigación. Así, en el Instituto de Automática Industrial perteneciente al CSIC existen varios proyectos en marcha relacionados con el posicionamiento y localización empleando técnicas de ultrasonidos [53, 175]. También dentro de tecnologías de posicionamiento se encuentra el sistema UNIZAR [58] de la Universidad de Zaragoza que combina tanto radio frecuencia como ultrasonidos. Otra tecnología que se está explorando para dar servicios de localización es RFID[8]. El grupo MAmI de la Universidad de Castilla La Mancha investiga nuevas formas de

[8]Radio Frequency Identification

interacción implícita [41] empleando etiquetas RFID. Este grupo está interesado en pantallas que permitan visualizar información pública que varía dependiendo del contexto de los usuarios que se acerquen al panel [40]. En esta línea, también los grupos LFCIA y MADS de la Universidad de la Coruña trabajan en el desarrollo de un sistema capaz de proveer información a dispositivos móviles dependiendo de su ubicación [36], donde ésta se obtiene a partir de etiquetas RFID. Otras aplicaciones donde se emplea esta tecnología son las relacionadas con servicios de identificación, tanto de personas como de objetos. Un campo que está teniendo un gran impulso en los últimos años son las redes de sensores (véase el apartado 3.2.1). La aplicación de redes de sensores inalámbricas en *entornos activos* esta siendo estudiada dentro del ámbito de la telemonitorización domiciliaria por grupos de investigación de la Universidad Politécnica de Madrid [21].

El diseño de circuitos integrados es una disciplina sólidamente establecida en España. Varios grupos expertos en este área se están centrando en aplicar los conocimientos en este campo a los sistemas ubicuos. Así, el diseño de sensores inteligentes tiene cabida dentro del grupo de Microelectrónica de la Escuela Técnica Superior de Ingenieros Industriales [59]. Este grupo también ha liderado el proyecto WeMS (Weareble Micro Systems) dedicado a la investigación de dispositivos vestibles [224]. Siguiendo dentro del campo de los vestibles, hay que destacar los trabajos del Departamento de Ingeniería Electrónica de la Universidad Politécnica de Cataluña en baterías piezoeléctricas alimentadas por el propio cuerpo humano [113].

Existen varias plataformas que pretenden potenciar el uso de dispositivos móviles teniendo en cuenta que estos dispositivos presentan capacidades limitadas tanto de computación como de comunicación. Éste es el caso de la plataforma de objetos distribuidos *EMI2lets* [80] que facilita el desarrollo y despliegue de aplicaciones móviles sensibles al contexto. Un objetivo similar persigue la plataforma de agentes móviles basada en el lenguaje de comunicación FIPA [231] desarrollada en la Universidad Carlos III por el *PerLab* (Pervasive Computing Laboratory). Este mismo grupo trabaja en modelos de seguridad y autentificación para la *Computación Ubicua* [19]. Otro enfoque lo plantea la plataforma SENDA [184] del grupo Arco de la Universidad de Castilla La Mancha. Este grupo propone una implementación alternativa de CORBA[9] enfocada a dispositivos de muy bajo costo (sensores, actuadores, dispositivos móviles ...). Esta plataforma posibilita integrar estos dispositivos como objetos distribuidos, salvando las limitaciones computacionales que impone CORBA.

Un grupo con unos intereses similares a la propuesta de esta tesis es el Laboratorio de Sistemas Ubicuos (*LSUB*) de la Universidad Rey Juan Carlos de Madrid el cual también trabaja en entornos ubicuos. Este grupo ha desarrollado un sistema operativo denominado Plan B[10] [26] que permite la integración de dispositivos físicos como recursos virtuales basándose en la exportación de sistemas de ficheros. Todos los recursos disponibles se manejan como sí fueran ficheros de manera que se homogeneiza el acceso a los mismos. Esta abstracción es muy versátil ya

[9]Common Object Request Broker Architecture

[10]basado en el sistema operativo Plan 9 desarrollado en los laboratorios Bell

que permite interaccionar con el mismo interfaz tanto con dispositivos de control, gráficos y audiovisuales.

Otra plataforma relacionada con la propuesta de esta tesis es OCP[11] [194] la cual es una capa intermedia desarrollada por la Universidad de Murcia que permite gestionar y fusionar información contextual proveniente de diferentes fuentes. Dentro del desarrollo de esta plataforma también se propone un modelo de información contextual basado en una ontología. El modelo que se plantea en esta tesis se centra en una propuesta concreta de información contextual en el dominio de los *entornos inteligentes* mientras que el propuesto en OCP tiene un enfoque más general. Se podría ver este último como un meta-modelo que facilita la creación de modelos de información contextual.

Las tecnologías asistivas y los entornos activos coinciden en tres grupos de investigación pertenecientes a las universidades del País Vasco, Sevilla y Zaragoza, que vienen colaborando en diferentes proyectos orientados a desarrollar sistemas de computación ubicua para dar soporte a hogares inteligentes principalmente centrados a las necesidades de las personas con discapacidad y ancianas. De esta colaboración surgió el proyecto *DomoSilla* [2] que consiste en integrar una silla de ruedas inteligente dentro de un entorno domótico. Actualmente las mencionadas universidades se encuentran inmersas en el proyecto Heterorred I y II [3] que persigue el estudio y desarrollo de una red heterogénea de área local para la interoperabilidad y el acceso a servicios y comunicaciones inalámbricos. Otra visión de las tecnologías asistivas aplicadas a la *Inteligencia Ambiental*, la presenta el laboratorio MedICLab (Medical Image Computing Laboratory) de la Universidad Politécnica de Valencia. Este grupo combina técnicas de realidad virtual con la psicología clínica para el tratamiento de trastornos psicológicos [168].

El CEDINT (Centro de Dómotica Integral), adscrito a la Universidad Politécnica de Madrid, agrupa conjuntamente esfuerzos institucionales y empresariales para crear un ámbito de formación e investigación de nuevas tecnologías aplicadas a la domótica y la inmótica. Uno de los logros más relevantes de dicho centro lo constituye el proyecto MAGIC BOX, en el que se ha creado un casa inteligente que emplea únicamente energías sostenibles. Este proyecto se enmarca dentro del concurso Solar Decathlon, una competición internacional patrocinada por el Departamento de Energía de los Estados Unidos. También investigando en el campo del hogar inteligente, se haya el Departamento de Ingeniería Electrónica de la Universidad de Oviedo, el cual colabora activamente en el proyecto iDorm de la Universidad de Essex en la integración de controladores de lógica difusa [167] en electrodomésticos.

Otro factor que respalda la penetración de la *Computación Ubicua* en España es el creciente interés que tiene la industria en este área. En el ámbito nacional se pueden encontrar importantes empresas involucradas en proyectos de investigación relacionados con la *Computación Ubicua*. Dentro de los posibles campos dentro la *Computación Ubicua*, los *entornos activos* han atraído a varias empresas. Una de ellas es Fagor Electrodomésticos, que con el apoyo tecnológico de Ikerlan, ha desarrollado un entorno de inteligencia ambiental enfocado al hogar

[11]Open Context Platform

[126]. En esta aplicación la mayor parte de las acciones que se pueden realizar sobre los electrodomésticos se activan mediante la voz. La propia Ikerlan ha implementado una plataforma de servicios distribuidos para soportar aplicaciones móviles [269], en la línea de las plataformas anteriormente mencionadas. Otra de ellas, es TEKNIKER, que ha trasladado la *Inteligencia Ambiental* al entorno industrial desarrollando un laboratorio inteligente cuyo objetivo es facilitar la tarea a los operadores de taller que se encuentran trabajando en el mismo [261]. Cabe destacar también, que los *entornos activos* están teniendo repercusión dentro del mundo empresarial. Tanto proveedores de servicios como operadores de telefonía móvil han mostrado un enorme interés en la creación de servicios sensibles a la localización. Empresas como Telefónica Móviles, Amena, Vodafone, SOLUZIONA o Ericsson participan activamente y se posicionan como empresas precursoras e impulsoras de este tipo de servicios.

Finalmente, el proyecto *Interact* [15] de la Universidad Autónoma de Madrid, ha creado una arquitectura e infraestructura de integración que permiten la interacción natural con el *entorno activo* mediante el uso concurrente de módulos que implementan técnicas ya conocidas de interacción persona-ordenador. Esta tesis se ha desarrollado en el marco de este proyecto.

Capítulo 3

Arquitecturas para la computación ubica

3.1. Introducción

Según el estándar IEEE 1471 [149] publicado en el año 2000 por el organismo internacional IEEE [1] una arquitectura se define como:

> La organización fundamental de un sistema según sus componentes, las relaciones entre los mismos y el entorno, y los principios que guían su diseño y evolución.

Esta capítulo presenta los componentes de un *entorno inteligente* así como son las relaciones entre ellos. En el capítulo anterior se han repasado las aplicaciones ubicuas más relevantes que caen en las categorías de *entornos inteligentes* y *aplicaciones sensibles al contexto*, presentando a las primeras como un caso particular de las segundas. Se ha realizado un estudio de cada sistema informático como si de una caja negra se tratara. Este punto de vista permite visualizar los dominios de aplicación y los logros obtenidos en estos campos. Una vez determinado para qué sirven, el siguiente paso es describir cómo funcionan. Esta descripción se facilita si se sigue la estructura marcada por la arquitectura del sistema.

La organización fundamental que se ha elegido para describir un *entorno inteligente* consta de tres bloques o capas:

- **Capa física.** Es la más cercana al mundo real. Se encarga de capturar información del entorno a través de los sensores, y de transmitir las respuestas al usuario utilizando dispositivos audio-visuales y otros actuadores. También es función de esta capa distribuir la información, tanto de entrada como de salida, entre los distintos componentes del *entorno inteligente*. La capa física está integrada por una o más de redes de datos y un conjunto de dispositivos conectados a estas redes.

[1] Institute of Electrical and Electronics Engineers

- **Capa intermedia.** Esta capa se encuentra formada totalmente por componentes *software*. Su misión es presentar una visión uniforme a la capa de aplicación de todos los componentes de la capa física. Por regla general en la capa física se emplea más de una tecnología. Al tener que distribuir información de distinta naturaleza (información proveniente de los sensores, audio, vídeo...), no existe una única red capaz de dar un rendimiento óptimo en todas las situaciones. La capa intermedia no solamente encapsula la complejidad de los distintos dispositivos, sino que también añade información contextual, que aporta un mayor nivel de abstracción al incluir entidades y relaciones que no aparecen directamente en la capa física. Finalmente, además, incluye nuevas funcionalidades a las ofrecidas por la capa física.

- **Capa de aplicación.** Está compuesta por módulos, procesos, agentes... que utilizan la funcionalidad ofrecida por la capa intermedia para ofrecer servicios específicos. Esta capa se encuentra estrechamente relacionada con la interfaz de usuario.

El presente capítulo analiza el estado del arte de las dos primeras capas, ya que la capa de aplicación ha sido cubierta en el capítulo 2. Por un lado, en la capa física se hace hincapié en los componentes que permiten obtener el contexto y en las redes de comunicación que conectan a estos componentes. Por otro lado, la capa intermedia se trata más en detalle, realizándose un estudio de las diferentes capas intermedias disponibles actualmente. Este estudio va a servir para extraer las características que debe tener un capa intermedia que gestione el contexto producido en un *entorno inteligente*. Las conclusiones de este capítulo van a dirigir las decisiones de diseño tomadas en la propuesta de capa intermedia de esta tesis.

3.2. Capa física

Este apartado repasa las diferentes tecnologías ligadas a la capa física que son necesarias para implementar una *aplicación sensible al contexto*. Éstas se pueden agrupar en tres grandes bloques:

- **Adquisición.** En este bloque se incluyen los dispositivos que miden directamente el mundo físico. Consta de un conjunto variado de sensores, los cuales obtienen información muy detallada pero pobre en cuanto a abstracción. Además, hay que tener en cuenta que la información adquirida por los sensores puede ser poco fiable.

- **Comunicación.** Este bloque incluye diferentes tecnologías de redes de comunicación que se encargan de distribuir la información contextual recolectada del mundo físico.

- **Actuación.** Finalmente, este bloque incluye los dispositivos que permiten actuar sobre el mundo físico, los cuales se conocen comúnmente como actuadores.

3.2.1. Dispositivos para la adquisición del contexto

Los dispositivos más frecuentemente empleados en la adquisición del contexto se pueden clasificar en sensores, etiquetas y videocámaras. A continuación se detallan cada uno de ellos.

Sensores

Un sensor es un tipo de dispositivo que detecta un determinado evento físico, como un cambio en la temperatura, en la presión, etc. Estos eventos consisten normalmente en cambios en un determinado parámetro físico. La evolución de estas variaciones en el tiempo se denomina señal. Por tanto, los sensores se encargan de traducir fenómenos físicos de diversa índole a una representación común, habitualmente a una señal eléctrica. La salida de los sensores es distribuida a los procesadores, que pueden ser analógicos o digitales. Nuestra propuesta se limita a estos últimos, por lo que se requiere un proceso de discretización que convierta la señal eléctrica continua en una señal digital.

A continuación se describen aquellos que se emplean más frecuentemente en un *entorno activo*, atendiendo a su funcionalidad:

- **Contacto.** Dentro de este grupo se encuadran aquellos sensores que permiten detectar los estados de abierto o cerrado dependiendo si hay o no contacto. Pueden consistir en una célula fotoeléctrica y un emisor que detecta cuándo se corta el haz, o en un electroimán formado por dos polos que diferencia cuándo los polos están unidos o no. Se emplean para detectar la apertura de puertas o el cruce de objetos por determinados puntos.

- **Gases.** Determinan si ha habido fugas de gas en el entorno.

- **Luminosidad.** Informan sobre la luminosidad de un entorno. Junto con otros sensores pueden ser útiles para detectar la actividad del usuario, si está durmiendo o viendo la televisión, por ejemplo. Se implementan habitualmente mediante fotoresistencias que son dispositivos que varían su resistencia en función de la luz recibida.

- **Humedad.** Estos dispositivos permiten detectar el nivel de humedad de un entorno. Una variante más interesante de estos sensores permite medir la humedad de la piel. Éstos son conocidos por sus aplicaciones en detectores de mentiras. La humedad de la piel se mide en función de las variaciones en la conductividad de la piel. Este tipo de sensores se ha empleado para determinar el interés de un sujeto por un objeto, de forma que se activa una vídeo cámara [136] cuando éste es elevado, y se desactiva en caso contrario.

- **Incendio.** Permiten detectar un incendio en el entorno. Comúnmente se emplea la combinación de detectores de calor junto con detectores de humo.

- **Inundación.** Detectan fugas de agua. Se colocan cercanos al suelo en entornos sensibles a este tipo de accidentes como cocinas o cuartos de baños.

- **Lluvia-Nieve.** Detecta la presencia de lluvia o nieve en el exterior de un hogar.

- **Movimiento.** Sencillos y fáciles de instalar, permiten detectar el movimiento en una habitación. Aunque no suelen ser muy fiables, aportan una primera estimación de la presencia en un entorno. Normalmente se basan en ultrasonidos o infrarrojos.

- **Presión.** Miden la presión ejercida sobre ellos. Una combinación de varios de estos dispositivos se ha utilizado para inferir la identidad de una persona por cómo se sienta [145] o cómo pisa [200].

- **Rotura de cristales.** Se emplean en ventanas como primera medida de seguridad para detectar si el cristal se ha roto.

- **Sonido.** Se usan para descubrir fuentes sonoras. Una versión más sofisticada de estos sensores permite clasificar el tipo de sonido. Se han empleado combinados con sensores de aceleración [169] para reconocer actividades manuales tales como serrar, martillear, atornillar...

- **Temperatura.** Permiten medir la temperatura de una habitación o de un objeto.

- **Tensión eléctrica.** Estos sensores detectan cambios en una tensión eléctrica. Se pueden utilizar en diversas aplicaciones como detectar caídas de tensión, llamadas de teléfono, detectar si un aparato está alimentado, comprobar si una señal de vídeo o audio está siendo emitida...

- **Velocidad y aceleración.** Estos últimos son los más populares para la detección de actividad. Un buen ejemplo de cómo detectar diferentes actividades se describe en [28]. También se pueden emplear para detectar que dos dispositivos son portados por la misma persona [166].

La colocación de un sensor en un *entorno activo* puede efectuarse de varios modos. Éste puede ser parte del entorno, como un sensor de luminosidad o de temperatura, también puede estar ligado a un dispositivo, como un teléfono móvil sobre el cual se quiere medir su orientación, o puede ser llevado por una persona, como los acelerómetros que detectan la actividad del portador.

La proliferación de distintos tipos de sensores y los avances en los procesos de fabricación, que permiten obtener sensores de escalas milimétricas [141, 283], han propiciado el estudio de redes de sensores conectadas por tecnologías inalámbricas [127]. Estas nuevas tecnologías, que ya se están comercializando, permiten distribuir una gran cantidad de sensores de bajo consumo que posibilitan muestrear espacialmente una o más variables contextuales. Por ejemplo, siguiendo esta línea, en el proyecto *Home_n* del *MIT Media Lab* se está empleando una combinación de miniaturización de sensores y etiquetas activas. En este proyecto, cada objeto de la casa cuenta con una etiqueta activa. El ocupante porta un diminuto

lector de tarjetas, de tal forma que cada vez que se acerca a un objeto, queda registrado el evento. Posteriormente, se extraen patrones de comportamiento según las secuencias de objetos a los que se ha ido aproximando [265].

En el otro extremo a estos diminutos sensores se encuentran los artefactos activos, utensilios de la vida diaria que están dotados de capacidades de computación. En esta línea se pueden recopilar en la literatura diferentes tipos de objetos como mesas [246], puertas y paredes [109], sillas [188], tazas de café [32], guantes de cocina [164], etc.

Ambas tecnologías permiten capturar el contexto de manera no intrusiva. Los primeros porque consiguen hacerse invisibles por su tamaño; los segundos porque se funden en la vida cotidiana del usuario.

Etiquetas

Las etiquetas [2] son también sensores, aunque el modo de funcionamiento cambia respecto a los sensores presentados en el apartado anterior. Una etiqueta es una marca que se coloca en un objeto o en un lugar de forma que se dispone de uno o varios detectores capaces de localizarla. Si la etiqueta está sobre un objeto móvil se puede detectar mediante varios detectores cuándo se mueve el objeto y qué trayectoria sigue. Recíprocamente, se puede averiguar cuándo un detector se acerca a un objeto o lugar que tenga un etiqueta. Otra aplicación consiste en identificar objetos dotando a cada etiqueta de una identificación unívoca.

Una de las primeras ideas que surgió en dentro de los sistemas sensibles al contexto fue la propuesta de etiquetas activas [278] [3]. Éstas son dispositivos que emiten periódicamente una señal por radiofrecuencia. Un conjunto de balizas estratégicamente colocadas recogen la señal y localizan al portador mediante técnicas de triangulación. Cada etiqueta se corresponde unívocamente con un usuario, por lo que —aparte de la localización— se consigue la identificación. A raíz de este idea han surgido varios proyectos similares que obtienen más información contextual, aparte de la localización y la identidad, a base de emplear etiquetas activas más complejas. De esta tesitura son los proyectos *Smart Badge* [251] y *Sensor Badge* [100].

Una tecnología que está demostrando ser bastante prometedora son las etiquetas pasivas. En oposición a las etiquetas activas, las pasivas no requieren fuente de alimentación, sino que el contenido de cada etiqueta se lee empleando un dispositivo activo con una antena que emite una señal en radiofrecuencia. Estas etiquetas se colocan en objetos, lugares y/o personas. Esta tecnología se emplea en librerías, almacenes, control de acceso de edificios, identificación de maletas ... en general en cualquier área que necesite el seguimiento e identificación de objetos.

Uno de los inconvenientes de las etiquetas pasivas es el volumen de la antena que se precisa. Aun así, en un futuro se prevé que el tamaño de los lectores se reduzca drásticamente [4]. Partiendo de esta premisa se está estudiando la posibilidad reconocer la actividad de una persona según se acerca a determinados objetos

[2] *Badges*, aunque recientemente también se puede encontrar traducido por pegatinas
[3] ActiveBadge
[4] Hasta el tamaño de un reloj de pulsera

[210]. Para ello se dotaría a todos los objetos de una etiqueta pasiva y a la persona de un lector fácilmente portable.

Videocámaras

Finalmente, un dispositivo bastante versátil para capturar contexto son las cámaras de vídeo. Éstas se han utilizado en identificación de personas mediante reconocimiento de caras [208], detección del foco mirada [78], seguimiento de la persona dentro del entorno [72] y detección de la actividad [180]. Dos son los problemas que implican la captura de vídeo: por un lado la gran capacidad de cómputo que se precisa para extraer la información relevante a partir de las imágenes; por otro lado el despliegue de cámaras choca frontalmente con la privacidad del usuario, siendo uno de los métodos más intrusivos.

3.2.2. Redes de comunicación

El segundo gran bloque dentro de la capa física lo forman las redes de comunicaciones. Un *entorno activo* tiene que estar soportado por una red de comunicaciones que permita la interacción entre los dispositivos que lo componen. Estas redes se encargan de distribuir la información entre los componentes del entorno. A la información que fluye a través de una red de comunicaciones se le denomina tráfico.

Se va a estudiar un subconjunto de todas las posibles redes de comunicaciones. De nuevo, se va a exigir una digitalización de la información que se transporta. Esto elimina del estudio los sistemas analógicos que históricamente han acaparado los medios de difusión de voz, vídeo y audio; léase telefonía, televisión y radio.

Dentro de los sistemas de comunicación digitales se analizarán aquellos que están pensados para desplegarse en un espacio limitado por un entorno. En esta categoría entran las redes de área local (LAN)[5] cuyo alcance abarca unos centenares de metros. Casos particulares de LAN son las redes SOHO [6] que en castellano se conoce como oficina virtual. Son redes orientadas a pequeñas oficinas, como pueden ser las instaladas dentro de una vivienda. En relación a las redes que se instalan en un hogar se ha acuñado el termino en inglés *Home Networking*, que en castellano se puede encontrar traducido por *Redes Domésticas*, y que se refiere al conjunto de todas las infraestructuras digitales que se despliegan en un hogar para comunicar los dispositivos y electrodomésticos entre sí y con el exterior.

Como ya se ha comentado (véase el apartado 1.1) en esta tesis se ha tomado el hogar como entorno de referencia. En el ámbito de las redes de comunicación desplegadas en un hogar se distinguen dos posibles clasificaciones: bien atendiendo a la naturaleza del tráfico distribuido o bien atendiendo al medio físico en el que se implementan. La primera clasificación divide las redes según el tráfico que distribuyen. Por un lado, existe un tipo particular de tráfico que se denomina continuo. Éste requiere que la información que se distribuye se consuma instantáneamente. Para ello, el retardo entre dos fragmentos consecutivos de información no puede

[5] Local Area Network
[6] Small Office/ Home Office

superar un máximo establecido; si se supera este retardo se afecta a la calidad de la información recibida. La voz o el vídeo en tiempo real son ejemplos de tráfico continuo. Las redes que son capaces de distribuir este tipo de tráfico continuo se van a llamar redes multimedia.

Otra restricción a tener en cuenta es que el tráfico multimedia necesita un ancho de banda mínimo para poder transmitirse. Éste además es bastante elevado, especialmente en el caso del vídeo, y aumenta cuanto mayor sea la calidad exigida. En la tabla 3.1 se muestran —tanto para audio como para vídeo— los diferentes anchos de bandas mínimos requeridos para distintas calidades. Para el caso de la señal de vídeo, estas redes tienen que proveer de altas velocidades de transmisión (> 2 Mbps).

Contenido	Ancho de Banda requerido
Audio	
Calidad CD	706 Kbit/s (44100 muestras/s, 16-bit x muestra)
Calidad telefonía digital	64 Kbit/s (8.000 muestras/s, 8-bit x muestra)
Vídeo	
Calidad mínima	566 Kbit/s (1024x768, 30 marcos/s, 3 color, 8-bit x color)
Calidad TV	
Sin comprimir	96 Mbit/s
HDTV	19 Mbit/s
DVD	3-8 Mbit/s
MPEG-1	1.5 Mbit/s
MPEG-2	6 Mbit/s

Tabla 3.1: Clasificación de distintos tipos de tráficos multimedia según el ancho de banda requerido

Por otro lado, las redes que no están especializadas en tráfico continuo se van a denominar con el nombre genérico de redes de datos. Estas redes pueden transportar cualquier tipo de tráfico, siempre y cuando la tecnología base soporte la tasa binaria requerida. Se pueden emplear para distribuir tráfico continuo, pero a diferencia de las redes multimedia, no se asegura que la calidad de servicio se mantenga.

Dentro de las redes de datos, se van a distinguir la redes de control. La filosofía de diseño proviene del campo industrial, en el cual se precisa enviar mensajes cortos y de forma asíncrona, donde el tráfico no requiere una alta velocidad de transmisión (<9600 bps). El diseño de los protocolos y la fabricación de los dispositivos se ajusta a este tipo de tráfico lo que permite ahorrar coste de implementación. Los protocolos de control típicamente no tienen el ancho de banda necesario como para poder cursar tráfico multimedia.

Por otro lado, la infraestructura de las redes multimedia es demasiada cara como para que sean utilizadas también como redes de control. Esto da lugar a que, por el momento, existan dos redes en paralelo dentro de la vivienda automatizada.

La segunda clasificación atiende al medio físico que se emplea para transportar la información. Así se puede hablar de:

- **Redes cableadas.** El medio físico que se utiliza es algún tipo de cable ya sea de cobre u óptico.
- **Redes inalámbricas.** En este caso la información se transporta por ondas electromagnéticas a través del aire.

La tecnología inalámbrica aporta una serie de ventajas frente al cable que se pueden resumir en dos:

- **Movilidad.** Cuesta imaginarse nodos móviles que estén de algún modo cableados. El conjunto de aplicaciones que requieren terminales móviles se encuentra vetado al cable. La posibilidad de poder conectar cualquier dispositivo independientemente de su situación geográfica es una de las claves para conseguir una verdadera computación ubicua.
- **Flexibilidad.** Al contrario que las infraestructuras cableadas, que son muy costosas de reconfigurar, la configuración de las redes inalámbricas es mucho más sencilla de modificar.

Los principales problemas que presenta una red inalámbrica son consecuencia del medio físico empleado. La señal cuando viaja por el espacio libre es más propensa a sufrir interferencias que cuando viaja por un cable. Esto afecta a la calidad del servicio. La disponibilidad de la conexión también se ve perjudicada por la aparición de obstáculos al desplazarse los terminales. Otro punto importante es la seguridad: capturar una señal inalámbrica es más sencillo que capturar una señal confinada en un cable. Finalmente, los terminales tienen que depender de baterías al no disponer de cables de alimentación. La autonomía de una batería es otro punto débil que afecta a la disponibilidad.

Redes de control

En la actualidad, existen diversas opciones disponibles para desplegar una red domótica. Hay que tener en cuenta que no es previsible que se imponga un único estándar en un futuro próximo, sino que el entorno más probable será uno donde convivan múltiples tecnologías. En Estados Unidos habitualmente se ha utilizado X10 [298], un protocolo propietario muy sencillo de implementar y relativamente económico, pero con serias limitaciones. Actualmente, junto a X10, pugnan por el mercado dos nuevos estándares: LonWork [89] y CeBUS [24]. Tampoco en Europa ha habido un claro ganador, habiéndose realizado la mayoría de las instalaciones domóticas de forma ad hoc. Sin embargo, en los últimos años han surgido diversos buses con proyección europea, destacando EIB [114], EHS [23] y Batibus [31]. EHS fue inicialmente propuesto por la empresa Trialog, y posteriormente adoptado por la EHSA (European Home System Association). Actualmente no existen apenas instalaciones reales realizadas con EHS. Batibus es un bus de control industrial

que ha sido utilizado para entornos domóticos pero que, debido a sus carencias y obsolescencia, cada vez es menos usado. EIB es la propuesta de carácter más moderno y que muestra más esperanzas de prosperar en Europa. Recientemente estas tres tecnologías se han unido bajo el consorcio Konnex, para así trabajar en la interoperatibilidad de los tres sistemas, que tendrá como resultado en el futuro un único bus.

El estándar EIB (European Instalation Bus) ha sido propuesto por la EIBA (European Installation Bus Association). La EIBA es la organización que reúne a empresas europeas de instalación eléctrica europea para impulsar el desarrollo de sistemas de control de edificios y conseguir ofrecer en el mercado europeo un sistema único de alta fiabilidad.

Este estándar permite controlar, comunicar y vigilar las funciones de servicio dentro de un edificio utilizando una única línea. Básicamente se trata de un bus en serie por donde se transmiten informaciones desde los elementos sensores hasta los actuadores. La información se transmite modulando una señal continua que también sirve como alimentación. Utiliza un protocolo CSMA-CA similar al de las redes Ethernet. Dada las características físicas de este bus, sólo puede ser empleado para aplicaciones de bajo ancho de banda.

Redes multimedia

Las redes multimedia que se desarrollen para ambientes domésticos en un futuro próximo deberán permitir la comunicación entre PC y PC así como con otros periféricos, por ejemplo impresoras. Estas redes permitirán que varios PCs de una casa puedan compartir un enlace de banda ancha con un proveedor de Internet y también deberán ser capaces de transferir contenidos multimedia y otros servicios de banda ancha entre los diferentes dispositivos existentes en los hogares. Estos nuevos dispositivos digitales pueden ser escáneres, reproductores de DVD, sistemas DBS (Digital Business System), cámaras digitales, teléfonos celulares y agendas electrónicas, que a su vez deberán ser compatibles con otros productos analógicos como pueden ser TV y reproductores de vídeo VCR. Hay que tener en cuenta que los dispositivos electrónicos que actualmente trabajan como periférico de un PC, en un periodo de dos o tres años se venderán como equipos independientes directamente conectables a una red. En este entorno serán viables múltiples configuraciones, como por ejemplo indicar qué monitor o televisor será empleado como dispositivo de salida para un equipo multimedia. Por otra parte, empezarán a ser habituales en los hogares nuevos servicios, tales como la recepción de música en diferentes formatos digitales, MP3, WAV... competiciones de juegos en directo, vídeo bajo demanda o videoconferencia. En general todas estas aplicaciones requieren una red con un ancho de banda grande con velocidades superiores a 1 Mb/s.

Otras aplicaciones que se están desarrollando necesitan un ancho de banda menor tales como dispositivos manos libres, terminales de Internet y teléfonos para Web (Voz sobre IP). En estos casos las velocidades de transmisión por la red no necesitan ser mayores que las actualmente empleadas por los módem telefónicos 56, 64 o 128 kB/s.

Un ejemplo de estándar diseñado para soportar tráfico multimedia es IEEE 1394 [149], también conocido como *FireWire*. Éste consiste en un bus de datos serie a alta velocidad que puede transportar grandes cantidades de datos entre computadoras y periféricos. Sus principales ventajas son que tiene un cableado sencillo, una velocidad de trasferencia de hasta 400 Mb/s y un mecanismo de control que permite que los equipos puedan ser conectados y desconectados sin necesidad de eliminar la alimentación de la red eléctrica. Hoy en día muchos dispositivos de electrónica de consumo emplean este estándar para transferencia de video. Esta tecnología funciona correctamente para conectar pocos dispositivos, pero no escala adecuadamente ya que las distancias de transmisión son cortas.

Un enfoque distinto sería emplear una red de datos tradicional como espina dorsal para transportar el tráfico multimedia. Estas redes no disponen de mecanismos para el control de la calidad de servicio que requiere el tráfico continuo, con lo que el problema se traslada a los protocolos de niveles superiores. Éste sería el caso de emplear una red Ethernet. Éste es un protocolo de área local desarrollado por la empresa Xerox Corporation en cooperación con la DEC e Intel en 1976 [179]. Ethernet es el nombre comercial que se estableció para denominar el protocolo. Aunque se ha comprobado el éxito notable de Ethernet en el sector empresarial, sin embargo queda por dilucidar cuál puede ser su futuro como red de tráfico multimedia en un hogar. El estándar parte con una experiencia de más de veinte años de desarrollo que lo han consolidado como una tecnología de alta fiabilidad y elevado rendimiento. Además se pueden destacar tres importantes ventajas:

- Está soportado por un estándar internacional y ampliamente reconocido como es IEEE 802.3.
- Al estar ampliamente soportada la interoperatibilidad de Ethernet con IP, queda fácilmente resuelta la pasarela entre la red de interna de la vivienda y la conexión a Internet.
- Permite velocidades de transmisión desde 10 Mbps a 10 Gbps, lo cual la hace especialmente atractiva para transportar tráfico multimedia, pudiendo servir de espina dorsal (backbone) para otras redes más lentas.

En la parte negativa se encuentra, por un lado, que el precio de la instalación es elevado. Esto es debido al cableado que utiliza, que requiere cables de alta calidad. Además, como los hogares no suelen poseer la infraestructura básica, casi siempre hay que desplegar la red desde cero. Por otro lado, la necesidad de concentradores aumenta el precio de la instalación.

Siguiendo la misma línea, otro estándar aplicable es HomePNA [142] (Home Phoneline Networking Alliance). Éste se basa en utilizar los actuales cableados telefónicos existentes ya en los hogares. Mediante esta tecnología se pueden comunicar dispositivos a velocidades de 1 Mb/s sin necesidad de añadir concentradores, filtros u otros elementos. Además el uso de la red es compatible con el uso simultáneo del sistema telefónico. HomePCA esta basado en protocolos Ethernet que utilizan como medio de acceso CSMA. Se está implementando una versión compatible con la actual que permita velocidades de 10 Mb/s. Empresas como

3Com, AT&T, Hewlett-Packard, IBM, Intel y Lucent forman parte de esta alianza.

Redes inalámbricas

Las redes inalámbricas tienen como principal ventaja la ausencia de cableado, lo que permite la movilidad de los dispositivos. En las redes inalámbricas de área local se emplean dos tecnologías: infrarrojos y radiofrecuencia. La tecnología infrarroja se caracteriza por utilizar la banda de infrarrojos en el espectro radioeléctrico y necesitar que los terminales tengan una línea de visión sin obstáculos para poder comunicarse. Esta restricción limita bastante sus aplicaciones. La redes inalámbricas por radiofrecuencia utilizan bandas de espectro superiores en longitud de onda; habitualmente se emplea la banda de 2.4 GHz, ya que esta banda está sin asignar a escala mundial.

Los tres estándares más importantes en tecnología inalámbrica para redes de área local emplean radiofrecuencia. Éstos son HomeRF, Bluetooth y IEEE 802.11. Cada uno de ellos está enfocado a solucionar distintos problemas.

La misión de *HomeRF* [143] es conseguir la interoperatividad entre en el mayor número de dispositivos diferentes que estén ubicados en cualquier punto de un hogar. HomeRF pretende posicionarse como el estándar inalámbrico para redes domóticas. Para ello establece un estándar abierto y sin licencia basado en una comunicación digital mediante radiofrecuencia. El resultado ha sido el desarrollo de SWAP (Shared Wireless Access Protocol). HomeRF está compuesto de unas cuarenta empresas de las que el núcleo duro son Compaq, Ericsson Enterprise Networks, Hewlett-Packard, IBM, Intel, Microsoft, Motorola, Philips Consumer Communications, Proxim y Symbionics. Una parte de las especificaciones de *HomeRF* han sido adoptadas por el consorcio *Zigbee* [18]. El objetivo de *Zigbee* son las redes de radiofrecuencia de bajo coste, donde se incluyen tanto dispositivos de control como sensores.

Bluetooth está centrado en dispositivos móviles como teléfonos y agendas. Bluetooth surge como una propuesta de un consorcio de fabricantes de telefonía móvil en busca de un estándar de comunicaciones inalámbricas para datos y voz. La tecnología Bluetooth se basa en radioenlaces de corto alcance. Soporta tanto redes basadas en infraestructura como redes ad-hoc, siendo en éstas últimas donde está alcanzando su máxima difusión. Bluetooth, al igual que HomeRF, opera en la banda de 2,4 GHz. Bluetooth se está enfocando a lo que se denomina redes inalámbricas personales (WPAN)[7]. Éstas tienen como función conectar fácilmente dispositivos que se encuentran próximos entre sí, habitualmente portados por una o varias personas.

Finalmente, IEEE 802.11[8] [296] apunta por ser la solución más robusta y fiable, por lo que también la más costosa. IEEE 802.11 es una de las soluciones inalámbricas más completas y potentes que existe en el mercado. El estándar trata de abarcar un conjunto amplio de escenarios, independientemente de que las estaciones que formen la red sean fijas, portátiles o móviles. IEEE 802.11 se

[7]Wireless Personal Area Network
[8]Tecnología WiFi

ha convertido en estándar de facto para redes de datos inalámbricas gracias a sus prestaciones.

3.2.3. Actuadores

El tercero de los bloques que conforman la capa física lo ocupan los actuadores. Mientras que los sensores permiten detectar cambios en un determinado parámetro físico, los actuadores se encargan de modificar el mundo físico a voluntad del usuario o del sistema. En este apartado sólo se van a tener en cuenta aquellos mecanismos que se pueden accionar electrónicamente. Tampoco se va a entrar en detalle en la descripción de los mismos, ya que el objetivo de este trabajo apunta a la captura y distribución de información contextual. Ahora bien, dado que los actuadores forman parte de los entornos activos, se requiere una reseña de este tipo de dispositivos.

Se han identificado distintos tipos de actuadores. Uno de los actuadores que se emplea con más frecuencia son los relés. Éstos se pueden encontrar individualmente o como parte de otros actuadores más complejos. Un relé consiste en un interruptor electrónico que permite controlar el paso de corriente a través de un determinado circuito. El interruptor se activa o se desactiva dependiendo de una tensión de control. Una ventaja que aportan los relés es la separación entre la tensión accionadora y la tensión que atraviesa el relé, de forma que se pueden manejar elevados voltajes con tensiones de control pequeñas. Entre las funcionalidades de un relé se puede encontrar la de encender o apagar dispositivos conectados a la red eléctrica dependiendo de una señal eléctrica de control. Esta señal puede ser generada como resultado del procesamiento de las entradas del sistema. Otra funcionalidad a destacar consiste en la posibilidad de cambiar electrónicamente las conexiones de un circuito. Así, por ejemplo, se podría conmutar la salida de circuito de audio que conectara dos altavoces con una cadena musical para que el audio saliera por un altavoz u otro.

El siguiente actuador, que se ha denominado regulador, se podría describir como un tipo relé. Mientras que los relés funcionan a modo de interruptores electrónicos, es decir, realizan un control de todo o nada, los reguladores permiten modificar de forma continua la corriente. Por ejemplo, se emplean para controlar la emisión de luz de una lámpara. Además, un regulador puede incluir una etapa de potencia que amplifique la señal, como por ejemplo, el amplificador de un altavoz.

Así como relés y reguladores actúan sobre una corriente eléctrica, el siguiente actuador, denominado electroválvula, permite controlar el paso de una corriente de agua o gas. Se instalan en las tuberías para cortar o abrir automáticamente el flujo en caso de emergencia.

Existen otros tipos actuadores que permiten accionar objetos físicos. Dentro de esta categoría se encuentran los servomecanismos y motores de continua. En los *entornos activos* estos dispositivos permiten accionar los diferentes componentes mecánicos. Así, estos actuadores permiten automatizar la apertura y cierre de puertas, persianas y cortinas, la orientación de cámaras de video o de altavoces, etc.

Hasta ahora se han identificado componentes individuales. También es interesante disponer de una visión más general de las posibilidades de los actuadores. Para ello interesa analizarlos desde un perspectiva holística en la que se incluyan dispositivos y sistemas completos. Actualmente, están comenzando a aparecer las primeras gamas de electrodomésticos *inteligentes*. Estos dispositivos incluyen sistemas embebidos que permiten controlarlos remotamente. Este control va más allá del encendido y apagado automático que realiza un relé, ya que toda la funcionalidad del dispositivo está disponible. Frigoríficos, lavadoras, lavavajillas, robots de cocina, cafeteras, hornos, y un sin fin de electrodomésticos, están o estarán disponibles para poder accionarse automáticamente. Finalmente, un conjunto de actuadores y sensores que en el campo de la domótica reciben un tratamiento separado lo forman el sistema de calefacción y aire acondicionado. En inglés se denomina con el término HVAC[9], e incluye a todos los dispositivos que se encargan de mantener unas condiciones climáticas adecuadas.

3.3. Capa intermedia: Plataformas *software* para el contexto

En la organización fundamental que se propone en la introducción del capítulo (véase el apartado 3.1) el segundo bloque lo ocupa la capa intermedia, que sirve como nexo de unión entre la capa física y la capa de aplicación. Para el desarrollo de una aplicación no es imprescindible contar con una capa intermedia; la conexión entre la aplicación y el mundo físico se puede resolver directamente en la capa de aplicación. El resultado es una aplicación muy dependiente de la tecnología de sensores disponible. La modificación de este tipo de aplicación es compleja, ya que está estrechamente ligada a los sensores que se eligieron en el diseño, y al partir de cero en la implementación, los costes son mayores. Por este motivo, en el desarrollo de aplicaciones distribuidas se prefiere confiar en un capa intermedia. Así se comprueba en el apartado 2.4.1, donde la mayor parte de las aplicaciones descritas se sostienen sobre una capa intermedia que facilita la recopilación y tratamiento de la información contextual, y que permite desacoplar a las *aplicaciones sensibles al contexto* del mundo físico.

Una capa intermedia aporta un conjunto de servicios que facilitan la programación de nuevos sistemas distribuidos así como la integración de sistemas ya existentes. Las capas intermedias orientadas a objetos como CORBA o DCOM son buenos ejemplos de cómo funciona este tipo de tecnología. El interés de esta tesis se centra en proponer una capa intermedia para la gestión de información contextual en entornos inteligentes, por lo que se van a estudiar capas intermedias para la gestión del contexto, y capas intermedias empleadas en *entornos inteligentes*. Para facilitar el análisis se establecen varias clasificaciones de las capas intermedias que se pueden encontrar.

Atendiendo al modelo de programación las capas intermedias se pueden dividir en:

[9]Heating, Ventilating, and Air-Conditioning

- **Orientadas a servicio.** Estas plataformas se basan en el modelo cliente-servidor. Las *aplicaciones sensibles al contexto* acceden al contexto a través de una capa de servicios que define una interfaz estándar de comunicación. A los clientes se les proporcionan facilidades para descubrir los servicios disponibles y para crear nuevos servicios mediante la agregación de varios.
- **Orientadas a datos.** Estas capas intermedias se centran en la representación del contexto. La distribución se realiza mediante un conjunto reducido de operaciones.

Los mecanismos de distribución de la información se pueden clasificar según el acoplamiento en dos dimensiones: temporal y espacial. Siguiendo una clasificación similar a la que propuso Cabri [51] para modelos de coordinación, los mecanismos de comunicación se pueden dividir en:

- **Acoplado temporal y espacialmente.** Un proceso se comunica únicamente si conoce al receptor y coincide en el tiempo. Éste es el caso de una comunicación directa empleando algún mecanismo de llamada a procedimiento remoto.
- **Acoplado temporalmente y desacoplado espacialmente.** Los procesos se agrupan temporalmente para comunicar la información pero no requieren conocerse unos a otros. Sería el caso de un mecanismo editor/suscriptor basado en eventos.
- **Desacoplado temporalmente y acoplado espacialmente.** El proceso emisor debe conocer a quién envía los datos, pero no se requiere que ambos estén simultáneamente activos. Los mecanismos basados en colas de mensajes son un buen ejemplo de este tipo de comunicación.
- **Desacoplado temporalmente y espacialmente.** Los procesos no conocen a quién envían el mensaje ni tienen por qué coincidir temporalmente. La comunicación se realiza mediante algún mecanismo de compartición de forma que los procesos emisores dejan la información en un contenedor que es recogida de forma anónima por los receptores. La comunicación generativa [110] y las arquitecturas pizarra pertenecen a este última categoría.

Winograd [292] clasifica las capas intermedias según la tecnología que se escoja para su implementación:

- **Objetos distribuidos.** La entidad básica es un objeto, el cual puede ser activo o pasivo. Los primeros se asocian con fuentes de información contextual que recopilan el contexto del entorno, y que se pueden consultar mediante algún mecanismo de llamada a procedimiento remoto. Estas plataformas se asemejan a las propuestas de objetos distribuidos de CORBA y DCOM.
- **Infraestructura.** La plataforma se basa en una infraestructura de servidores conocidos, fiables y de acceso público que proveen de una serie de servicios que permiten a los clientes adquirir el contexto. Esta solución es similar a la propuesta por las tecnologías que implementan la red Internet.

- **Pizarra.** Se emplea un mecanismo centralizado en el que las *aplicaciones sensibles al contexto* almacenan y recuperan la información contextual de un repositorio común conocido y accesible mediante un sencillo protocolo de comunicación. Este tipo de plataformas se asemejan a los espacios de datos compartidos como *JavaSpaces* o *TSpace*.

Finalmente, se pueden también clasificar las capas intermedias según el objetivo que persiguen:

- **Orientadas a la adquisición y procesamiento del contexto.** Estas plataformas se centran en la capa de adquisición y en parte de la capa de interpretación. Proveen de mecanismos para estandarizar la comunicación con las diferentes tecnologías de sensores, así como para fusionar los datos obtenidos de varios sensores. Dependiendo de los casos realizan algún tipo de procesamiento que permite obtener una representación más abstracta de los datos obtenidos.

- **Orientadas a la distribución.** Estas capas intermedias se marcan como objetivo conseguir una distribución eficaz del contexto a todos los componentes del sistema. Para ello tratan de homogeneizar la información proveniente de la capa de adquisición y distribuirla mediante alguno de los paradigmas vistos anteriormente.

3.3.1. Capas intermedias orientadas a la *Computación Ubicua*

Dentro de esta apartado se van a describir capas intermedias que se han desarrollado para facilitar el desarrollo de sistemas ubicuos. Se incluyen tanto capas creadas a partir de proyectos de investigación como plataformas comerciales actualmente disponibles en el mercado. Se han ordenado las capas intermedias por orden alfabético.

Cooltown

CoolTown [160] es una iniciativa de los laboratorios de Sistemas Móviles e Internet pertenecientes a las empresa HP[10]. Este proyecto propone un modelo basado en la Web para escenarios de computación móvil en los cuales los usuarios requieren mantener la conectividad a medida que se desplazan por diferentes entornos. En *Cooltown* se combinan la tecnología Web, las redes inalámbricas y los dispositivos móviles.

La propuesta se basa en asociar a cada objeto físico y lugar una URL. Los dispositivos móviles disponen de *sensores* que les permiten recolectar URLs a medida que se aproximan a los objetos. La URL de cada objeto apunta a una página web a través de la cual tanto usuarios como aplicaciones pueden interaccionar. Esta idea de relacionar los objetos del mundo físico como si fueran recursos web también se puede encontrar en otros proyectos [101].

[10]Helwett-Packard

La plataforma aporta un servicio de descubrimiento de URLs dada una localización. Cada lugar es un directorio donde se guardan la descripción y URL de los recursos disponibles dentro de él. Un usuario al acceder a un nuevo lugar obtiene la URL asociada a ese entorno. Esta URL facilita acceso a un portal que organiza los servicios y recursos del entorno como páginas web enlazadas de forma que el usuario puede navegar a través de ellas.

Por otro lado, se incluyen dos mecanismos para intercambiar información entre dispositivos. En el primero, el dispositivo emisor sube directamente la información al receptor. En el segundo es un mecanismo indirecto que se lleva a cabo en dos pasos. El emisor envía al receptor una URL que apunta a la ubicación donde se encuentra el contenido de la información. A continuación, el receptor se encarga de descargar la información a través de esta URL.

CORBA

CORBA [123] es una capa intermedia genérica y estándar basada en objetos distribuidos. *CORBA* permite interconectar objetos de forma transparente a la tecnología con la que han sido implementados. *CORBA* ha sido empleado en diferentes proyectos relacionados con las *aplicaciones sensibles al contexto* y los *entornos inteligentes*. Así sirve para comunicar los agentes que controlan la casa inteligente MavHome [75] de la Universidad de Texas en Arlington, y como pegamento para los componentes de la capa intermedia Gaia (véase el apartado 3.3.3). También se encuentra como parte de la infraestructura del sistema de localización en interiores *Bat* [132] desarrollado por el laboratorio *Senting Computing* de la Universidad de Cambridge. Estos mismos laboratorios han liberado una implementación de un objeto mediador [11] denominado *omniORB* [54].

Hive

Hive [181] es una plataforma de agentes móviles desarrollada en Java dentro del MIT Media Lab. *Hive* se compone de tres elementos: *celdas*, *sombras* y *agentes*. Por cada procesador disponible existe una *celda*. Las celdas constituyen una infraestructura que aporta a los agentes diferentes recursos que varían de celda a celda. Las *sombras* se encargan de encapsular la interacción con los recursos locales de cada celda. Finalmente, cada agente se encuentra en una celda en cada momento, pudiendo desplazarse de una a otra.

IBM Websphere

WebSphere es el nombre que se emplea para designar a todo un conjunto de soluciones cliente-servidor que comercializa IBM. Esta capa intermedia proporciona desde soporte para aplicaciones móviles hasta interfaces vocales. Su interés radica en que es la tecnología que se emplea en el *Pervasive Computing Lab* perteneciente a la división de investigación de IBM.

[11] broker

Jini

Jini [274] es una plataforma desarrollada por Sun Microsystems, confeccionada usando Java y utilizando como protocolo de comunicación RMI. Su objetivo es convertir la red en un sistema flexible y fácil de administrar en el cual se puedan encontrar rápidamente los recursos disponibles tanto por clientes humanos como por clientes computacionales. Un entorno Jini consiste en un sistema distribuido basado en la idea de grupos federativos de usuarios y de recursos requeridos por otros usuarios. Los recursos pueden ser implementados tanto por dispositivos *hardware* como *software*.

El planteamiento de *Jini* se centra en poder desplegar y configurar una red de dispositivos desde cero de forma automática. Los mecanismos de descubrimiento y *leasing* pretenden facilitar las tareas de añadir y eliminar dispositivos de la red. Con el concepto de servicio, se provee al programador de un mecanismo genérico y bastante flexible que permite modelar la funcionalidad de un dispositivo. *Jini* no ofrece una ontología de servicios, sino que deja libertad a los desarrolladores de establecer sus propias interfaces.

Ninja

En la misma línea, el proyecto *Ninja* [116] de la Universidad de Berkeley trata de desarrollar una infraestructura *software* basada en el concepto de servicio. El objetivo se centra en proveer servicios con alta disponibilidad, tolerantes a fallos y que se puedan componer de forma escalable. *Ninja* consiste en cuatro elementos que están presentes en todas sus aplicaciones: Bases, que son un *cluster* de estaciones de trabajo que tiene instalado una plataforma *software* basada en Java; Unidades, que es como se llama a los dispositivos que emplea el usuario para conectarse a los servicios; *Proxies* activos, que son elementos que permiten adaptar de manera ad-hoc las unidades o los servicios a una aplicación en concreto; y Rutas, que se emplean para facilitar la composición de servicios. Una ruta es una abstracción que especifica cómo se conectan las unidades, los servicios y los *proxies* activos. Esta abstracción también se sugiere en la *Context Fabric* de Hong [144].

One.World

Un enfoque distinto dentro de las capas intermedias enfocadas a la *Computación Ubicua* es *One.World* [117]. Esta infraestructura de capa intermedia ha sido desarrollada en el Departamento de Ciencias de la Computación de la Universidad de Washington y se ha probado en el proyecto *Portolano* [96].

One.World propone un modelo de programación [118] que se basa en tres principios: exponer los cambios, composición dinámica y separar datos y funcionalidad. En los sistemas distribuidos tradicionalmente se persigue ocultar la distribución de los recursos. Se favorece la transparencia en el acceso a los recursos ignorando las diferencias entre los recursos locales y remotos. Pero esto obliga a que los fallos en la red o la caída de un recurso se traten como un fallo crítico. Por el contrario, *One.World* propone mostrar los cambios, especialmente los fallos del

sistema, ya que en un entorno ubicuo éstos son bastante frecuentes. Así, el programador puede decidir sus propias estrategias para manejar estos cambios. Como estas modificaciones son muy frecuentes, se propone también un mecanismo que permita expandir las aplicaciones y servicios fácilmente. Este mecanismo trabaja en tiempo de ejecución. La configuración se puede trastocar dinámicamente mientras el sistema se mantiene funcionando. Finalmente, se propone una separación explícita entre datos y funcionalidad, de manera que puedan evolucionar y gestionarse por separado. La comunicación entre los componentes de *One.World* se realiza mediante un mecanismo basado en tuplas. Éstas homogeneizan el formato de representación de la información que se intercambia. La funcionalidad se obtiene mediante un conjunto de servicios genéricos que emplean las aplicaciones para sus propios servicios. La capa intermedia provee además de facilidades para persistencia de las tuplas, reasignación de recursos y seguridad.

UPnP

UPnP[12] [67] aborda el mismo problema que *Jini*. En ambos casos se trata de automatizar la incorporación, búsqueda y eliminación de recursos de una red de área local. *Jini* se basa en la idea de un servidor de páginas amarillas, donde se indexan todos los servicios ofrecidos, pero junto a la entrada de cada servicio se guarda parte del código. Por su parte, *UPnP* puede funcionar también basado en un servidor central, aunque ofrece la posibilidad de configurar la red de modo totalmente distribuido. *UPnP* es una arquitectura pensada para conectar entidad-a-entidad cualquier tipo de dispositivos tales como PCs, electrodomésticos inteligentes, periféricos o terminales inalámbricos. A diferencia de *PnP*, las comunicaciones no están centradas sobre el PC, sino que sirve para cualquier tipo de dispositivos. Es una arquitectura distribuida y abierta que se basa en los protocolos TCP/IP, UDP, HTTP y XML.

UPnP permite la configuración de los dispositivos desde cero, sin la intervención del usuario, así como el descubrimiento de nuevos recursos. El protocolo permite que éstos se añadan dinámicamente a la red, obtengan una dirección IP y anuncien su nombre y servicios al resto. Además, un dispositivo puede abandonar la red sin dejar información espúrea. A diferencia del modelo de *Jini*, que se basa en una API propietaria, *UPnP* se construye a partir de protocolos y estructuras de datos abiertos. *UPnP* es independiente tanto del lenguaje como del sistema operativo. Otra diferencia con *Jini* es que la arquitectura no se basa en el desplazamiento o descarga de código. La idea que subyace tras *UPnP* es la de una web de servicios. Ésta se consigue mediante mecanismos sencillos de descubrimiento y navegación de servicios.

3.3.2. Capas intermedias para el contexto

Las siguientes plataformas han sido diseñadas para facilitar la implementación de *aplicaciones sensibles al contexto*. Se ha seleccionado un conjunto variado que incluya tanto plataformas orientadas a la adquisición como a la distribución. Todas

[12]Universal Plug and Play

las capas intermedias que se describen pertenecen a proyectos de investigación. Se presentan ordenadas alfabéticamente.

Active Campus

El proyecto *Active Campus* [120] pretende explotar la tecnología inalámbrica implementando un sistema que provea de servicios a la comunidad universitaria de la Universidad de California en San Diego. Este proyecto ha desarrollado una capa intermedia centrada en infraestructura [119] y orientada a servicios. Esta plataforma presta especial atención a la localización como información contextual. Otro punto fuerte de esta arquitectura es la composición de servicios para conseguir un sistema fácilmente extensible e integrable.

Context Toolkit

Una de las plataformas que primero se desarrolló y ha tenido mayor repercusión ha sido la *Context Toolkit* [86] de la Universidad de Georgia Tech. Consiste en una arquitectura de objetos distribuidos donde cada objeto representa una fuente de información contextual. La *Context Toolkit*, pretende facilitar la interacción con sensores de distinto tipo y tecnología. Para poder abstraer los detalles de bajo nivel de la capa física, la *Context Toolkit* se basa en la metáfora de *widget*, proveniente del campo de las interfaces de usuario gráficas. Un *widget* es un componente *software* que facilita la separación entre la semántica y la implementación interna del componente. Así, un botón tiene unas características y funcionalidad definidas que para un usuario son iguales en cualquier tipo de plataforma, aunque el modo de implementarlo varíe de una a otra. Los *widget* proveen de mecanismos de persistencia y registro, de forma que se pueden recuperar los cambios realizados en cada sensor. Esta arquitectura provee de dos sencillos mecanismos de agregación e interpretación para crear información contextual de mayor nivel de abstracción a partir de los sensores. La plataforma *Context Toolkit* se ha empleado con éxito en distintas aplicaciones [87, 85, 86, 189].

Context Broker Architecture

La *Context Broker Architecture* [62] plantea una arquitectura basada en un servidor especializado que permite repartir el contexto entre los dispositivos, servicios o agentes que lo precisen. *CoBrA* explora el empleo de lenguajes y protocolos propios de *Web Semántica*, como son RDF y OWL, para definir y publicar ontologías sobre el contexto en un *entorno inteligente*. *CoBrA* está implementada utilizando la plataforma de agentes JADE[13] que soporta el lenguaje de comunicación FIPA[14]. La información contextual se comparte a través de un agente intermediario denominado *ContextBroker* que mantiene el modelo del contexto del entorno. Este componente centralizado facilita la compartición tanto de las ontologías que representan el contexto, como de la información contextual en sí. El resto de los

[13] Java Agent DEvelopment Framework

[14] Foundation for Intelligent Physical Agents

componentes son agentes de la plataforma o módulos externos que se comunican con el *ContextBroker* mediante SOAP[15]. El agente *ContextBroker* está formado por cuatro módulos: (a) Una base de información contextual, la cual almacena el contexto y las ontologías que lo modelan; (b) Un motor de razonamiento, que permite inferir nueva información contextual; (c) Un modulo de adquisición de conocimiento, que facilita la comunicación de los sensores; y (d) además incluye un lenguaje basado en políticas [156] que permite a los usuarios y dispositivos definir reglas para limitar el uso y acceso a su información contextual privada.

Context Fabric

Una aproximación de capa intermedia para la gestión de contexto que propone un enfoque orientado a servicios es la *Context Fabric*. El contexto se accede mediante un conjunto de servicios disponibles al cual se conectan los clientes. Éstos localizan y acceden a los servicios a través de la red gracias a la definición de protocolos abiertos compartidos por ambos (HTTP, SOAP, SMTP...). Siguiendo esta línea Jason Hong y James Landay [144] plantean una infraestructura de servicios contextuales y proponen un lenguaje de especificación de contexto para describirlos. Dos son los servicios básicos que propone Hong. El primero consiste en la creación automática de rutas [172, 158]. Éste consiste en, dada una consulta, crear automáticamente varias rutas que conecten los sensores e intérpretes para obtener la información contextual de respuesta. Dependiendo de los sensores disponibles en cada momento se proveerá una u otra ruta. El segundo propone un servicio de descubrimiento basado en proximidad, de forma que, dada una localización, el servicio encontraría todos los sensores y dispositivos cercanos.

GlObal Smart Space

La capa intermedia *GLOSS* [81] consiste en una infraestructura que facilita servicios de localización. *GLOSS* se encarga de recopilar, distribuir, almacenar y explotar esta información contextual de localización. La arquitectura presta especial atención a la distribución de la información a nivel local y global. La comunicación local se basa en interconectar los diferentes componentes mediante buses y tuberías lógicas. Los buses permiten difundir un mensaje a todos los componentes conectados al bus. Las tuberías conectan dos componentes punto a punto. La comunicación a escala global se sustenta por un esquema *peer-to-peer* híbrido. Así , el mundo se divide en regiones de modo que a cada región le corresponde un servidor. Los servidores de este nivel se comunican mediante una red *peer-to-peer*, mientras que las regiones, a su vez, se organizan de forma jerárquica.

InCA

InCA [267], junto con la *Context Toolkit*, forma parte del proyecto *Aware Home*. *InCA* es una capa intermedia basada en infraestructura cuyo objetivo es

[15]Simple Object Access Protocol

facilitar el desarrollo de *aplicaciones sensibles al contexto*. La infraestructura consiste en modelos distribuidos que ayudan a la captura, almacenamiento, y acceso de flujos de información. Las tareas comunes de desarrollo a todas las aplicaciones son abstraídas dentro de la infraestructura. Se emplea un diseño centrado en datos que permite simplificar la forma de especificar qué variables van a ser capturadas. Las tareas de capturar, acceder y almacenar se manejan como conceptos separados, haciendo que el desarrollo de una aplicación compleja sea más sencillo.

MUSE

MUSE [60] es una arquitectura orientada al procesamiento del contexto. Ha sido desarrollada en el laboratorio de Sistemas Multimedia del Departamento de Ciencias de la Computación de UCLA[16]. La motivación de este proyecto es desarrollar una capa intermedia basada en infraestructura que permita construir aplicaciones y servicios a partir de sensores incrustados en el entorno o en los dispositivos. El procesamiento del contexto se centra en construir para cada servicio disponible una red bayesiana fundiendo la información proveniente de los sensores. La infraestructura se basa en Jini (véase el apartado 3.3.1) y provee tres componentes principales: el servidor de inferencia que calcula las distribuciones de probabilidad de cada variable contextual; un servicio de búsqueda que permite saber que sensores y servicios están disponibles, y un componente de memoria para almacenar los cambios en los sensores.

QoSDREAM

QoSDREAM [190] es una capa intermedia que pertenece al laboratorio *Senting Computing* de la Universidad de Cambridge, al igual que el mediador *omniORB* (véase el apartado 3.3.1). La novedad que aporta esta capa intermedia es soportar tráfico multimedia en *aplicaciones sensibles al contexto*. *QoSDREAM* provee de mecanismos para evitar que la calidad de servicio se degrade: las aplicaciones tienen garantizado el flujo de datos multimedia en tiempo real. La información contextual que se maneja es la localización. Se propone un modelo espacial simple que divide las entidades en localizadores y localizables. Las primeras representan objetos que pueden determinar su localización, como un GPS o una etiqueta activa. Las segundas son objetos para los que se necesita calcular su localización. Cada localizable tiene asociado un localizador. La información de localización se distribuye mediante un mecanismo de eventos. Previo a la notificación se realiza un filtrado para enviar a las aplicaciones sólamente los cambios relevantes en la localización. Las aplicaciones disponen de dos componentes centralizados: un gestor de relaciones espaciales y un gestor de base de datos. El primero recibe todos los cambios en la localización proveniente de los sensores y detecta qué cambios corresponden a una misma región. El segundo almacena información estática sobre el sistema, como por ejemplo información geográfica e información relacionada con personas o dispositivos. Finalmente, las aplicaciones se construyen siguiendo un enfoque orientado a componentes. Para ello se incluye una colección

[16]University of California, Los Angeles

de componentes estándar que soportan servicios multimedia tales como cámaras o pantallas. La implementación de una aplicación será el resultado de interconectar adecuadamente estos componentes.

TEA

Una buen ejemplo de plataforma centrada en la interpretación del contexto es la que utiliza el proyecto *TEA* [240]. Esta plataforma abstrae los valores provenientes de los sensores en dos etapas. Primeramente, se calcula un conjunto de estadísticos que resume los datos de los sensores. A continuación, se convierten estos estadísticos en información contextual simbólica mediante un conjunto de reglas. Esta misma plataforma se ha mejorado y probado en el proyecto *Smart-Its* [111] que dotaba a distintos dispositivos móviles de capacidad para percibir e interpretar el contexto.

3.3.3. Capas intermedias para *entornos inteligentes*

El último bloque que se va a presentar lo constituyen plataformas orientadas al despliegue e implementación de *entornos inteligentes*. De nuevo, las capas intermedias elegidas se ordenan alfabéticamente.

Accord

Accord es un proyecto del Instituto Sueco para las Ciencias de la Computación. En este proyecto se ha desarrollado una herramienta [12] que permite configurar los dispositivos de un entorno doméstico de forma flexible. Esta herramienta se basa en la existencia de un espacio de datos accesible por todos los dispositivos donde éstos comparten su estado en forma de tuplas. Lo novedoso del proyecto es un editor [147] que permite reconfigurar dinámicamente los dispositivos del entorno como si de piezas de un puzzle se tratara. En el mercado existen plataformas como Jini, UPnP o CoolTown cuyo objetivo es conseguir una configuración rápida y flexible de un conjunto de dispositivos. A diferencia de estos proyectos, el editor de *Accord* está orientado a que la configuración la realice el usuario final, no el desarrollador. La metáfora del puzzle permite fácilmente al usuario personalizar la configuración de su hogar, incluso crear sus propias aplicaciones.

BEACH

El proyecto *BEACH* [262] surge como continuación del proyecto *COAST* [244], el cual es una plataforma orientada a objetos desarrollada en SmallTalk que facilita la creación de aplicaciones hipermedia colaborativas. *BEACH* se centra en investigar nuevas técnicas de trabajo colaborativo empleando múltiples dispositivos heterogéneos. *BEACH* significa *Basic Environment for Active Collaboration with Hypermedia* y su objetivo es proveer de una infraestructura *software* para facilitar la colaboración síncrona entre dispositivos que operan dentro de un *entorno inteligente*. *BEACH* ofrece componentes básicos y genéricos que aportan

control de acceso, mecanismos de transacciones, replicación de objetos compartidos, y control de concurrencia. Se propone, además, una metodología para el desarrollo de aplicaciones colaborativas que separa cada aplicación en cinco modelos: interacción, entorno, interfaz de usuario, aplicación y datos. Todos estos modelos, excepto el de interacción, son implementados como parte de un espacio de objetos compartidos. De esta manera, se facilita la reutilización de los modelos en el desarrollo de una aplicación ubicua. Esta capa intermedia es el núcleo de los proyectos *i-LAND* [257] y *Roomware* [263].

Gaia

Otro ejemplo de capa intermedia enfocada a *Computación Ubicua* en entornos inteligentes es *Gaia* [227]. Ésta ha sido desarrollada en la Universidad de Illinois para el proyecto *Active Spaces* [226]. El enfoque de *Gaia* es similar al de un sistema operativo distribuido tradicional. Se compone de una colección de servicios que proveen de una interfaz de programación que permite tratar el entorno y los recursos que contiene como si fuera una entidad única y programable.

Uno de los componentes de *Gaia* es una capa intermedia orientada al contexto. Ésta se basa en objetos distribuidos que emplean componentes activos como agentes móviles [216] que se comunican empleando DAML+OIL [218]. La arquitectura parte de una serie de componentes que se pueden dividir en un conjunto de proveedores de contexto, un conjunto de consumidores y un conjunto de sintetizadores. La capa intermedia ofrece un servicio genérico de búsqueda que pone en contacto los proveedores con los consumidores. Este servicio se puede especializar dependiendo de las características del contexto, de forma que si se requiere una variable contextual más abstracta ejecuta un sintetizador antes de comunicar con el consumidor. El modelo del contexto que se utiliza está definido por una ontología descrita en DAML+OIL. Finalmente, también se ofrece un servicio que permite recuperar la historia del contexto.

ICrafter

Siguiendo con los *entornos activos* y dentro de un enfoque orientado a servicios se encuentra la capa intermedia *ICrafter* [213] empleada en el proyecto *Interactive Workspace* [155]. En el diseño de *ICrafter* se ha hecho especial hincapié en la composición de servicios de manera flexible. En concreto se busca facilitar la creación de interfaces de usuario a partir de los servicios disponibles en un momento dado, donde un servicio se puede entender como un dispositivo o como una aplicación. Esta capa intermedia se construye a partir de un componente centralizado denominado *EventHeap* [154], el cual es común a todas las aplicaciones del proyecto. Éste es un sistema de comunicación basado en eventos que sigue el modelo de pizarra expuesto por LINDA[110] y se implementa empleando los *TSpaces* de IBM [297]. Los procesos, por un lado, envían eventos al *EventHeap* cada vez que se quiere comunicar un cambio. Por otro lado, los procesos se subscriben a patrones de eventos del *EventHeap*. Cada vez que entra un nuevo evento en el *EventHeap* que case con un patrón se notifica al proceso correspondiente.

InConcert

InConcert [47] es la capa intermedia empleada en el proyecto *EasyLiving* de la división *Microsoft Research*. *InConcert* aporta un sistema asíncrono de comunicación de mensajes con direccionamiento independiente de la máquina y basado en XML. *InConcert* suple alguna de las carencias de CORBA y DCOM. Gracias a emplear un mecanismo asíncrono se evitan bloqueos innecesarios y problemas de ineficiencia. Además, una comunicación asíncrona permite a los programas encolar las operaciones de manera natural. El espacio de nombres y el servicio de búsqueda se integran dentro del mecanismo de comunicación. Cuando se inicializa un componente se le asigna un identificador, de tal manera que el componente se encarga de avisar al servicio de búsqueda periódicamente para que no caduque el identificador. Este nombre es único para cada instancia del componente, y se mantiene constante incluso si la instancia se cambia de máquina. Esto permite que los componentes migren a través de los dispositivos para ir ajustando el rendimiento del sistema. El servicio de búsqueda conoce en cada momento la localización de cada instancia del sistema.

Metaglue

Una de las primeras plataformas empleadas en *Computación Ubicua* fue *MetaGlue* [212]. *Metaglue* proporciona un mecanismo de coordinación para grandes grupos de agentes *software* en el que se incluyen facilidades para descubrir nuevos servicios y para arbitrar en la adquisición de recursos. *MetaGlue* se ha diseñado como una extensión del lenguaje de programación Java. El lenguaje *MetaGlue* surge como la aproximación basada en agentes del lenguaje *SodaBot* [71]. Junto con la arquitectura multiagente *ScatterBrain* [72], se ha utilizado como base del proyecto *Intelligent Room* [70], y actualmente forma parte de las tecnologías que sustentan la iniciativa Oxygen [202] del MIT.

OSGi

OSGi [112] es una capa intermedia cuya funcionalidad es la de servir de pasarela entre dos mundos que, hasta la penetración en los hogares de Internet, se encontraban separados. El objetivo es definir y promover un estándar abierto para permitir conectar los servicios ofrecidos en redes metropolitanas (WAN) a redes de área local (LAN) o a viviendas (HAN). La aportación de *OSGi* radica en la estandarización de las conexiones entre los equipos de fuera y dentro de la vivienda, lo cual facilita la creación de servicios tales como voz sobre IP, TV bajo demanda, control a distancia, etc. Aunque OSGi se define como una organización independiente y sin ánimo de lucro, el hecho es que las especificaciones están centradas en la tecnología Java.

El núcleo de las especificaciones consiste en una colección de APIs que definen la pasarela de servicios. Esto permitirá la conexión de la próxima generación de dispositivos inteligentes que se puedan encontrar en un hogar (oficina) con los servicios externos a la vivienda (oficina) ofrecidos a través de Internet. De esta forma, los Proveedores de Servicios de Internet (ISP), operadores de red y

fabricantes de equipos pueden ofrecer una amplia gama de servicios a los usuarios finales utilizando todos la misma pasarela. La pasarela de servicios[17] es un servidor embebido que se implanta en la propia red para conectar la red Internet externa con los clientes internos. La SG se inserta entre la red de Proveedores de Servicios y la red interna de la vivienda, de tal forma que se separa la topología interna y externa de ambas redes. Esta pasarela debe ser capaz de manejar tanto flujo de datos como multimedia.

OSGi ha sido empleado con éxito en proyectos de investigación combinado con ontologías de servicios que se emplean para formalizar y poder razonar sobre los servicios ofrecidos [124].

Semantic Space

Esta capa intermedia [275] se define como un infraestructura para *entornos inteligentes* basada en contexto. Se basa en tres aspectos claves: (a) Una representación explícita del conocimiento [276]. Para ello, emplea los lenguajes RDF y OWL, proponiendo dos niveles de representación. Un primer nivel contiene los conceptos básicos comunes a todos los entornos, en donde se distinguen tres clases relativas a objetos del mundo real, que son el usuario, la localización y la entidad computacional, y una clase referente a un objeto conceptual, que es la actividad. Un segundo nivel se compone de diversas ontologías hechas a medida para cada tipo de entorno. (b) Un motor de consultas basado en RDQL. Éste permite realizar consultas sobre la base de conocimiento que almacena la información contextual obtenida del entorno. (c) Un razonador basado en encadenamiento hacia delante que permite inferir nuevas situaciones a partir de la información almacenada en la base de conocimiento.

La infraestructura se compone de varios módulos denominados *wrappers*, un agregador, una base de conocimiento, un motor de consultas y un razonador. Los *wrappers* se encargan de obtener el contexto de los sensores y aplicaciones *software* transformándolo en la representación interna. El componente agregador se encarga de recolectar el contexto creado por los *wrappers* e introducirlo en la base de conocimiento. Tanto los *wrappers* como el agregador se implementan como servicios *UPnP*. Gracias a las características de UPnP el descubrimiento y configuración de nuevos *wrappers* se puede realizar de forma dinámica. En concreto, el agregador puede realizar un seguimiento de los *wrappers* disponibles en cada momento. Las *aplicaciones sensibles al contexto* pueden emplear el motor de consultas o el razonador para obtener el contexto, o pueden suscribirse a los cambios de éste directamente en los *wrappers*.

3.4. Conclusiones

La *Computación Ubicua* se presenta como una nueva área de investigación que nace como una rama de los sistemas distribuidos y la computación móvil incorporando requerimientos propios de la interacción hombre-máquina. La fusión

[17] Services Gateway (SG)

de estas dos áreas se resume en dos objetivos propuestos por la *Computación Ubicua*: **ubicuidad y transparencia.**

La ubicuidad lleva a que la capacidad de computación deje de ser dominio exclusivo de los ordenadores personales. Se propone desplegar multitud de dispositivos interconectados, entre los que se incluyen objetos de la vida cotidiana (véase los apartados 2.2 y 2.5.1). Este cambio también afecta al *software* dando como resultado nuevas aplicaciones (véase el apartado 2.4) que se sustentan en novedosas plataformas *software* (véase el apartado 3.3.1). Tal como se describe en el apartado 2.3 son diversas las causas que han propiciado el lanzamiento de la *Computación Ubicua*. Este avance, hasta el momento, se ha materializado en mejoras en el *hardware*, quedando más camino que recorrer en la ubicuidad del *software*.

La transparencia se aplica a la interacción entre el usuario y el sistema. El objetivo final persigue conseguir una interacción tan fluida como la comunicación entre personas. Para alcanzar este ambicioso objetivo se requiere de pequeños pasos intermedios que vayan añadiendo mayor transparencia a las interfaces actuales. Entre los múltiples pasos a realizar se señalan dos íntimamente relacionados.

Primero, crear aplicaciones que también tomen la iniciativa en el diálogo con el usuario. Mayoritariamente, el papel de los ordenadores en la interacción con el usuario es pasivo. Las interfaces de ventana cuentan con unos dispositivos de entrada/salida diseñados en ese sentido. En los sistemas ubicuos la proactividad se coloca en un primer plano. La iniciativa del dialogo se decide según la situación y las interfaces disponibles en cada momento.

Segundo, para poder alcanzar el propósito anterior se precisa de capturar la información implícita en la interacción entre el usuario y el ordenador. El empleo del contexto permite construir aplicaciones que reaccionan activamente, ya que se pueden detectar situaciones en las cuales se requiere una determinada acción pero el usuario no la expone explícitamente. El contexto es parte fundamental de la comunicación humana, y como tal tiene que ser considerado por una aplicación que busque una interacción transparente. La importancia del contexto dentro de la *Computación Ubicua* ha quedado plasmada en la práctica por las numerosas *aplicaciones sensibles al contexto* desarrolladas (véase el apartado 2.4.1).

La propuesta que se plantea en esta tesis se orienta a este segundo objetivo, aunque no se centra tanto cómo se captura la información contextual sino en cómo se representa y se distribuye a las *aplicaciones sensibles al contexto*. En este sentido, una cuestión sobre la que se entrará más a fondo en el capítulo 4 es dilucidar qué información se considera parte del contexto del usuario. A tenor del resumen realizado en el apartado 2.4.1, según la información que utilizan las *aplicaciones sensibles al contexto*, se podrían identificar inicialmente al menos cuatro variables: **identificación del usuario, localización, actividad** y el **tiempo.**

Dentro de los sistemas ubicuos proactivos, los *entornos inteligentes* son un ejemplo de *aplicación sensible al contexto* (véase el apartado 2.5). Un *entorno activo* se caracteriza por ser una infraestructura disponible para un conjunto de personas, aplicaciones y dispositivos que comparten un mismo espacio físico. La ubicuidad del sistema queda acotada por unas barreras físicas. Los objetivos propuestos se mantienen, pero se abordan con más facilidad al quedar limitados

Genéricas	CoolTown, CORBA, Hive, IBM Websphere, Jini, Ninja, One.World, UPnP
Orientadas a contexto	Active Campus, CoBrA, Context Toolkit, Context Fabric, GlOBal Smart Space, InCA, MUSE, QoS-DREAM, TEA
Orientadas a *entornos inteligentes*	Accord, BEACH, Gaia, ICrafter, InConcert, Metaglue, OSGi, Semantic Space

Tabla 3.2: Clasificación de las capas intermedias según el dominio de aplicación para que el fueron diseñadas.

espacialmente. Los *entornos inteligentes* se han desplegado en dominios variados (véase los apartados 2.5.1) en los cuales el hogar ocupa un lugar destacado.

Para que las aplicaciones dentro de un *entorno activo* puedan emplear la información contextual se requiere de tres tipos de tecnología base. Primeramente, se precisa de dispositivos capaces de detectar y capturar el contexto del entorno. Se han presentado diversas soluciones en el apartado 3.2.1. Segundo, se necesitan redes de comunicación que interconecten los sensores con las aplicaciones, y éstas entre sí. Las redes que se pueden encontrar en un *entorno activo* se han presentado en el apartado 3.2.2. En tercer lugar, se requieren dispositivos que permitan modificar el mundo físico. Estos dispositivos se denominan actuadores y se han recogido en el apartado 3.2.3.

Un problema asociado a las tecnologías anteriores surge de la **heterogeneidad** de dispositivos y protocolos de comunicación empleados. (véase el apartado 1.2). Estos componentes se encuentran **distribuidos** por el entorno, lo que supone una dificultad adicional en el modelo de programación. A esto hay que añadir que, lejos de tener una configuración estática, los componentes aparecen y desaparecen del entorno **dinámicamente**.

Otro problema importante es la **representación del contexto**. La información contextual que manejan las *aplicaciones sensibles al contexto* se compone de información capturada directamente en los sensores, y de información de mayor nivel de abstracción que se genera a partir de la anterior. Este tema se tratará más en profundidad en el capítulo 4.

Las dificultades expuestas permiten apreciar que el desarrollo de un sistema ubicuo partiendo desde cero conlleva un considerable esfuerzo. Esto ha motivado la aparición de diferentes plataformas *software* que solucionen parte de los problemas más comunes, entre las que se encuentra la propuesta de esta tesis. Estas plataformas sirven de nexo de unión entre el mundo físico y la capa de aplicación. En el apartado 3.3.1 se han descrito diversas capas intermedias relacionadas con la *Computación Ubicua*. Las soluciones que aporta cada plataforma dependen del dominio de aplicación al que estén orientadas. Tal como se muestra en la tabla 3.2 se han dividido en tres campos: genéricas, orientadas al contexto y orientadas a los *entornos inteligentes*.

Dentro de las capas intermedias genéricas, *CORBA* es una capa intermedia

con una larga trayectoria en sistemas distribuidos. En *Computación Ubicua* ha sido empleada con éxito en varios proyectos (véase el apartado 3.3.1) como soporte para construir una capa intermedia más centrada en el problema que se trataba de resolver. Dentro de estos problemas uno de los que más preocupa es la configuración dinámica de los recursos del sistema ubicuo, donde un recurso puede ser un servicio, un dispositivo, un componente software, etc. En esta línea se encuentran *Jini* (véase el apartado 3.3.1), *UPnP* (véase el apartado 3.3.1), y *CoolTown* (véase el apartado 3.3.1). Estas capas intermedias proveen de mecanismos que permiten seguir dinámicamente el rastro de los recursos disponibles, y localizarlos bajo demanda del cliente. *CoolTown* apuesta por la tecnología web que tiene como ventaja estar basada en protocolos estándar y muy difundidos. Nuestra propuesta comparte con *CoolTown* el empleo de URL para designar recursos y objetos del mundo físico (véase el capítulo 5).

Las plataformas *Ninja* y *One.World* persiguen objetivos similares a los anteriores, añadiendo una nueva característica. Ambas aportan mecanismos que permiten la composición dinámica de nuevos recursos a partir de los existentes en un momento dado. *Hive* se ha presentado como ejemplo de plataforma de agentes móviles. En este caso la reconfiguración dinámica se consigue desplazando los agentes donde están los recursos. De esta forma, las llamadas a los recursos son locales en vez de remotas.

Las plataformas anteriores tienen gran utilidad ya que solucionan problemas comunes que aparecen dentro de cualquier sistema distribuido. Tal como se ha comprobado, éstas forman parte de diversos proyectos de investigación. Aun así, las *aplicaciones sensibles al contexto* requieren de soluciones específicas que pongan en un primer plano la gestión de la información contextual.

De las capas intermedias orientadas a contexto que se muestran en la tabla 3.2, se extrae un conjunto que se caracteriza por centrarse en la representación y distribución del contexto. Estas plataformas son *CoBrA*, *Context Toolkit* y *Context Fabric*. *CoBrA* se explica más adelante por su especial relevancia. La plataforma Context Toolkit está considerada como una plataforma pionera dentro las capas intermedias orientadas a contexto. Introduce la necesidad de abstraer la información proveniente de los sensores para obtener información contextual más elaborada, para ello propone componentes que representen a sensores genéricos que encapsulan a los sensores, y otros componentes, denominados intérpretes, que generan información de mayor nivel de abstracción. La plataforma *Context Fabric* propone la creación de una infraestructura de servicios contextuales, y presenta un enfoque similar al que emplea Ninja en cuanto a la composición de servicios.

Las plataformas anteriores manejan un sentido amplio del contexto, en el que incluye información contextual diversa, mientras que *Active Campus* (véase el apartado 3.3.2) y *GLOSS* (véase el apartado 3.3.2) son dos buenos ejemplos que se especializan en procesar la localización espacial. Estas capas intermedias entran dentro de la categoría que se denomina computación sensible a la localización [18]. Estas plataformas aportan servicios para localizar espacialmente recursos y usuarios. Así, presentan modelos de representación espacial que sirven de referencia

[18]Location-Aware Computing

para las aplicaciones sensibles a la localización. Manteniendo una línea similar, se encuentra la capa intermedia *InConcert* (véase el apartado 3.3.3). A diferencia de las anteriores, la representación espacial en *InConcert* se ajusta para aplicaciones en entornos interiores, como los *entornos inteligentes.*

El desarrollo de aplicaciones multimedia ubicua ha suscitado un gran interés. Las interfaces audio-visuales son de especial relevancia en un *entorno inteligente*(véase el apartado 2.5.1). Dar una solución genérica para gestionar el tráfico multimedia constituye una tarea compleja, ya que el manejo de tráfico síncrono requiere de consideraciones especiales para mantener la calidad de servicio. A esto se le añade la diversidad de formatos y protocolos disponibles. De los diversos esfuerzos en este campo dentro de la *Computación Ubicua*, se han destacado dos. El primero es *BEACH*, una capa intermedia que da soporte para aplicaciones colaborativas multimedia y que ha sido empleada con éxito en diversos proyectos relacionados con *entornos inteligentes* (véase el apartado 3.3.3). El segundo es *QoSDREAM* (véase el apartado 3.3.2). Esta capa permite el desarrollo de *aplicaciones sensibles al contexto* capaces de distribuir el tráfico multimedia según la localización. La propuesta de esta tesis recoge la necesidad de gestionar de manera contextual el tráfico multimedia que se distribuye dentro del entorno. Para ello lo aborda —como se verá en el apartados 4.7.2 y 7.6.2— incluyendo los flujos multimedia como un componente más del entorno. La propuesta parte de la existencia de una o varias tecnologías de difusión de tráfico multimedia que permiten controlar la distribución de los flujos multimedia, de modo que este control sea independiente de la tecnología empleada.

Por último, se han presentado *TEA* y *MUSE* (véase el apartado 3.3.2) como dos ejemplos interesantes de capas intermedias orientadas a la adquisición y procesamiento del contexto. La primera se basa en redes neuronales que se encargan de extraer estadísticos que resuman la información sensorial. La segunda emplea la inferencia bayesiana para fusionar la información proveniente de los sensores. Estas dos capas presentan interés para el propósito de esta tesis no tanto por la finalidad que persiguen —que se encuentra fuera de los objetivos de la tesis— sino por representar dos muestras de la hetereogenidad que se puede encontrar en las técnicas empleadas en el procesamiento de información proveniente de los sensores.

Dentro de los *entornos inteligentes*, las iniciativas que se aproximan mejor al tratamiento de información contextual son CoBrA (véase el apartado 3.3.2), Gaia (véase el apartado 3.3.3) y Semantic Space (véase el apartado 3.3.3). Éstas interesan especialmente en el ámbito de esta tesis porque se hacen eco de los problemas que aparecen tanto en la gestión del contexto como en la gestión de un *entorno inteligente.*

CoBrA aparece posteriormente a los desarrollos de esta tesis. Su arquitectura es similar a la que se propone en este documento. Se implementa empleando una plataforma de agentes en la que se proporciona un agente central denominado *Context Broker*, el cual recopila y almacena el contexto. *CoBrA* pone de manifiesto la necesidad de una representación explícita del contexto. Para ello emplea lenguajes de representación de ontologías propios de la *Web Semántica.* Además incluye un motor de razonamiento integrado con la base de conocimiento. Desde

nuestro punto de vista, esta decisión rompe con la separación entre funcionalidad y datos. Este motor facilita la inferencia de nueva información contextual, pero limita la expansión del sistema en cuanto a la capacidad de razonamiento mediante componentes desarrollados por terceras partes. Finalmente, una característica importante de *CoBrA* es la inclusión de un lenguaje basado en reglas que permite definir la privacidad de la información contextual. La privacidad es una cuestión que ha estado expuesta desde los comienzos de la *Computación Ubicua*, y en la cual todavía quedan muchos puntos por resolver.

Gaia es un sistema operativo diseñado para *entorno activo*. Incluye distintos servicios asociados a la gestión de los componentes del entorno, entre los cuales se incluyen un gestor contextual de archivos y un servicio de localización de personas y dispositivos. *Gaia* apuesta por una solución basada en objetos distribuidos similar a la *Context Toolkit*, añadiendo un servicio genérico de búsqueda de proveedores contextuales. Este enfoque —como se argumentará más adelante— diverge del que se expone en este documento.

Finalmente, la plataforma *Semantic Space* —al igual que *CoBrA*— expone un enfoque conceptualmente similar a la propuesta de esta tesis. *Semantic Space* dispone como conceptos básicos de primer nivel, el mundo real, el usuario, la localización, la entidad computacional y la actividad. El modelo del mundo que se propone en el capítulo 4 comparte un conjunto de entidades similares. La principal diferencia entre *CoBrA* y *Semantic Space* es la forma en cómo gestionan la localización de los recursos del entorno. Mientras que *CoBrA* descansa en la plataforma JADE, *Semantic Space* se beneficia de las ventajas de UPnP. En nuestra propuesta esta parte queda dentro del modelo de datos. Cada recurso, ya sea *software* o *hardware*, se describe usando un lenguaje común y forma parte del modelo que conforma el contexto del entorno. La inclusión de un nuevo recurso se corresponde con la aparición de un cambio en el contexto. Ambos se difunden al resto de los componentes empleando los mismos mecanismos de comunicación.

De las capas intermedias empleadas en *entornos inteligentes*, *OSGi* es la que se encuentra más enfocada al hogar. *OSGi* es una solución en la línea de *Jini* y *UPnP*, que permite gestionar servicios en red basados en componentes. A diferencia de las anteriores nació con el propósito de encargase de conectar la red interna del hogar con las redes metropolitanas externas. Para ello define un conjunto de servicios estándar accesibles a través de una pasarela que conecta los dos mundos. Esta iniciativa se encuentra avalada por un consorcio de empresas multinacionales. Aunque *OSGi* no se ha posicionado como la solución definitiva para pasarelas residenciales, su seguimiento es interesante ya que es un claro candidato para definir los servicios estándar que tendrá el futuro hogar digital.

Más allá del dominio donde se aplican, las capas intermedias estudiadas se pueden dividir de acuerdo al modelo de programación que imponen al desarrollador. En el apartado 3.3.1 se diferencian dos grandes bloques: orientadas a servicio y orientadas a datos. En la tabla 3.3 se puede observar cómo se distribuyen las diferentes plataformas en cada uno de los bloques.

En el primer modelo la unidad básica la forma el servicio. Los componentes del sistema se representan según la funcionalidad que ofrecen. Cada recurso se modela como un conjunto de servicios, que están disponibles cuando el recurso

Orientadas a servicios	Active Campus, BEACH, Context Fabric, Gaia, Hive, Ninja, OsGi
Orientadas a datos	Accord, CoBrA, Context Toolkit, ICrafter, InCA, One.World, Semantic Space

Tabla 3.3: Clasificación de las capas intermedias según el modelo de programación.

lo esté. La misión de la capa intermedia es doble. Por un lado, provee de un conjunto de servicios genéricos que se han comprobado útiles para la mayor parte de las aplicaciones. Por otro, se encarga de registrar los servicios ofrecidos por los recursos del sistema, y facilitar la búsqueda a las aplicaciones que lo soliciten. Un requisito que está tomando relevancia en las plataformas orientadas a servicio es la composición dinámica de servicios. Éste consiste en la creación de nuevos servicios a partir de los ya existentes respondiendo a las necesidades del sistema en un momento dado. Ya se han nombrado *Ninja* y *One.World*, a las que hay que añadir las capas intermedias *Active Campus* (véase el apartado 3.3.2), *Context Fabric* (véase el apartado 3.3.2), *ICrafter* (véase el apartado 3.3.3) y *Metaglue* (véase el apartado 3.3.3).

En contraposición al modelo funcional, la programación dirigida por datos propone un paradigma declarativo. En el primero se describe qué es lo que puede hacer el sistema, mientras que en el segundo se expresa qué es lo que existe. Los recursos del sistema y la información que éstos generan se representan mediante un modelo de datos. Éste puede ser de diversa naturaleza: pares atributo-valor, documento XML, red semántica, tuplas, etc. La función de la capa intermedia consiste en facilitar el intercambio de los datos entre los componentes del sistema. Para ello se pueden emplear diversos mecanismos de comunicación: comunicación directa, sistemas de eventos, paso de mensaje, pizarra, o una mezcla de varios. Observando el conjunto de capas intermedias orientadas a datos, la mayor parte optan por una comunicación de tipo pizarra. Así, dentro de este conjunto se encuentran: *Accord*, *CoBrA*, *ICrafter*, *One.World*, y *Semantic Space*. *QoSDREAM*, aunque se basa en comunicación directa entre las aplicaciones, también incluye dos componentes basados en pizarra que recogen la información de localización.

En la propuesta de esta tesis se ha considerado preferible elegir el segundo modelo, ya que presenta una serie de características que se ajustan a la solución de las dificultades expuestas en el desarrollo de *entornos inteligentes*.

En primer lugar, este modelo hace explícita la separación entre funcionalidad y datos, lo cual permite evolucionar ambas partes individualmente. De esta forma, una capa intermedia orientada a datos puede ser la base para construir una capa intermedia orientada a servicios, con la ventaja que la misma capa de datos se puede reutilizar para distintos servicios.

En segundo lugar, los mecanismos de comunicación empleados para distribuir los datos permiten un mayor desacoplo entre los componentes de la arquitectura. En concreto, la comunicación tipo pizarra —que se explicará en más detalle en el apartado 5.2— facilita la coordinación de las aplicaciones que interaccionan con el

entorno, ya que estas no precisan estar sincronizadas ni temporal ni espacialmente. En el primer caso se permite que existan desajustes en cuanto a los tiempos de ejecución de cada aplicación, de forma que no es necesario que ambas aplicaciones se encuentren simultáneamente en ejecución para poder intercambiar información. En el segundo caso, se propicia que las dos aplicaciones se puedan coordinar sin tener que conocer una la existencia de la otra. Ambos casos de desacoplo son posibles gracias al uso de la pizarra como elemento intermedio a través del que pasan y se almacenan todos los datos. Este tipo de comunicación facilita la reconfiguración de entornos dinámicos como los ubicuos, ya que la aparición y desaparición de nuevos componentes y aplicaciones se realiza de forma transparente.

En tercer lugar, la información contextual se compone de medidas recopiladas en los sensores y de información deducida. La casuística de los procesos para obtener la información deducida a partir de la información sensorial es muy diversa. En cambio, la representación del contexto —una vez fijada— se puede reutilizar para multitud de aplicaciones. Las técnicas de extracción de características pueden mejorar obteniendo información cada vez más fiable y precisa, pero las variables contextuales que se miden seguirán siendo esencialmente las mismas. En este sentido trabaja una arquitectura basada en datos en la que los procesos pueden variar mientras que el modelo de datos se mantiene inalterable.

En cuarto lugar, las variables que aparecen más frecuentemente como parte de la información contextual (véase el apartado 2.4.1) se pueden representar de diversas formas pero, una vez decidido el modelo, el número de aplicaciones y servicios que se pueden construir sobre él es ilimitado. El esquema para representar el contexto podrá ser más o menos detallado, o variar en su extensión. Ahora bien, los conceptos que se manejan —como identidad, localización, actividad o tiempo— parecen bastante afianzados dentro de la definición de contexto. En cambio, las *aplicaciones sensibles al contexto* evolucionarán más rápidamente y en función de las necesidades de los usuarios y la tecnología disponible.

En quinto y último lugar, como ya se ha comentado, un aspecto importante a tratar en la *Computación Ubicua* son las interfaces de usuario (véase los apartados 1.2 y 2.5.1). La generación de interfaces ubicuos constituye un tema de investigación en alza. Un punto clave consiste en mantener un único modelo de la aplicación y generar la interfaz dinámicamente adaptándose al dispositivo que requiera el usuario en cada momento [205]. Las plataformas orientadas a datos facilitan está separación. Esta idea es la que utilizan dos de las capas intermedias expuestas: *Accord* e *ICrafter* (véase el apartado 3.3.3), y también se explota en la presente tesis (véase los apartados 6.3.1 y 7.7). La diferencia entre ambas radica en que mientras que la primera emplea un espacio de datos basado en tuplas, la segunda propone un módulo denominado *EventHeap*.

A pesar de lo anteriormente expuesto, un enfoque orientado a datos no excluye que la capa intermedia provea de servicios básicos. Por ejemplo, almacenar un histórico de la información contextual es un servicio interesante a tenor del número de aplicaciones que lo demandan (véase el apartado 2.4.1).

PARTE II

Desarrollo

Capítulo 4

Información Contextual

4.1. Introducción

Tal como se explicó en el capítulo 1, el contexto es una parte fundamental de la comunicación humana. Un diseño centrado en el usuario debería incluir el contexto en su lista de prioridades. El empleo de la información contextual es esencial para la consecución de una interacción persona-ordenador más cercana a la comunicación humana que a la comunicación entre ordenadores. En esta tesis se propone una aproximación para diseñar aplicaciones centradas en el contexto y que trabajan en el marco de un *entorno inteligente*. En concreto, este capítulo se centra en identificar cuál es la información contextual relevante que tales aplicaciones requieren y cómo se puede representar esta información. En los siguientes capítulos se estudiará cómo obtenerla, almacenarla y distribuirla entre los diferentes componentes del entorno.

Una conclusión que se extraerá de este capítulo es que no existen características intrínsecas que puedan definir una información como contexto. Por el contrario, ésta adquiere dicha categoría dependiendo de cómo se interprete. En otras palabras, la información que obtenemos del entorno se convierte en contexto cuando es usada. Por tanto, cualquier información puede llegar a ser utilizada como contexto, pero sin embargo es imposible que un modelo pueda abarcar toda la información de un sistema. Un buen modelo del contexto, entonces, tendrá que resaltar la información que tiene más probabilidades de ser requerida por las aplicaciones sensibles al contexto.

Durante el congreso *Computer Human Interaction (CHI2000)* de la ACM [1] tuvieron lugar dos reuniones de trabajo relacionadas con la *Consciencia del Contexto*. La primera de ellas se tituló *Situated Interaction in Ubiquitous Computing* y la segunda *The What, Who, Where, When, and How of Context-Awareness* [242]. En este última se resaltaba la falta de entendimiento del concepto de *Consciencia del Contexto* y la inmadurez de la tecnología y aplicaciones dentro de este campo. Se establecían una serie de objetivos que intentaban paliar estas carencias. Estos eran: (a) la definición de una taxonomía y un modelo de representación uniforme del contexto, (b) el desarrollo de infraestructuras sobre las cuales asentar futuras

[1] Association for Computing Machinery

aplicaciones, y (c) la búsqueda y definición de estas aplicaciones. Las aplicaciones y la infraestructuras ya han sido tratadas en anteriores capítulos (véase los capítulos 2 y 3). El estudio y elaboración de un modelo del contexto va constituir la parte central de este capítulo. En estos seminarios se plantearon las siguientes preguntas: ¿qué es el contexto? ¿quién se beneficia de una consciencia de su contexto? ¿dónde se puede explotar la consciencia del contexto? ¿cuándo es útil la consciencia del contexto? ¿por qué son útiles las aplicaciones sensibles al contexto? y ¿cómo se puede implementar la consciencia del contexto? A lo largo de este capítulo se responderá explícita o implícitamente a estas preguntas para obtener una mejor comprensión del significado de lo que es información contextual y de aquellas aplicaciones que la utilizan en mayor o menor medida.

Este capítulo se estructura como sigue: primeramente se va a reflexionar sobre el término contexto. Para ello se va a partir de la noción de contexto que se tiene en la vida cotidiana. A continuación, se va introducir cómo se entiende el contexto en las ciencias de la computación, para seguidamente estudiar el papel del contexto dentro de la *Computación Ubicua*, y su relación con los anteriores. Se hará un inciso para clarificar las diferencias y similitudes entre los conceptos de entorno, contexto y localización, que muchas veces son confundidos. Finalmente, los dos últimos apartados del capítulo se dedicarán a plantear un modelo de información contextual para *entornos inteligentes* que se empleará en los siguientes capítulos.

4.2. ¿Qué entendemos por contexto?

La manera más sencilla de responder a esta preguntar es consultar la definición que se encuentra en el diccionario de la Real Academia de la Lengua Española [215]. Éste, en su vigésimo segunda edición de 2001, dice que la palabra contexto proviene del latín y aporta cuatro significados:

1. m. Entorno lingüístico del cual depende el sentido y el valor de una palabra, frase o fragmento considerados.

2. m. Entorno físico o de situación, ya sea político, histórico, cultural o de cualquier otra índole, en el cual se considera un hecho.

3. m. p. us. Orden de composición o tejido de un discurso, de una narración, etc.

4. m. desus. Enredo, maraña o unión de cosas que se enlazan y entretejen.

La primera definición expone que el contexto es un entorno lingüístico que modifica el significado de una palabra o frase. Para conocer este entorno lingüístico hace falta conocer el marco establecido por la teoría lingüística de la comunicación. Según esta teoría, los elementos que forman parte de la comunicación entre dos personas son: emisor, receptor, canal, mensaje y contexto. El contexto se refiere a los elementos compartidos por el emisor y el receptor, externos al mensaje, y que modifican el significado de éste. De lo anterior se desprende la importancia que tiene la información contextual en la comunicación humana. El contexto actúa como

un árbitro que establece el significado de la palabra, simplifica la comunicación y permite desambiguar el mensaje. El contexto enriquece la comunicación humana haciendo que sea un acto dinámico y creativo, muy alejado del automatismo de las computadoras.

La segunda acepción considera el contexto como el entorno en el cual se considera un hecho. Esta definición resalta la relación entre el contexto y el tiempo. Cada hecho se encuadra en un contexto determinado que varía con el transcurso del tiempo.

De estas dos definiciones se puede extraer que el contexto es un concepto relativo a algo, en la primera a una palabra, y en la segunda a un hecho, y que por lo tanto tiene que ser considerado desde el punto de vista de ese algo. Se podría pensar en que cada persona tiene su propio contexto, de tal forma que el lenguaje que emplea es modificado por su contexto. Así, cuando dos personas comparten el mismo contexto logran entenderse.

Las dos últimas acepciones de la definición se refieren a significados en desuso de contexto, en especial la última que proviene directamente de la etimología de la palabra. Tal como se ha señalado, la palabra contexto deriva del latín, en concreto de la palabra latina *contextus*, la cual a su vez proviene del término *contextere*, cuyo significado es tejer o entrelazar.

Buscando la definición de contexto en otros diccionarios de lengua inglesa se observa que las dos acepciones principales coinciden. Así el diccionario Merriam-Webster [170] en su edición de 2003 dice que contexto es:

1. The parts of a discourse that surround a word or passage and can throw light on its meaning.

2. The interrelated conditions in which something exists or occurs.

El *The American Heritage Dictionary of the English Language* [30] en su cuarta edición de 2000 define como contexto:

1. The part of a text or statement that surrounds a particular word or passage and determines its meaning.

2. The circumstances in which an event occurs; a setting.

El *The Free On-line Dictionary of Computing 1993-2003* [102] expone que contexto es aquello que rodea y da significado a algo [2].

Por último, la red semántica Wordnet[295] desarrollada en la Universidad de Princeton en su versión 1.6 de 1997 establece que contexto es:

1. Discourse that surrounds a language unit and helps to determine its interpretation (syn: linguistic context, context of use)

2. The set of facts or circumstances that surround a situation or event; "the historical context" (syn: circumstance)

[2]That which surrounds, and gives meaning to, something else

Esta sucesión de definiciones perfila el significado de la palabra contexto. Más allá de su uso dentro de un entorno lingüístico, la acepción común de contexto apunta al conjunto de circunstancias que rodean a algo. Esta definición por sí sola es demasiado vaga para tomarla prestada para un modelo computacional de información contextual. En cualquier caso, sí que puede servir como punto de partida sobre el cual confeccionar el modelo. Si se pretende que las aplicaciones se acerquen a los humanos, la definición de contexto no puede estar muy alejada de lo que se entiende comúnmente por contexto.

4.2.1. El contexto en las ciencias de la computación

Cuando buscamos una definición de contexto dentro de las ciencias de la computación, el resultado, tal como apunta Patrick Brézillon [49], es una falta total de consenso. El mismo autor señala más de diez dominios distintos dentro de las ciencias de la computación en los que el término contexto adquiere diferentes significados:

- Un conjunto de preferencias y/o creencias [52].
- Una ventana [7].
- Una lista de atributos [225].
- Un conjunto de rutas en recuperación de información [39].
- Botones personalizables y accesibles [177].
- Posibles mundos [186].
- Una estructura de datos especial que controla la actividad del sistema [29].
- Entidades, cosas o eventos relacionados de alguna manera [197].
- La posibilidad de permitir escuchar lo que se dice y lo que no se dice [293].

Una aproximación formal a la definición de contexto se encuentra dentro del campo de la inteligencia artificial en los trabajos de John MacCarthy [178] y Ramanathan V. Guha [125]. Estos autores han dado una definición aplicable a la lógica de predicados. El contexto es considerado como un objeto de primera clase dónde la relaciones básicas son del tipo:

$$ist(c, p) \tag{4.1}$$

que significa que la proposición p es verdadera en el contexto c.

Esta variedad de *contextos* en los que se aplica la definición de contexto hace necesario realizar un estudio preliminar del significado que se le ha dado dentro de la *Computación Ubicua*.

4.3. La naturaleza del contexto en la *Computación Ubicua*

Las distintas acepciones que se han presentado del uso del término contexto dentro de las ciencias de la computación dan una idea de la dificultad de encontrar una definición que satisfaga a toda la comunidad. Si se restringe el dominio aplicación a la *Computación Ubicua* tampoco se dispone de una definición consensuada. Dentro de esta área, una de las definiciones que ha tenido más repercusión es la que propone Anind Dey [83]:

> Contexto es cualquier información que pueda ser usada para caracterizar la situación de una entidad, (donde una entidad es una persona, un lugar, o un objeto), y que se considere relevante a la interacción entre el usuario y la aplicación, incluidos estos últimos.

Esta definición aporta gran parte de las claves para entender el significado del contexto aplicado a la *Computación Ubicua*, pero no incluye algunos aspectos importantes. Así, es necesario atender a otras aportaciones que se encuentran en la literatura para visualizar en toda su amplitud el significado del término contexto. A partir de la definición Dey y del resto de las aportaciones que se detallan en los siguientes párrafos, se han extraído un conjunto de características que ayudan a comprender la naturaleza del contexto. Éstas son:

El contexto es información relevante a la interacción persona-ordenador.

Una de las características más importantes que la definición de Dey pone de manifiesto es el contexto como elemento enriquecedor de la interacción persona-ordenador. Esta misma idea ya se expuso en los apartados 1.2, 2.2, y 2.5.1. La propuesta de la *Computación Ubicua* se apoya en la información contextual para conseguir una interacción transparente persona-ordenador. Tanto el *software* como el *hardware* tienden a desaparecer de la atención del usuario, por lo que cobra especial relevancia el conocimiento de la situación en la que se encuentra éste. La caracterización de este conocimiento es lo que se denomina información contextual. Este punto constituye una de las diferencias de uso del término contexto en la *Computación Ubicua* con respecto a otras áreas.

El contexto es información implícita.

Por información implícita se entiende aquella que no ha sido introducida directamente por el usuario (véase el apartado 1.2). Aunque esta característica no queda reflejada en la definición de Dey, otros autores [239] si lo han puesto de manifiesto. Así lo expresan Jöelle Coutaz y Gäetan Rey [77] cuando caracterizan el contexto mediante lo que ellos denominan *variables de estado periféricas*. Éstas son variables que no son centrales en el diseño del sistema pero tienen un impacto relevante. Así, el contexto es información complementaria a la entrada del usuario, y necesaria para el diseño de sistemas interactivos en la *Computación Ubicua*.

Keiichi Sato [232] remarca esta idea cuando expone que el contexto lo constituyen las relaciones entre variables colaterales al diseño, y que potencialmente afectan al comportamiento del usuario y al rendimiento del sistema.

La barrera entre las entradas explícitas del usuario y el contexto a veces es difusa [115]. Una misma información puede ser considerada entrada o contexto dependiendo de cómo haya sido capturada. Así, la acción de teclear es a la vez la entrada explícita del usuario a un formulario, y el contexto implícito que indica que el usuario está presente en una determinada localización y realizando una determinada actividad.

El contexto es relativo al observador.

Este noción de contexto la presenta Jason Pascoe [203] cuando expuso que el contexto es un concepto relativo a la entidad que lo percibe, o en sus propias palabras:

> El contexto podría ser generalmente descrito como el subconjunto de los estados físicos y conceptuales de interés de una entidad en particular.

De esta definición se puede extraer el concepto de observador, donde se pone de manifiesto que el contexto lo determina el interés de una entidad en particular. Distintos observadores no tienen por qué percibir la misma información contextual. Ahora bien, la comunicación es efectiva cuando los observadores comparten el mismo contexto [292]. Este hecho encaja con la idea que ya se indicó en la introducción sobre cómo se mejora la eficiencia en la comunicación humana cuando ambos interlocutores participan del mismo contexto (véase el apartado 4.2).

El observador es el que determina desde su punto de vista qué información se considera contexto y cuál no. Por este motivo, no existen unas características intrínsecas que permitan clasificar una información como contextual: la información se transforma en contexto cuando se usa. En palabras de Terry Winograd [292]:

> Algo se puede tratar como contexto por el modo en que se interpreta, y no en función a sus propiedades inherentes.

Para un observador el contexto lo conforma la información implícita relevante para la interacción persona-ordenador. Por ejemplo, ¿la tarea que está desarrollando el usuario se puede considerar información contextual? La respuesta es afirmativa siempre y cuando la tarea sea una información relevante para la interacción entre el usuario y la aplicación en ese instante. Dependiendo del tipo de interacción será o no contexto. La labor del diseñador de aplicaciones contextuales es caracterizar y codificar esta información implícita, de manera que la aplicación mantenga un modelo del contexto similar al que maneja el usuario. La construcción de este modelo se complica teniendo en cuenta que potencialmente cualquier información puede formar parte del contexto del observador. Desde de un punto de vista de implementación, es necesario acotar el contexto a un conjunto de

propiedades y entidades. Según la funcionalidad de la aplicación este conjunto variará, ya que distintas aplicaciones tienen distintas maneras de interaccionar con el usuario. El diseñador incluirá en el modelo aquellas propiedades y entidades que sean relevantes para la interacción de su aplicación con sus potenciales usuarios. Un repaso a las *aplicaciones sensibles al contexto* (véase el apartado 2.4.1) permite identificar una serie de propiedades que habitualmente han formado parte del contexto.

El contexto se puede caracterizar por un conjunto de propiedades y entidades.

En la literatura se han señalado una amplia variedad de propiedades y entidades que se han empleado para definir el contexto. Así, en una de las primeras recopilaciones sobre los diferentes usos del contexto en la *Computación Ubicua* que realizó Mari Korkea-aho [162], se proponen como propiedades del contexto: (a) propiedades físicas (ej. dimensiones y localización en la habitación), (b) propiedades informativas (ej. datos y tareas relevantes a la habitación), (c) propiedades sociales (ej. personas presentes en la habitación, actividad del grupo y relaciones sociales), (d) propiedades del entorno (ej. temperatura, luminosidad, niveles de ruido), (e) propiedades de las personas (salud, actividad, actitud) y (f) propiedades del sistema (ej. tráfico de red, estado de los ordenadores).

En general, los primeros proyectos de *Computación Ubicua* coincidían en la lista de propiedades que definían el contexto. Así, Bill Schilit *et al.* [234] subrayan que los tres aspectos más importantes de la información contextual son: dónde se encuentra el usuario, quién está con el usuario, y qué recursos hay cerca del usuario. En la misma línea, Peter J. Brown [45] sostiene que el contexto es una combinación de elementos que las aplicaciones deben de conocer sobre el entorno, siendo estos elementos: la localización, la adyacencia de otros objetos, los estados críticos del sistema, el estado de los ordenadores y el tiempo. En general, la lista de propiedades no se solía definir de forma completa, sino más bien se enumeraba una serie de ejemplos de los que se podía extraer la noción de contexto. Así, el propio Brown *et al.* [46] habla, por ejemplo, de localización, la hora del día, la estación del año y la temperatura, mientras que Ryan *et al.* [230] destaca la localización, el tiempo, la temperatura y la identidad del usuario.

De las propiedades descritas, la localización del usuario es una de las más recurrentes [278, 6] y, también, a la que se han dedicado más esfuerzos de medición y caracterización [281, 132]. Esta misma conclusión también se obtuvo en el apartado 3.4, tras analizar las distintas aplicaciones y capas intermedias dedicadas a esta propiedad. La importancia de la localización como parte del contexto del usuario ha dado lugar a toda una rama dentro de las *aplicaciones sensibles al contexto* que se denomina *computación basada en la localización*, que trata de resolver el problema de determinar la localización del usuario y personalizar las aplicaciones en función de ésta. La tentación de equiparar ambos conceptos es tan fuerte que se ha tenido que poner expresamente de manifiesto que el contexto es algo más que la localización [241].

El contexto también ha sido caracterizado en ocasiones por las entidades que lo

definen. De un examen de las diferentes definiciones expuestas de contexto (véase el apartado 4.3) y de las distintas aplicaciones sensibles al contexto (véase el apartado 2.4.1) se obtienen diversos candidatos. Según Anind Dey [83] existen tres tipos de entidades: personas, lugares y cosas. Esta misma clasificación se repite en el proyecto *Cooltown* [161]. Terry Winograd [292] habla de aplicaciones, dispositivos y usuarios. En el proyecto GLOSS se mencionan personas, artefactos y lugares [81], mientras que la ontología sobre el contexto CONON [276] propone como conceptos básicos a las personas, la actividad, las entidades computacionales y la localización. CONON diferencia dos tipos de localizaciones: interiores y exteriores. Finalmente, Harry Chen propone [131] Lugar, Agente, (donde un agente puede ser una Persona o un AgenteSoftware), y Evento. Un lugar puede estar compuesto de varios lugares, así se distingue entre *LugarAtómico* y *LugarCompuesto*. En una versión posterior [63] se añade el concepto *Evento* como algo que presenta unas coordenadas espacio-temporales.

El contexto cambia con el tiempo.

El contexto surge a partir de la percepción de un observador. Esta percepción varía con el tiempo a medida que la situación en la que se encuentra cambia. Distintas situaciones se caracterizan por una información contextual distinta. La información se transforma dinámicamente en contexto a medida que pasa el tiempo. Jöelle Coutaz y Gäetan Rey [77] recogen esta idea cuando proponen el contexto como una composición de situaciones a lo largo del tiempo, donde una situación es un vector de estado que contiene un conjunto de variables observables en un instante t.

4.4. Entorno, contexto y localización

En el apartado 4.2 se introdujo la relación entre contexto y entorno. A efectos de esta disertación se va a establecer una distinción entre lo que se entiende por entorno y por contexto. Ambos conceptos hacen relación a aquello que rodea o circunscribe a una entidad, pero se imponen dos diferencias fundamentales. La primera define el entorno como la parte estática mientras el contexto como la parte dinámica. Así, el entorno lo constituyen aquellos elementos que no cambian para el usuario o que varían lentamente. Sin embargo, el contexto se compone de las circunstancias concretas en las que se encuentra el usuario en un momento dado, las cuales cambian sensiblemente con el paso del tiempo. Estas circunstancias concretas es lo que se denomina información contextual. La segunda diferencia establece que el entorno es común a todas la entidades, es decir, se percibe de la misma manera por todas, mientras que el contexto es particular para cada entidad. Cada entidad recibe una información contextual distinta. Por ejemplo, el conjunto de personas que acompaña a un usuario es distinto para cada usuario.

En este sentido se puede diferenciar entre entorno físico y localización. El entorno físico [3] es un espacio físico (como una habitación o un edificio), junto con

[3]También se puede denominar ambiente

los elementos que lo integran (ventanas, puertas, iluminación, ordenadores...), mientras que la localización es una parte de la información que conforma el contexto de la entidad. Esta última depende de cada individuo en particular y varía a medida que el usuario se mueve entre los distintos entornos.

Aunque se ha puesto de manifiesto que el contexto va más allá de la localización [241], la gran mayoría de los proyectos relacionados con computación sensible al contexto incluyen la localización del usuario como parte del contexto. Se puede afirmar que la localización de una entidad constituye una parte básica de la información contextual. Así lo expone Alan Dix [88]:

> La localización se convierte en un dispositivo de indexación útil, a partir del cual poder inferir más contexto [4]

A la hora de representar la localización del usuario se pueden establecer básicamente dos modelos: modelos geométricos, donde la localización se representa como un conjunto de coordenadas, como es el caso del proyecto *Easy Living* [47] de *Microsoft Research*, y modelos simbólicos, en los cuales la localización se caracteriza mediante conceptos tales como una habitación, una planta o un edificio. Este modelo se utiliza, por ejemplo, en el sistema *Active Badge* [235]. En ambos modelos la información se suele organizar de forma jerárquica para solucionar problemas de escalabilidad y abstracción. Así lo plantea Bill Schilit [238], que propone una jerarquía simbólica de contenedores. En los modelos geométricos las soluciones pasan por estructuras que permitan indexar la información geométrica tales como los árboles-R [193] o los árboles-Quad [282]. También se encuentran proyectos en los que se combinan ambos modelos, como el que propone Ulf Leonhardt [165].

Más allá del modelo elegido, resulta importante que éste sea capaz de poder resolver fácilmente los dos tipos de consultas básicas que se le hacen a un sistema de este tipo: dado una entidad obtener su localización, y dada una localización determinar la lista de entidades que se encuentra en ella. Además, el modelo debe soportar la actualización dinámica de la información en tiempo real.

El modelo que se plantea en esta tesis se encuadra dentro los modelos simbólicos. Tal como se detallará en los siguientes apartados, la localización se representa como una relación bidireccional entre una entidad entorno y una entidad persona. Cada entorno corresponde a un espacio físico y puede subdividirse en varios entornos de forma jerárquica, que se organiza mediante un árbol. Se impone la restricción de que los entornos que se encuentren en el mismo nivel de la jerarquía no pueden solaparse.

La elección de esta representación responde a la necesidad de obtener un modelo simple, ya que se persigue que sea la información mínima de la cual se debe disponer. Un modelo geométrico sería excesivo para la mayor parte de las aplicaciones y complicaría innecesariamente el desarrollo. Además, es más sencillo construir la representación simbólica a partir de los datos aportados por los sensores, dado que no siempre se dispone de dispositivos con la suficiente resolución. Mientras que con un simple sensor de movimiento se puede llegar a informar de que alguien todavía permanece en el entorno, es mucho más difícil de determinar

[4] Location becomes a useful indexing device from which to infer further context

su localización exacta. En caso de que se dispusiera de la tecnología adecuada, y de que fuera necesario, la representación simbólica se puede complementar con las coordenadas geométricas. Por otro lado, la sencillez del modelo permite que las actualizaciones sean rápidas.

4.5. Representación del contexto

La idea que se ha dado sobre el contexto en los apartados previos sirve como sustento para empezar a construir un modelo de información contextual enfocado a *entornos inteligentes*. El primer paso consiste en decidir el mecanismo de representación que permita modelizar tanto el estado de una entidad individual como el de todo el entorno. Guanling Chen y David Kotz [61], por un lado, y Thomas Strang y Claudia LinnhoffPopien [256], por otro, recopilan un buen resumen de las distintos mecanismos que se han empleado para representar el contexto. Estos se podrían condensar en:

- **Pares clave-valor.** La información contextual se representa como un conjunto de variables definidas por pares con el nombre de la variable (o clave) y su valor. Este tipo de representación se empleó en los primeros proyectos de aplicaciones sensibles al contexto, como en el entorno de computación móvil PARC [237] y en el proyecto Mobisaic [271]. Aunque simple, permite realizar búsquedas basadas en expresiones regulares y se pueden implementar estructuras complejas si se permite recursión en la definición de los atributos, de forma que el valor de un atributo sea la clave de otro. Siguiendo un enfoque similar, el servidor de *Localización de Información* de los laboratorios *Philips* gestiona la información mediante un servicio de directorio X.500, que permite pares atributo-valor almacenados en forma de árbol.

- **Lenguajes de marcado.** Se emplea un modelo de datos semi-estructurado definidos por algún lenguaje de tipo XML. Este modelo se utilizó por primera vez para representar las *Stick-e* [203]. Estas *pegatinas* son una especie de *Post-it* electrónico que se añade a un objeto para definir su contexto. Un lenguaje XML para definir contexto es *ConteXtML* [229], aunque no ha tenido gran repercusión. Dentro de esta categoría entraría también el lenguaje desarrollado por el *W3C* denominado *Composite Capabilites/Preferences Profile (CC/PP)* [273] o el lenguaje *User Agent Profile (UAProf)* [17] empleado por el consorcio *Open Mobile Alliance (OMA)*. Ambos se centran en el contexto del dispositivo y se emplean esencialmente para especificar de forma abstracta y estándar las capacidades del dispositivo y las preferencias de usuario.

- **Tuplas.** Una tupla consiste en una agrupación de valores de distinta naturaleza. Por ejemplo, las tuplas se emplean como la unidad básica de intercambio en el mecanismo de comunicación generativa propuesto por Gelernter [110]. Todas las tuplas se almacenan en una memoria compartida que puede ser accedida empleando un conjunto pequeño de operaciones. Este modelo

ha sido empleado en proyectos como *One.World* y *Gaia* estudiados anteriormente (véase el apartado 3.3.1).

- **Modelos basados en diagramas entidad-relación.** Estos modelos emplean el paradigma de entidad-relación de bases de datos, o proponen modelos fácilmente derivables a un modelo ER. Éste es el caso de Karen Hericksen el al. [140] quienes han publicado un extensión del lenguaje gráfico ORM (Object-Role Modeling) [129] para modelar información contextual. Otro lenguaje que también combina el modelo orientado a objetos con el modelo ER es OM (Object Model). Éste ha sido empleado por Belotti *et al.* [34] para modelar *aplicaciones sensibles al contexto.*

- **Modelos orientados a objetos.** Tanto ORM como OM comparten similitudes con el modelado orientado a objetos pero se orientan más al modelo ER. Propuestas que emplean únicamente el paradigma de orientación objetos son el modelo de Objetos Activos (Active Object Model), introducido en el proyecto *GUIDE* [65, 79], y la aproximación del proyecto *TEA* [243]. Los detalles del procesamiento del contexto y el acceso al mismo se encapsulan mediante el empleo de objetos.

- **Modelos lógicos.** El contexto se representa como un conjunto de hechos y reglas que se describen empleando lógica de predicados de primer orden, o algún sistema equivalente. Este modelo se ha empleado en sistemas multimedia orientados a localización [25]. Más recientemente un enfoque similar lo proponen los anteriormente mencionados Philip Gray y Daniel Salber [115] para el diseño de aplicaciones interactivas.

- **Modelos basado en ontologías.** Estos guardan una estrecha relación con los modelos lógicos, ya que se pueden llegar a establecer equivalencias entre ambos. Estos modelos se basan en una categorización jerárquica del contexto. Este paradigma se ha utilizado tanto para representar la información contextual [63, 131, 276] como en *Computación Ubicua* [218]. Estrechamente relacionados con la representación basada en una ontología se encuentran las redes semánticas. Éstas han sido empleadas con éxito en *entornos activos*, como por ejemplo, dentro del proyecto Aire [209].

Realizando un análisis de los mecanismos representación se puede establecer una distinción entre mecanismos de bajo y de alto nivel. Los primeros incluyen los pares clave-valor, las etiquetas XML y las tuplas, mientras que los segundos abarcan los modelos basados en entidad-relación, orientados a objetos, lógicos y basados en ontologías. Los mecanismos de alto nivel aportan un esquema que describe los diferentes objetos existentes y las relaciones entre ellos, mientras que los mecanismos de bajo nivel se encuentran más ligados a la parte de implementación. En la arquitectura propuesta se ha optado por una combinación de dos mecanismos, uno de cada nivel. De esta forma, se establece una clara separación entre el diseño del modelo y la implementación del mismo.

Dentro de los mecanismos de bajo nivel se ha optado por XML, ya que los lenguajes de marcado suponen una ventaja a la hora de implementar protocolos

de intercambio de información. Estos se construyen a partir de una gramática estándar, de forma que se benefician de la existencia de herramientas ampliamente difundidas que solucionan el análisis léxico y sintáctico (véase el apartado 6.1). Las claves pares-valor fueron uno de los primeros mecanismos que se empleó en las *aplicaciones sensibles al contexto*. Aportan un mecanismo simple, pero son menos potentes que los lenguajes de marcado a la hora de representar estructuras complejas. Finalmente, la representación empleando tuplas aporta cierta estructuración, así como mecanismos sencillos y potentes de consulta. Son una solución muy extendida como mecanismo de coordinación. Tienen como desventaja frente a XML que, aunque son un mecanismo tipado, es más complicado establecer un esquema donde se puedan jerarquizar conceptos.

Dentro de los mecanismos de alto nivel, tal como apuntan Strang y LinnhoffPopien [256], los modelos basados en objetos y en ontologías son los más adecuados para la representación del contexto. Ambos comparten ventajas y desventajas. Los dos se basan en una división en clases e instancias, donde las clases se organizan jerárquicamente y cada instancia pertenece a una o varias clases de la jerarquía. También ambos se pueden transformar directamente a una representación gráfica, facilitando las labores de diseño. Como diferencia, los modelos orientados objetos encapsulan la parte algorítmica dentro de los propios datos. En cambio, las ontologías se centran en la definición de los datos, pudiendo establecerse una separación más clara entre la información y los servicios que emplean esta información. Tal como se motivó en el apartado 3.4 se ha elegido un modelo de programación orientado a datos frente a uno orientado a servicios. En este sentido, los mecanismos basados en ontologías se presentan más adecuados que los basados en objetos.

Tal como se describe en los siguientes apartados y capítulos, se ha optado para el diseño del modelo de información contextual un mecanismo basado en redes semánticas, mientras que para la implementación se ha escogido un lenguaje de marcado. Ésto es así por que creemos más oportuno emplear un paradigma de alto nivel para definir el modelo del contexto, y luego convertirlo a un documento basado en XML que facilite la interoperatibilidad en la fase de implementación.

4.5.1. Modelo basado en redes semánticas

Para representar el contexto se ha elegido un mecanismo genérico y flexible basado en redes semánticas. Las redes semánticas son bastante atractivas como mecanismo de representación porque permiten visualizar y organizar fácilmente la información y tienen un modelo de ejecución muy simple [228].

El mecanismo que se plantea tiene dos componentes básicos: concepto y entidad. Cada concepto representa cada una de las clases de entidades que pueden existir. Las entidades son realizaciones de alguno de los conceptos presentes en el modelo.

Cada entidad se compone de:

- **Nombre.** Una cadena alfanumérica que es única para todo el sistema.
- **Tipo.** Cada entidad pertenece a la categoría que indica el tipo.

- **Propiedades.** Un conjunto pares nombre-valor que modela características intrínsecas de la entidad, es decir, aquellas que son universalmente aceptadas. Se exige que dentro de una entidad no puedan existir dos propiedades con el mismo nombre. El valor de las propiedades puede ser de alguno de los siguientes tipos: cadena de caracteres, enteros, reales, booleano y enumerado. Un tipo enumerado define el conjunto de símbolos que puede tomar como valor la propiedad.

- **Relaciones.** Cada relación se define mediante el nombre de la relación y el nombre de la entidad con la que se relaciona. Una misma entidad puede tener más de una relación con el mismo nombre, siempre y cuando la entidad con la que se relaciona sea distinta. No se permite que una entidad se relacione consigo misma.

Los conceptos se disponen jerárquicamente de forma que las propiedades se heredan de padres a hijos. La herencia múltiple no esta permitida, a diferencia de otros modelos de redes semánticas. En el capítulo 6 se tratará más en detalle el lenguaje de representación.

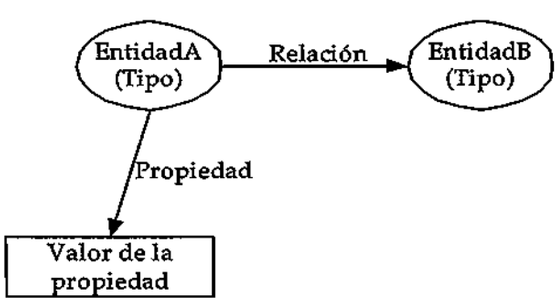


Figura 4.1: Notación gráfica que se emplea para representar las instancias del modelo

Para representar el esquema del modelo se emplea una sintaxis similar a la que emplea UML en los diagramas de clase, mientras que para visualizar gráficamente la instanciación del modelo se ha optado por un grafo dirigido (véase la figura 4.1). Un grafo es una estructura de datos que se compone de un conjunto de nodos unidos por un conjunto de arcos. Existen dos tipos de nodos: el primero de ellos define una entidad de un determinado tipo y el segundo almacena el valor de una propiedad. También existen dos tipos de arcos: aquellos que definen una relación y aquellos que definen una propiedad. Las relaciones se establecen entre dos entidades, mientras que las propiedades definen una asociación entre una entidad y valor determinado. En la figura 4.1 se muestra gráficamente la convención que se va a emplear para representar las instancias del modelo. Las entidades se representan mediante elipses que incluyen el nombre y el tipo. Los valores de las propiedades son nodos en forma de rectángulos. Tanto las propiedades como las relaciones se muestran como arcos direccionales que parten de una entidad y terminan en un valor o en otra entidad respectivamente.

4.5.2. Un modelo de información contextual

Una vez elegido el mecanismo de representación del modelo, el siguiente paso consiste en definir qué componentes van a formar parte del mismo. Para ello hay que determinar los diversos conceptos, propiedades y relaciones que caracterizan el contexto. La definición académica de contexto apunta al conjunto de circunstancias que rodean a algo (véase el apartado 4.2), donde algo puede ser una persona, una situación, un hecho, un entorno lingüístico, un entorno físico... Para que esta definición se pueda integrar dentro del modelo, primeramente hay que fijar a qué se refiere el algo, y después hay que determinar cuáles son las circunstancias que rodean a ese algo. Realizando un repaso del apartado 4.3 se puede extraer que dentro de la computación ubicua el contexto se refiere a las personas, a los lugares, y en menor medida a los objetos y dispositivos. En cambio no existe un consenso claro que permita señalar cuáles son las circunstancias a modelar. Más aun cuando se ha puesto de manifiesto que no se puede encontrar un conjunto de propiedades que permitan definir a priori estos intereses, sino que la información que percibe el observador se convierte en contexto en función de su interpretación. Siguiendo este enfoque, el modelo que se realice de la información contextual tiene que coincidir con el modelo del mundo, ya que cualquier información puede ser considerada —en algún momento dado— como contexto. Ahora bien, cualquier modelo consiste en una representación simplificada de la realidad, lo que lleva a tomar decisiones que suponen eliminar ciertos aspectos de la realidad que no se incluyen en el mismo. La decisión de qué partes se añaden y qué partes se eliminan se toma según el propósito para el cual se va a emplear el modelo.

En el caso de la *Computación Ubicua*, Anind Dey marca un punto de partida cuando expone que se considere el contexto como aquella información que se estime relevante a la interacción entre el usuario y la aplicación (véase el apartado 4.3). De nuevo, clasificar la información tomando como criterio la relevancia de la misma conlleva realizar un cierto balance, ya que no es posible establecer un criterio absoluto. En el ámbito de esta disertación, hay que tener en consideración que las *aplicaciones sensibles al contexto* actúan dentro de un *entorno activo*. La información relacionada con el mundo físico que rodea a los usuarios y a las aplicaciones tiene la mayor relevancia. Así, por ejemplo, se tiene en cuenta la relación del usuario con el espacio, qué otros usuarios se encuentran en sus proximidades, o qué dispositivos se encuentran libres.

Un segunda consideración se encuentra en el apartado 2.4.1, donde uno de los resultados que se obtuvo fue la forma en que las *aplicaciones sensibles al contexto* emplean el contexto, y en consecuencia, la forma en cómo interaccionan con el usuario. Así, Guanling Chen y David Kotz [61] exponen que el contexto se puede utilizar de forma activa o pasiva. En el primer caso las aplicaciones se adaptan automáticamente a las actualizaciones del contexto, mientras que en el segundo reaccionan a los cambios del contexto a petición del usuario. Esta diferenciación sirve como ayuda a la hora de valorar si habría que considerar relevante una determinada propiedad o relación o no. El contexto activo se asemeja a las preferencias de usuario. Cuando se detecta un cambio en el contexto, se recupera cómo el usuario prefiere que se comporte la aplicación, y se actúa en

consecuencia. En este sentido, un cambio en el contexto se tendrá en cuenta dentro del modelo si afecta a las preferencias del usuario. Por ejemplo, el usuario entra en un nuevo entorno, de modo que cambia su localización. Este cambio supone una modificación relevante en el contexto, ya que las preferencias del usuario pueden variar de entorno a entorno. El contexto pasivo se entiende como una entrada más del usuario. Cuando éste realiza un comando —independientemente de la interfaz que lo transmita— la información que implícitamente se agrega a ese comando también se considera parte del contexto. Un ejemplo en este sentido sería cómo la actividad del usuario puede modificar una orden directa. El usuario selecciona una película para visionar. Este comando produce que la película se muestre en la pantalla más cercana al usuario y configura el entorno con unas condiciones de luminosidad y ruido adecuadas para ver el vídeo.

Como tercera consideración, y relacionada con el último punto, hay que subrayar la condición del contexto como información implícita. Volviendo otra vez al apartado 4.3, se ha postulado que el contexto se caracteriza por ser información que complementa la entrada del usuario. Así, otro filtro que se puede aplicar para determinar la relevancia de cierta propiedad es preguntar si puede considerarse información implícita o no. Se ponía el ejemplo en dicho apartado de la acción de pulsar una tecla. La tecla en sí que ha sido pulsada no es relevante para el contexto —aunque sí para la entrada del usuario— pero el hecho de que un determinado teclado este siendo utilizado sí puede tomarse en consideración como parte del modelo del contexto. Este cambio en la información contextual puede indicar que un usuario se encuentra en un determinada localización y realizando una determinada actividad.

La cuarta —y última— consideración a tener en cuenta para determinar la relevancia de la información se refiere a su granularidad. Tanto la información proveniente directamente de los sensores como la información que se deduce de la anterior pueden formar parte del contexto. Existen argumentos en contra y a favor de incluir una u otra. Los sensores aportan información cruda, esto implica: (a) elevado nivel de detalle; (b) puede cambiar en el tiempo rápidamente; y (c) es ruidosa. Estas tres características exigen mayor esfuerzo por parte del desarrollador de *aplicaciones sensibles al contexto*, ya que tendrá que realizar un filtrado previo para obtener una información más fiable y más abstracta. Ésto llevaría a pensar que, dado que el procesamiento se puede reutilizar, sólo habría que incluir en modelo la información una vez interpretada. Sin embargo, no siempre se puede asegurar que la tecnología para realizar este procesamiento esté disponible. Así, por ejemplo, no siempre se va a poder obtener el número de habitantes de un entorno, ésto depende de los sensores e interpretes de que se dispongan. De esta forma, la presencia de ocupantes en una habitación queda incompleta con la variable anterior, es necesario incluir información menos abstracta como puede ser la información directa que aportan los sensores de movimiento. Otra opción podría ser añadir en el modelo todos los posibles niveles de granularidad. De esta forma, las *aplicaciones sensibles al contexto* disponen de toda la información posible sobre un mismo aspecto de la realidad. Sin embargo, este caso también presenta un inconveniente en ciertas situaciones ya que algunas veces se puede suprimir la información del sensor sin que suponga ninguna pérdida significativa. Retomando

el ejemplo del teclado, se ha comprobado que añadir la propiedad de tecla pulsada aporta información interesante para el contexto. Ahora bien, esa misma propiedad se puede caracterizar mediante la presencia, que indica que el teclado percibe algo cerca, tan cerca que lo está tocando. De esta segunda manera, se consigue abstraer la propiedad particular de tecla pulsada a una propiedad más genérica que se puede reutilizar en otros objetos. En este caso, no tiene sentido que ambas propiedades convivan, sino que la más abstracta sustituye a la concreta.

Partiendo de las consideraciones anteriores, se proponen dos pasos para construir el modelo del contexto: el primero determinar qué conceptos de la realidad se identifican con más frecuencia como información contextual. El segundo paso consistirá en identificar para cada concepto cuáles son las propiedades y relaciones qué habitualmente se le asocian. En este proceso de diseño se ha tomado en consideración la experiencia previa de representaciones del contexto que han realizado los desarrolladores de *aplicaciones sensibles al contexto*. De esta manera, se incorporan las características que aparecen de forma reiterada cuando se discutió la naturaleza de la información contextual en el apartado 4.3.

El resultado del proceso de diseño consistirá en un modelo mínimo que abarque la información más relevante para las *aplicaciones sensibles al contexto* que trabajan en *entornos activos*. Hay que tener en consideración que existen entornos particulares en los que se requiere extender el modelo para incluir información contextual propia. Así, tan importante como identificar los conceptos principales que conforman el modelo, va a ser diseñarlo de forma que se pueda ampliar fácilmente. Este problema ya ha sido abordado en la literatura empleando diversos enfoques [83, 276, 63]. Todos ellos distinguen entre dos partes: básica y extendida. Nuestra parte básica se va a denominar contexto primario, mientras que la parte extendida se corresponde con el contexto secundario.

En el siguiente apartado se va a detallar el modelo de contexto primario. Seguidamente se expondrán dos ejemplos de contexto secundario aplicados al hogar digital y a la representación de flujos continuos como información contextual.

4.6. Contexto primario

Si se restringe el estudio del contexto al marco de los *entornos inteligentes*, la pregunta que se plantea ahora es ¿cuáles son los conceptos básicos que son relevantes para la construcción de *aplicaciones sensibles al contexto* dentro de estos entornos? A tenor del apartado 4.3 se han identificado tres conceptos básicos:

- **Persona.** Las personas se relacionan con el contexto en dos sentidos. Por un lado, son observadores del contexto. Poseen un esquema mental del contexto, que les ayuda traducir el significado de las diversas interacciones en las que se ven involucrados. Por otro lado, forman parte del contexto: las personas cercanas a una persona son parte del contexto de esta última. Igualmente, las personas próximas al usuario de una aplicación, también forman parte del contexto de la aplicación.

- **Lugar.** Un lugar se define como una porción del espacio físico que comparte unas propiedades que lo identifican como una unidad. Así, dentro del concepto lugar se incluyen países, campus universitarios, edificios, habitaciones, etc. El concepto Lugar se subclasifica en **Entorno** y **Área**. La subcategoría de **Entono** constituye una parte fundamental del contexto (véase la discusión del apartado 4.4). Un *Entorno* se encuentra delimitado por unas barreras que se extienden en las tres dimensiones. El espacio queda así dividido en dos regiones: dentro y fuera del entorno. Esta separación es la causa de que el observador perciba de forma diferente las entidades según se encuentren dentro o fuera del entorno. Por ejemplo, las personas que se encuentran dentro del entorno adquieren mayor relevancia para el observador. La subcategoría de **Área** hace referencia a un lugar delimitado por unas fronteras definidas virtualmente. Por ejemplo, un salón de actos es un entorno delimitado por unas barreras físicas como las paredes y el techo, que se puede dividir en dos áreas: la zona de las butacas y el estrado.

- **Recurso.** El término recurso se emplea para designar a entidades que comparten la característica de que son usadas por otra entidad para completar una actividad. Este concepto abarca recursos que tengan presencia física o que sean virtuales. En este último caso se puede diferenciar entre recursos pasivos, por ejemplo un archivo, y recursos activos, donde se incluyen servicios, procesos, agentes, aplicaciones, etc.

En la figura 4.2 se muestra la jerarquía del contexto primario, empezando por una clase genérica denominada raíz de la cual heredan el resto de los conceptos.

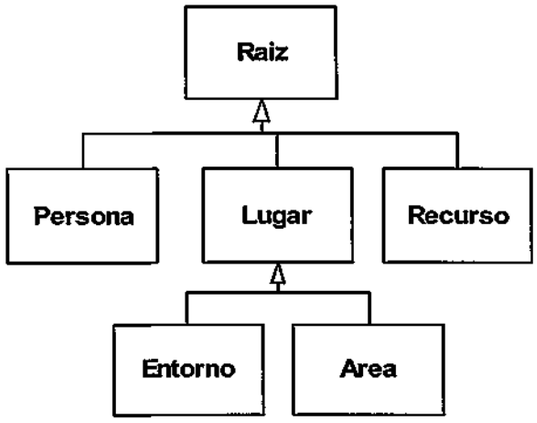


Figura 4.2: Representanción gráfica de la jerarquía de conceptos que conforman el contexto primario

Cada entidad que influye en el contexto de un *entorno inteligente* se encuentra dentro de alguna de las tres categorías definidas por los conceptos anteriores. En el estudio de la distintas *aplicaciones sensibles al contexto* se han extraído

una serie de características relacionadas con los conceptos anteriores que son las que habitualmente conforman la información contextual. Entre ellas se hallan la **identificación**, la **localización**, y la **actividad** (véase los apartados 2.4.1 y 4.3).

4.6.1. Identidad

La identificación de las personas o de los recursos que se encuentran en el entorno constituye un aspecto básico dentro de cualquier *aplicación sensible al contexto*. Para que el entorno se adapte —tanto activa como pasivamente— a cada usuario, se precisa de un mecanismo para poder distinguirlos unívocamente. En cierto modo, se puede afirmar que la identidad constituye el paradigma de información contextual. Como el contexto se ha definido relativo a un observador, es necesario conocer la identidad del mismo para poder determinar qué información es más relevante para las *aplicaciones sensibles al contexto* en cada momento. La importancia de la identidad se comprueba tanto en el campo de los *entornos activos*, como en otras áreas donde las aplicaciones se adaptan a la identidad de los usuarios, ofreciendo distintos servicios según el usuario, o el mismo servicio pero con una configuración distinta. Es más, la identidad tiene importancia incluso cuando no se conoce. Una aplicación que no pueda obtener la identidad de un usuario se comportará de manera distinta que si la pudiera determinar.

Una aspecto que hay que tener en cuenta es que dentro del contexto de un usuario —aparte de su identidad— también influye la identidad de todos los usuarios y recursos que un momento dado comparten un mismo lugar. La presencia de otras entidades modifica el comportamiento del usuario, y se espera que también modifique el comportamiento de las aplicaciones. Un interfaz vocal puede decidir enviar un mensaje por los altavoces dependiendo de la presencia o no de otros usuarios. Si además se puede establecer la identidad de estas personas, la adaptación de las *aplicaciones sensibles al contexto* podrá ser más precisa. Es un hecho que las personas se comportan de manera distinta dependiendo de la identidad de las personas que les rodean. En la vida cotidiana se puede comprobar cómo las personas cambian su conducta dependiendo de con quién se encuentren, o a quién se dirigen: no es de extrañar que las reacciones de una persona dependan de si se encuentra en presencia de su mejor amigo, o de un desconocido. En conclusión, la identificación de las personas próximas es un aspecto muy relevante del contexto del usuario.

En realidad esta relevancia de la identidad no se limita a las personas, sino que —en general— se puede considerar la identidad de cualquier recurso que forme parte del contexto. Ahora bien, la identidad en el caso de los recursos no tiene por qué ser tan precisa. Si un entorno contiene un altavoz, el conocimiento de la existencia del altavoz aporta una información contextual de por sí valiosa, más allá de que sea el sistema capaz en distinguir ese altavoz en particular de otro. En general, al usuario le interesa diferenciar entre diferentes recursos cuando quiere actuar directamente sobre ellos. Así, si se dispone de dos altavoces y se quiere apagar un altavoz y encender otro, es necesario poder discriminar entre ambos. No obstante, la identidad del altavoz hay que matizarla. Para referirse a uno de los altavoces basta con distinguirlo del otro altavoz, por ejemplo mediante su

posición relativa en el entorno. De este modo, para el usuario la identidad de los altavoces vendría dada por altavoz izquierdo y altavoz derecho, más que por asignarle una identidad propia. Además, existen otras situaciones en las que el desconocimiento de la identidad concreta de un altavoz no influye excesivamente el contexto del usuario. Por ejemplo, el hecho de que haya dos altavoces permite al sistema adaptar las preferencias del usuario respecto al nivel de ruido o al hilo musical deseado. Que se pueda determinar la identidad del altavoz no es tan determinante como el que éste influya en el nivel de ruido que prefiere el usuario.

Tal como se ha podido comprobar, la representación de la identidad incluye diversos aspectos. Para ciertas aplicaciones, el tipo de la entidad puede ser suficiente como identificación de la misma. En cambio otras precisan de una identificación más concreta. Cuando se necesita el mayor grado de resolución, se recurre al identificador que se asocia unívocamente a cada entidad mediante un nombre, tal como se describió en el apartado 4.5.1. Hay que tener en cuenta que el formato de este identificador puede variar según la implementación que se realice, lo cual implica que, en general, no tiene por qué existir una asociación explícita entre el identificador y la entidad que representa.

Dado que el identificador no tiene por qué ser legible por un humano, sería conveniente añadir propiedades que describan de manera más natural a la entidad, de forma que los usuarios puedan hacer uso de esas propiedades, ya sea para referirse, o para recuperar información de una entidad. Estas propiedades cambian según el tipo de la entidad. Así, la identidad de una persona no se caracteriza de la misma manera que la de un recurso. En el primer caso, el nombre y/o los apellidos de la persona son la mejor manera de identificarla en múltiples situaciones. En cambio, en el segundo caso, simplemente con ser capaz de distinguir entre distintos tipos de recursos es una buena aproximación. El tipo de un recurso suele ser una información determinante para identificarlo. Si requiere afinar la identidad, más que asignarle un nombre propio, puede tener más sentido emplear alguna característica intrínseca que lo diferencie. Más interesante que darle un nombre a un altavoz es caracterizarlo mediante propiedades que se empleen para distinguirlo de otros altavoces, como por ejemplo, su potencia acústica, que puede dar una estimación de cuál hay que apagar primero si se quiere reducir el ruido.

Hasta ahora se ha considerado la identidad de la persona como una información imperecedera. Normalmente el nombre y apellidos de una persona se mantienen inalterables a lo largo del tiempo. Sin embargo, en ciertas ocasiones interesa referirse a una persona en función de lo que está haciendo, más que por quién es. En este sentido, la identidad de una persona también viene definida por el rol que desempeña en un momento dado. Los roles que puede ocupar una persona no tienen una existencia independiente, sino que suelen aparecer interrelacionados como integrantes de una organización. En el caso de los *entornos inteligentes*, el propio entorno marca de forma natural los límites de una posible organización. Los diferentes roles que puede desempeñar el usuario varían dependiendo del tipo de entorno. Por ejemplo, en un aula se puede distinguir entre profesor y estudiante; en un hogar entre padres e hijos; en una reunión entre presidente, secretario y asistentes, y así sucesivamente.

Dos roles genéricos que están presentes en cualquier *aplicación sensible al con-*

texto que trabaja sobre un *entorno inteligente* son:

- **Usuario.** Así se denomina a la persona que interacciona con el entorno y cuya identidad se encuentra registrada.
- **Invitado.** Una persona actúa como invitado del entorno, bien cuando no está registrada como usuario o bien cuando, aun estando registrada, no ha sido identificada. Este último caso tiene especial relevancia en entornos donde las tareas de identificación requieren de la iniciativa del usuario.

En el modelo de datos, el rol se caracteriza como un concepto del cual heredan los distintos roles que se quieren representar. Entre una persona y un rol se establecen dos relaciones:

- **actor/desempeña.** La relación *actor* identifica dado un rol quién o quiénes son las personas que lo desempeñan. En el sentido contrario, la relación *desempeña* indica dada una persona, qué roles se encuentra desempeñando.

La relación entre las personas y los posibles roles que desempeñan se observa en la figura 4.3.

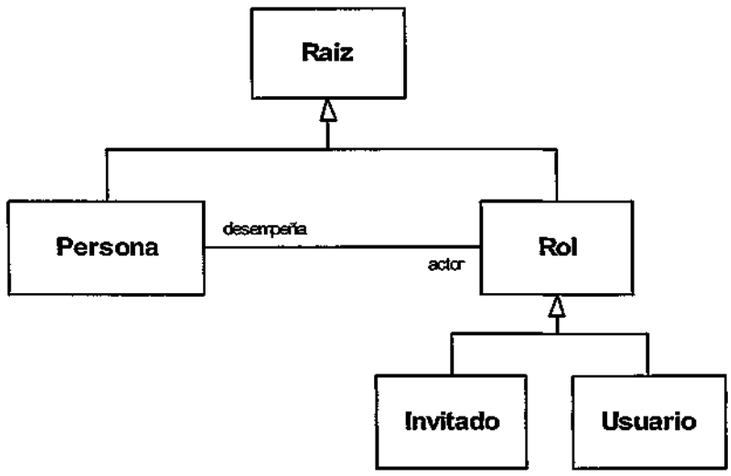


Figura 4.3: Representación gráfica entre los conceptos *Persona* y *Rol*

4.6.2. Localización espacial

Así como la identidad es el primer aspecto del contexto, en segundo lugar hay que considerar la localización (véase los apartados 2.4.1 y 4.4). Por esta razón, aunque en la literatura se pueden encontrar estudios en los que se emplea una noción de espacio más genérica [88, 191], el modelo que se va a desarrollar en este trabajo se centrará únicamente en el espacio físico.

Tal como se puso de manifiesto en el apartado 4.4, la localización establece una relación entre una entidad y un espacio físico. Hay que tener en cuenta que no a todas las entidades existentes se les puede asociar una localización espacial. Así, un recurso virtual, como una página web, no tiene por qué estar asociado a ninguna localización. Para distinguir entre las entidades con y sin localización espacial se define el concepto de cuerpo, que se empleará durante el desarrollo de este apartado para referirse a aquellas entidades que tienen asociada una localización espacial pero que no pueden contener a otras entidades.

La localización espacial consiste en una cualidad inherente a una entidad que determina su emplazamiento dentro de un espacio físico. Una entidad puede tener una única localización en el espacio, aunque ésta puede no ser conocida con total exactitud. Tal como se ha descrito en el apartado 4.4, se pueden emplear diferentes formalismos para representar el espacio físico. Habitualmente estos se dividen en dos grandes grupos: geométricos y simbólicos. Nuestro enfoque para caracterizar la localización se basa en la distinción que realiza Leonhardt en [165], donde se propone un modelo mixto geométrico y simbólico.

De esta forma la localización espacial de una entidad viene dada por dos relaciones: *estaEn* y *posición*. La primera da la localización dentro del modelo simbólico y la segunda dentro del modelo geométrico.

Los dos modelos definidos quedan ligados por la noción de lugar (véase apartado 4.6). En el modelo símbolo el espacio se divide en un conjunto de lugares relacionados entre sí mediante una ordenación parcial irreflexiva. Cada cuerpo se relaciona con el lugar en el que se encuentra en cada momento. Esta asociación dependerá de la granularidad de los sensores disponibles, y de la privacidad establecida. El modelo geométrico permite determinar con mayor exactitud la posición de un cuerpo allí donde sea insuficiente el nivel de detalle aportado por el modelo simbólico. Se puede pensar el modelo geométrico como un refinamiento del modelo simbólico. De esta forma, la localización en el modelo geométrico siempre se expresa relativa al lugar donde se halla.

Aunque ambos modelos se encuentran relacionados, y la información que mantienen debería ser consistente, no se exige que toda la información tenga que estar accesible. Dependiendo de la tecnología de localización disponible puede que sólo se pueda consultar el modelo simbólico, o que no se tenga información sobre la organización jerárquica de los lugares, pero sí sobre la posición que ocupa un cuerpo dentro del lugar.

El modelo simbólico que se propone se basa en la noción de lugar definida anteriormente (véase el apartado 4.6). Tal como se ha comentado existen dos tipos de lugares: entornos y áreas. Los primeros están acotados por barreras físicas que limitan el espacio que ocupan, mientras que los segundos vienen delimitados por fronteras virtuales.

El modelo simbólico divide el espacio físico (S) en dos partes: espacio exterior y espacio interior. Existe un único espacio exterior que abarca a todo el espacio conocido. Este espacio exterior incluye uno o más espacios interiores. Esta división entre espacio exterior e interior viene motivada porque las técnicas de posicionamiento que se emplean son distintas. Aunque las preguntas que se quieren responder son esencialmente las mismas (dónde se encuentra localizado un cuerpo

y qué otros cuerpos están más próximos) las particularidades que presentan los espacios interiores permiten particionar el espacio de forma natural empleando la estructura física que engloba al espacio interior. El modelo simbólico propuesto se centra en estructurar el espacio interior, ya que nuestro interés son las aplicaciones que operan en *entornos inteligentes*. Cada uno de los espacios interiores es un entorno que a su vez se compone de un conjunto de diferentes lugares que cumplen las siguientes propiedades:

- Un lugar nunca se puede solapar con otro lugar.
- Un entorno puede incluir a otros entornos y a otras áreas.
- Un área puede incluir a otras áreas pero no a otros entornos.
- Todas las áreas que están incluidas en un mismo lugar son adyacentes.
- La adyacencia entre dos entornos siempre se realiza a traves de una unión (véase más adelante).

La restricción de no permitir solapamiento entre dos o más lugares supone una perdida de generalidad, pero se corresponde en un alto porcentaje con las situaciones reales. Este mismo enfoque se puede encontrar en [48]. De esta manera, la localización de un cuerpo en el modelo simbólico queda unívocamente representada por el lugar en el que está contenido. La división en entornos responde a una partición natural del espacio interior según paredes y muros. Además, se añade la posibilidad de subdividir los entornos en diversas áreas facilitando la organización lógica de cada entorno.

Este modelo simbólico se basa únicamente en dos relaciones topológicas: inclusión y adyacencia. Aparte de estas dos relaciones, se pueden establecer otras relaciones topológicas entre dos regiones. Una de las teorías sobre regiones espaciales con más influencia es el cálculo *RCC-8* [5] [74]. Éste define que las relaciones topológicas entre dos regiones cualesquiera caen dentro de una de las siguientes: *disjunta*, *contiene*, *dentro*, *igual*, *adyacente*, *cubre*, *cubiertoPor* y *superpone*. Desde nuestro punto de vista esta división no se ajusta al modelo de localización que requieren los *entornos activos*. Por una parte las relaciones *cubre* y *cubiertoPor* raramente se utilizan. Por otra parte, se precisa matizar la relación de adyacencia, ya que además de saber si dos lugares son contiguos es necesario indicar si se encuentran conectados.

El modelo espacial propuesto ha sido diseñado para ajustarse a las necesidades de las aplicaciones que trabajan sobre *entornos activos*. La sencillez con que ha sido concebido le hace especialmente atractivo, tanto desde el punto de vista conceptual como desde el punto de vista de implementación. Para ello se parte de la premisa de que una distribución del espacio bastante natural consiste en dividirlo en regiones disjuntas siguiendo las fronteras que imponen los distintos habitáculos en los que está dividido un espacio interior. Como se establece una jerarquía de entornos, se permite definir distintos niveles en contención. Al permitir relaciones

[5] Region Connection Calculus

de inclusión entre los diferentes entornos se incluye la posibilidad de que convivan diferentes niveles de granularidad. La pregunta de dónde se encuentra un cuerpo queda respondida con el entorno en el que se encuentra. Otras relaciones espaciales quedan expresadas tomando el entorno como unidad. Así, los conceptos de co-localización y de cercanía se definen como todos los cuerpos que se encuentren en el mismo entorno, o en entornos adyacentes; y la distancia entre dos cuerpos se calcularía como el número de uniones que hay atravesar para llegar de uno a otro. El concepto de área se incluye en el modelo para permitir distinguir entre diferentes regiones dentro de un entorno en el que caso de que sea necesario.

El modelo geométrico queda fuera del alcance de esta tesis aunque se prevé su inclusión en un futuro. Este modelo tiene sentido como un ajuste fino de la localización dentro del propio entorno, de forma que se pueda obtener la mayor precisión posible. El espacio queda delimitado por las barreras físicas de cada entorno en particular, de forma que la posición de cada cuerpo queda expresada relativa al sistema de coordenadas de cada entorno. Así, las coordenadas del modelo geométrico sólo tienen sentido dentro de cada entorno. A partir de las posiciones se pueden calcular las relaciones espaciales necesarias entre dos cuerpos cualesquiera.

Presencia

La presencia es una propiedad espacial asociada a un lugar y que aporta información sobre la existencia de cuerpos en su interior. Esta propiedad permite representar la ocupación del lugar sin tener que conocer la identidad de los ocupantes. Éste es un caso que se da con relativa frecuencia, ya que los sensores que aportan información anónima sobre los ocupantes suelen estar disponibles en la configuración de un *entorno inteligente*. Ejemplo de estos sensores son los sensores de movimiento que miden si hay o no cuerpos desplazándose, o los sensores de paso, que permiten contar el número de personas y la dirección en que han cruzado un determinado umbral.

En función de los sensores de que se disponga, la presencia se puede describir tanto cualitativamente como cuantitativamente. Por ello, se han incluido dentro del concepto de *Lugar* dos propiedades (véase la figura 4.4): *vacío*, que indica si hay o no ocupantes en el entorno, y *numero de ocupantes*, que lleva la cuenta del número personas dentro del entorno.

Figura 4.4: La presencia como propiedades del concepto *Lugar*

Inclusión

Para representar la inclusión de un lugar dentro de otro se emplea la relación *contiene*. La propiedades de esta relación se definen exactamente igual que en [165], de forma que establece una ordenación parcial irreflexiva del conjunto de localizaciones. Esto es, tiene que cumplir las siguiente propiedades:

- Si l_2 *contiene* a l_1 entonces l_1 no puede contener a l_2.
- Si l_2 *contiene* a l_1 y l_3 *contiene* a l_2 entonces se cumple que l_3 *contiene* l_1

La relación de *inclusión* se corresponde con las relaciones de *contiene* y *cubre* del modelo *RCC-8*. La relación de inclusión también se puede aplicar entre un lugar y un cuerpo para expresar que un determinado lugar contiene un determinado cuerpo. Combinando la relación de contención entre dos lugares y entre un lugar y un cuerpo se extrae que se cumple la siguiente expresión:

- Si l_2 *contiene* a l_1 y l_1 *contiene* c entonces l_2 también *contiene* a c.

donde l_1 y l_2 son dos lugares pertenecientes a S, y c es un cuerpo del espacio.

Localización

La relación inversa a *contiene* es *estaEn*. Esta relación se puede aplicar tanto a cuerpos como a lugares. En el primer caso, la relación *estaEn* define la localización del cuerpo, mientras que en el segundo expresa que un lugar se encuentra dentro de otro.

Dado que *contiene* y *estaEn* son inversas, y que un cuerpo solo puede estar en un lugar al mismo tiempo, se cumple que:

- Si c *estaEn* l entonces se cumple c *contiene* l.

Adyacencia

Al igual que en [165], la relación de adyacencia establece que dos lugares están conectados, es decir, que desde un lugar adyacente a otro se puede ir sin atravesar un tercer lugar.

Esta relación no queda reflejada en el caso de que los lugares considerados sean áreas. Si dos o más áreas se encuentran dentro del mismo lugar no se representa si son adyacentes o no, aunque queda implícito que desde cualquier área se puede alcanzar a cualquier otra. Por el contrario en el caso de la adyacencia entre entornos sí que es necesario establecer explícitamente esta condición. Ahora bien, en vez de emplear una relación entre los dos lugares se incluye un nuevo tipo de entidad denominado unión[6] [193] que representa cómo es el elemento que conecta a los lugares adyacentes.

Existen dos tipos de uniones: libres y protegidas. Las primeras permiten el paso sin ningún tipo de obstáculos; las segundas pueden imponer restricciones al

[6] junction

paso si se cumplen ciertas condiciones. Las uniones se modelan como un cuerpo más que puede estar contenido dentro de un entorno, con la salvedad que son el único tipo de cuerpo que puede estar en varios entornos a la vez. Así, existirá una relación *contiene* por cada unión que haya dentro de un entorno.

Las uniones juegan un papel importante en cuanto a la adyacencia. Mientras que cualquier área es adyacente al área que la incluye, un entorno que contenga a otro será adyacente sólo si existe una unión entre ambos.

Las uniones a su vez guardan dos tipos de información: cuáles son los entornos que conectan y si puede permitir el paso libremente. Así, por cada entorno que conecte la unión se establecerá un relación *estaEn*. Las condiciones se representan mediante una propiedad booleana (*permiteSiemprePaso*) que indica si la unión es un paso libre o si bajo ciertas condiciones puede estar obstaculizada. La especificación de las condiciones queda fuera del modelo espacial, y entraría dentro del modelo de seguridad del entorno. El valor de la propiedad será falso en cuanto exista una barrera física que puede obstaculizar el paso a un cuerpo. Así, una puerta siempre se considerará una unión barrera ya que, independientemente de si está cerrada o no con llave, puede suponer un obstáculo para cuerpos como robots o niños pequeños. A partir de esta propiedad se puede hallar si entre dos entornos existe algún camino, si este siempre está libre o si es posible que se encuentren obstáculos al recorrerlo. El ejemplo de la figura 4.5 muestra dos habitaciones *labB403* y *Pasillo4* unidas por una puerta *PuertaB403*.

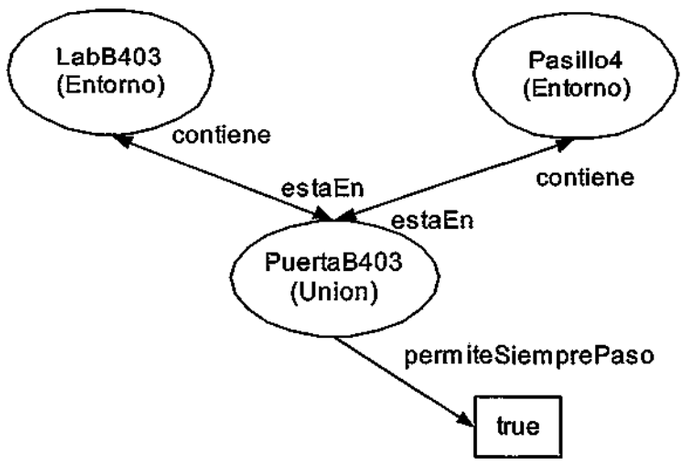


Figura 4.5: La relación espacial de *Adyacencia* representada mediante uniones

A modo de ejemplo, en la figura 4.6 se expone una representación simplificada de dos despachos de la *Escuela Politécnica Superior*. En la figura se muestra que el entorno *B403* se encuentra dividido en dos áreas, salón y oficina. En la primera se encuentra localizada la persona *montoro* mientras que en la segunda se encuentra la persona *phaya*. Este laboratorio se une al despacho *B420* a través de un pasillo *Pasillo4*, que en este momento se encuentra vacío. En cambio, en el despacho *B420* se halla localizada la persona *xavier*.

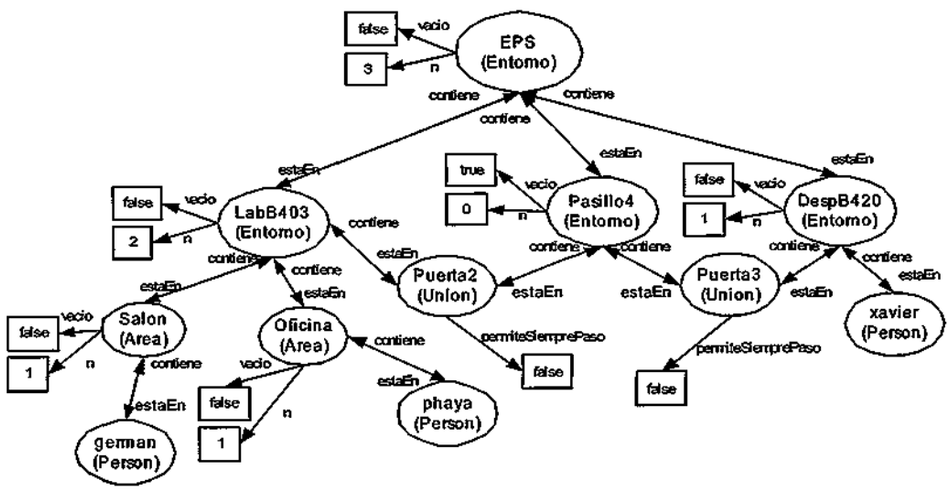


Figura 4.6: Ejemplo de instanciación del modelo de localización espacial

4.6.3. Actividad

El tercer aspecto que vamos a considerar es la actividad. El concepto de actividad refleja la ocupación de una persona en un momento dado. A partir de esta definición es complicado describir la actividad de una persona mediante un modelo computacional. Una primera dificultad surge por la variedad de actividades que una persona puede realizar. Esta diversidad no solo afecta al número de actividades sino también a la granuralidad temporal. Así, se puede considerar como actividad desde operaciones casi instantáneas, como abrir una puerta o descolgar un teléfono, hasta tareas que conllevan una duración mayor, como puede ser leer un libro, o visionar una película. A esto hay que añadir que detectar la actividad en curso que está realizando una persona puede llegar a ser un proceso difícil y propenso a errores, de cual normalmente se va a obtener información parcial. Es más, hay que tener en cuenta que una persona puede estar realizando varias actividades simultáneamente, como por ejemplo leer y escuchar música, o llevar cabo varias actividades secuencialmente, pero cambiando rápidamente entre ellas, como puede ser cocinar y fregar los platos.

A la hora de describir la actividad como información contextual se propone un enfoque de arriba-abajo que empieza desde lo más general avanzando a lo más particular. Este planteamiento trata de mitigar las dificultades expuestas anteriormente, y permite a los desarrolladores ajustar la tecnología disponible de forma que se pueda completar el modelo aunque sea parcialmente.

Como primer acercamiento a la actividad se definen dos propiedades relativas a la ocupación de una entidad. Estas son:

- **Ocupado**. Indica la disponibilidad de una persona para involucrarse en una actividad nueva. Este puede tomar dos valores: verdadero o falso.
- **No molestar**. Esta propiedad toma un valor booleano y se aplica única-

mente a entidades de tipo *Persona*. Establece que ésta desea recibir las menos interrupciones posibles para así poder concentrarse en la actividad que está realizando.

Siguiendo con la interpretación del concepto de actividad como parte de la información contextual, se han tenido presentes los problemas mencionados previamente, de forma que la descripción que se incluye se ha restringido a una definición más pragmática. Siguiendo esta línea, bajo el nombre de cada actividad se recoge un conjunto de tareas propias de una persona que, aun teniendo entidad propia, cuando se consideran como conjunto, adquieren una relevancia significativa en cuanto a las preferencias de usuario. Es decir, el criterio que se emplea se centra en considerar si el cambio de una actividad a otra produce una modificación significativa en las preferencias de esta persona y/o del estado del entorno. En caso contrario, el conjunto de tareas considerado no constituye una nueva actividad sino que se tendría que incluir dentro de alguna de las actividades ya existentes. Por ejemplo, dos de las actividades que se consideran son estar trabajando y estar descansando. El cambio de una a otra conlleva una modificación en las condiciones del entorno, ya que comúnmente el usuario requiere una configuración de los recursos y del propio entorno sensiblemente distinta. En cambio, coger un tenedor, llevarse comida a la boca, beber de un vaso, aun siendo tareas del usuario no se consideran como actividades separadas, sino partes de una actividad que las engloba denominada alimentarse.

En el marco que se ha establecido, se espera que las actividades presenten una cierta inercia, esto es, que la interrupción de una actividad sólo ocasione un cambio de actividad si esta interrupción se prolonga en el tiempo. Así por ejemplo, si el usuario está desempeñando la actividad de trabajar, un breve receso no se debería tener en cuenta para determinar un cambio en la actividad.

La actividad de un usuario se entiende en dos ejes complementarios. El primero hace referencia al fin mismo de la actividad donde se ven reflejados los usos y costumbres sociales. Dentro de este eje, se consideran dos niveles de actividades. En un primer nivel, se distinguen tres grandes bloques de actividades: obligaciones, actividades de tiempo libre, y cuidado personal. Las obligaciones hacen referencia a actividades que se desarrollan por algún tipo de compromiso adquirido con un tercero o consigo mismo. Las actividades de tiempo libre engloban a aquellas que el usuario realiza para obtener bienestar. Finalmente se encontraría el cuidado personal, donde se engloban aquellas actividades que tienen como fin mantener o mejorar el estado físico de la persona.

En un segundo nivel se subdivide cada uno de estos bloques en actividades más concretas. Así, las obligaciones pueden ser impuestas por algún motivo laboral o por una tarea doméstica. El tiempo libre se reparte en ocio y en descanso, y el cuidado personal se divide en asearse, tomar medicación y alimentarse.

La jerarquía que se propone se muestra en la figura 4.7.

El segundo eje considera la actividad en cuanto al modo en que se está llevando a cabo, el cual se encuentra estrechamente relacionado con el instrumento o instrumentos empleados. Esto es, la propia descripción de la actividad puede incluir a las herramientas sin necesidad de especificarlas por separado.

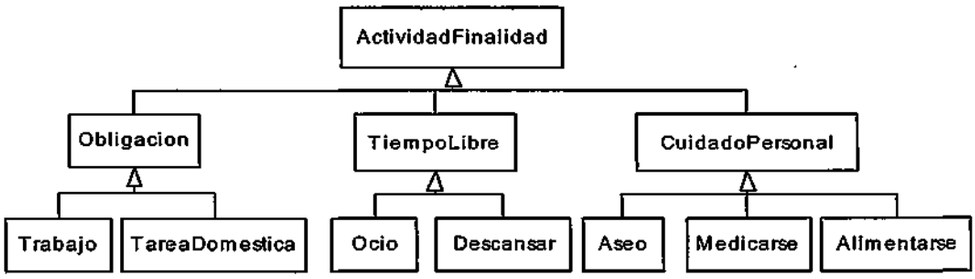


Figura 4.7: Organización jerárquica de las actividades atendiendo al fin que persiguen

La lista de actividades que se contemplan en un primer nivel es la siguiente:

- **Escribiendo.** El usuario se encuentra realizando un escrito mediante un instrumento que permita trazar signos sobre un papel.
- **Escuchando.** Esta actividad se refiere exclusivamente al hecho de que el usuario se encuentra atendiendo a una composición musical.
- **Leyendo.** El usuario se encuentra interpretando un texto escrito en papel.
- **Visionando.** La interpretación de esta actividad se restringe a que el usuario se encuentra viendo imágenes cinematográficas o televisivas en una pantalla.
- **Conversando.** Esta actividad hace referencia a que el usuario se encuentra comunicándose con otro usuario mediante la voz.
  - **Conversando por teléfono.** Esta actividad es un caso particular de la anterior en la que se indica que el usuario está empleando el teléfono para realizar la conversación.
- **Usando el ordenador.** Esta actividad indica que el usuario se encuentra empleando el ordenador.

La combinación de los dos ejes que se han presentado constituyen la actividad de una persona. Ésta se representa mediante una entidad que engloba la información tanto sobre el fin que se persigue cómo el modo en que se desempeña. Esta entidad se denomina *Actividad*, y queda relacionada con la persona que la desempeña mediante la relación *involucrado-en*. En la figura 4.8 se representa gráficamente la relación entre personas y actividades.

Cada entidad de tipo *Actividad* tiene dos propiedades que reflejan para qué se está realizando y cómo se está llevando a cabo la actividad. Los valores de cada una de las propiedades reflejan los dos ejes de actividad vistos anteriormente. Ambos son independientes entre si, y se pueden dar múltiples combinaciones en la vida cotidiana. Por ejemplo, una persona puede estar leyendo un libro porque es parte de su trabajo, o porque es parte de su tiempo libre. Siendo la misma actividad, el fin es distinto, lo que puede llevar a diferentes acciones por parte

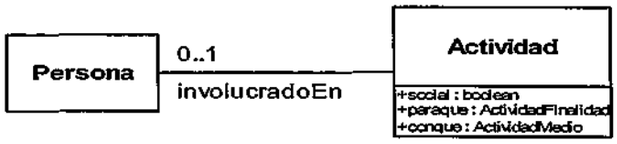


Figura 4.8: Relación entre personas y actividades

del sistema. Para completar el concepto de *Actividad* se añade una propiedad booleana denominada *social* que indica si la actividad en curso que desempeña la persona la está realizando en conjunción con otros personas, en el caso de que sea verdadera, o de forma individual, en el caso de que sea falsa.

La clasificación que se plantea constituye un modelo mínimo que se puede seguir extendiéndo dependiendo del dominio de aplicación. Así se podrían incluir otros tipos de actividades más concretas como afeitarse, cocinar, defecar, dormir, ducharse, fregar, lavarse las manos, limpiar, ordenar, limpiarse los dientes, orinar, planchar, etc.

El enfoque se ha seguido hasta ahora trata de condensar la información relativa a la actividad en unas pocas variables que sean suficientemente representativas. Esta aproximación simplifica el modelo pero elimina detalles sobre lo que ocurre en el entorno que pueden ser importantes. Se precisa proveer también de un planteamiento que permita capturar las relaciones entre los ocupantes del entorno y los recursos que se encuentran en activos. En este sentido, se definen dos relaciones bidireccionales entre una persona y cada uno de los recursos que emplea y/o transporta. Éstas son:

- **Usa/UsadoPor.** Esta relación identifica qué recursos están siendo utilizados por qué usuarios.
- **Transporta/TransportadoPor.** Esta relación se establece entre una persona y los recursos móviles que lleva consigo.

Gráficamente se puede observar en la figura 4.9 cómo se relacionan las personas y los recursos.

Finalmente, también se puede considerar parte del contexto el estado en que se encuentran los recursos del entorno, ya que éste puede dar una indicación sobre la actividad que se está desarrollando. Por esta razón todos los recursos tienen la propiedad *estado*, que puede tomar distintos valores dependiendo del recurso. Así, mientras que para una luz el estado es encendido o apagado, para un teléfono puede ser colgado, descolgado, comunicando, llamando o conversando.

4.6.4. Ejemplo de representación del contexto primario

Para acabar este apartado donde se han estudiado los tres aspectos principales del contexto primario, en la figura 4.10 se puede observar un ejemplo simplificado de una posible composición del modelo. Los círculos simbolizan entidades, y cada

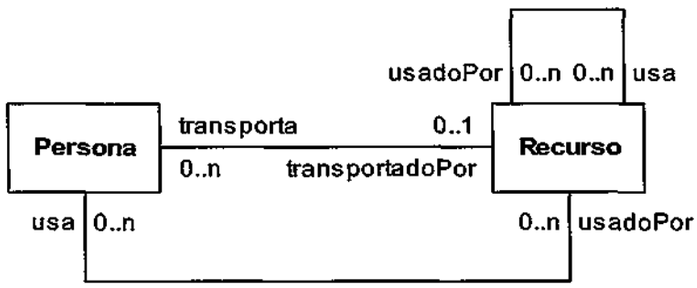


Figura 4.9: Relación entre personas y recursos

nodo entidad se compone de un nombre y un tipo. Este último se representa en cursiva y entre paréntesis. Así, el ejemplo representa cinco entidades: dos dispositivos, uno que se denomina *Altavoz1* y otro *Lamp1*; una habitación conocida como *Lab403*; una persona que se llama *Dave*; y la actividad en la cual se encuentra involucrado. Las propiedades se definen por su nombre y su valor encuadrado en rectángulo. Para asociar una propiedad a una entidad se usa el nombre de la propiedad. Así, se puede observar que la luz *Lamp1* tiene una propiedad denominada estado y que su valor es encendido. Las relaciones entre las entidades se presentan como vértices. Si el arco termina en dos flechas significa que la relación es bidireccional, en caso contrario es unidireccional. Tal como se deduce de la figura, la habitación *Lab403* tiene en su interior una persona (relación *contiene* de la entidad *Lab403*) y dos recursos. En este caso, se han incluido también las relaciones recíprocas, por lo que también se podría interpretar que la persona *Dave* se encuentra dentro de la habitación *Lab403* (relación *estaEn* de la entidad *Dave*). Finalmente, la entidad *Dave* está *involucrado-en* una actividad cuyo *paraque* es *TiempoLibre* y que consiste en escuchar música.

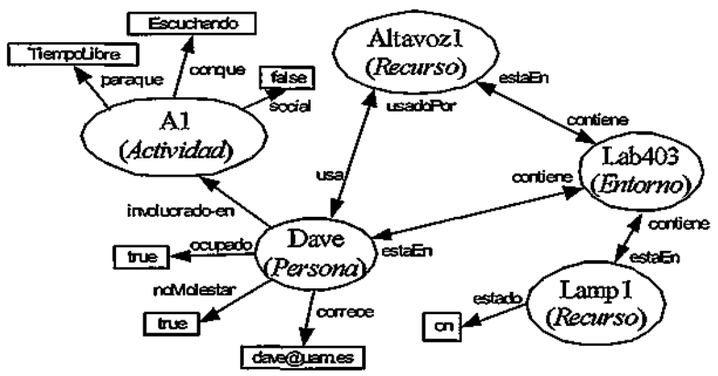


Figura 4.10: Ejemplo de una realización del modelo: entidades, relaciones y propiedades

4.7. Contexto secundario

No es posible abarcar en un único modelo toda la información contextual de una situación. El contexto primario que se analizó en el apartado anterior pretende capturar las propiedades y relaciones que se usan más frecuentemente en la literatura para definir el contexto, pero es necesario poder añadir nuevas características al modelo anterior que reflejen la parte de la información contextual que depende del dominio de aplicación. Por ejemplo, una parte fundamental del contexto primario es el entorno. Ahora bien, existen diversos tipos de entornos, cada uno con sus propias particularidades. Ésto lleva a que las *aplicaciones sensibles al contexto* que trabajan dentro del entorno requieran cierta información contextual particular. Por ejemplo, un reproductor de audio contextual puede necesitar la lista de canciones favoritas del usuario y las relaciones con el tipo de actividad que esté desarrollando. De esta forma, la aplicación puede decidir qué canción será la siguiente en función de si el usuario está trabajando o descansando. Toda esta información adicional se denomina contexto secundario. Esta información extiende el contexto primario añadiendo nuevos conceptos, entidades, propiedades y relaciones.

4.7.1. Contexto secundario para un hogar digital

El hogar constituye un excelente banco de pruebas para los desarrollos de la *Computación Ubicua*. Las personas consideran el hogar un espacio privado donde prima el bienestar en forma de tranquilidad y descanso. Este bienestar se ve favorecido por una interacción transparente como la que promueve la *Computación Ubicua*, por lo que no es de extrañar que sea uno de los dominios particulares donde se están desplegando más *entornos activos* (véase el apartado 2.5.1).

La información contextual que se maneja dentro del hogar presenta ciertas particularidades que tienen que ser contempladas por separado. Aspectos como el confort de los habitantes, el consumo de los dispositivos, y la seguridad son preocupaciones frecuentes en los usuarios de los hogares. Estas particularidades enriquecen el modelo de contexto primario sin alterar la esencia del mismo.

La ampliación del modelo consiste en la inclusión de nuevos conceptos, propiedades y relaciones. El primer paso será definir la estructura física de un hogar. Para ello, se precisa añadir tres nuevos conceptos que heredan de *Entorno*:

- **Habitación.** Es la denominación genérica para el tipo de entornos que se encuentran en un hogar. Las distintas habitaciones de un hogar constituyen una partición física que afecta a la representación de la localización espacial de los habitantes. Las habitaciones se destinan a diferentes usos, aunque estos pueden cambiar según las necesidades de los habitantes. A su vez, una habitación se puede subdividir en áreas, que reflejen zonas donde se realizan distintas actividades. Así, por ejemplo, un salón-comedor se puede dividir en un área para comer y una para descansar.

- **Hogar.** El hogar engloba a una serie de habitaciones donde habita un conjunto de personas que constituyen una unidad organizativa, que puede ser

una familia, un grupo de amigos, varios desconocidos, etc. Un hogar se considera tanto una unidad espacial como una unidad administrativa. Por un lado, establece una relación espacial entre una serie de habitaciones. Por otro lado, el hogar establece unos límites administrativos, de forma que las políticas y preferencias que se apliquen al hogar afectan a todas las habitaciones que lo conforman, y no a otras habitaciones pertenecientes a otros hogares. La administración de cada hogar recae sobre los dueños del mismo, que son los últimos responsable sobre las decisiones que se tomen.

- **Edificio.** Un edificio se define como una construcción que alberga una agrupación de entornos. Un hogar puede incluir más de un edificio, y viceversa, un edificio puede contener más de un hogar. Además, los edificios pueden incluir habitaciones comunes que no estén asociadas a ningún hogar. De nuevo, al igual que en el hogar, un edificio también establece unos límites administrativos. No obstante, las decisiones que se tomen sobre el edificio únicamente afectan a los lugares comunes, y no a los hogares: estos se rigen por sus políticas particulares.

La información contextual relacionada con estos tres tipos de entorno se caracteriza tanto por un conjunto de propiedades como por los recursos que habitualmente se pueden encontrar. Las propiedades que determinan el contexto de un hogar en general, y de una habitación en particular, se han divido en dos partes: aquellas relacionadas con el confort, y aquellas relacionadas con la seguridad. Estas propiedades se describen en los siguientes apartados.

Confort

El confort aporta una medida del bienestar de los habitantes. Dentro del confort se incluyen parámetros físicos del entorno que normalmente los habitantes ajustan para obtener mayor bienestar. El confort incluye tres conceptos, cada uno con una serie de propiedades que lo describen:

- **Temperatura.** Se representa mediante el concepto *Temperatura* que consta de tres propiedades: *valor*, *valorDeseado* y *unidad* de medida que se emplea. Este concepto se asocia con entorno mediante la relación *temperaturaActual*.

- **Luminosidad.** Mide la claridad del entorno. Se representa mediante el concepto *Luminosidad* que consiste en las propiedades: *valor*, *valorDeseado* y la *unidad* de medida que se emplea. Este concepto se asocia con entorno mediante la relación *luminosidadActual*.

- **Humedad.** Mide el nivel de humedad del entorno. Se representa mediante el concepto *Humedad* que consta de las propiedades: *valor*, *valorDeseado* y la *unidad* de medida que se emplea. Este concepto se asocia con entorno mediante la relación *humedadActual*

- **Nivel ruido.** Mide el nivel de ruido ambiente del entorno. Se representa mediante el concepto *NivelRuido*. Éste consta de dos propiedades: *valor* y

valorDeseado. El nivel de ruido puede tomar uno de los siguientes valores cualitativos: silencio, bajo, medio, elevado.

En la figura 4.11 se puede observar cómo quedan relacionados en el modelo los conceptos de *Lugar* y el *Confort*

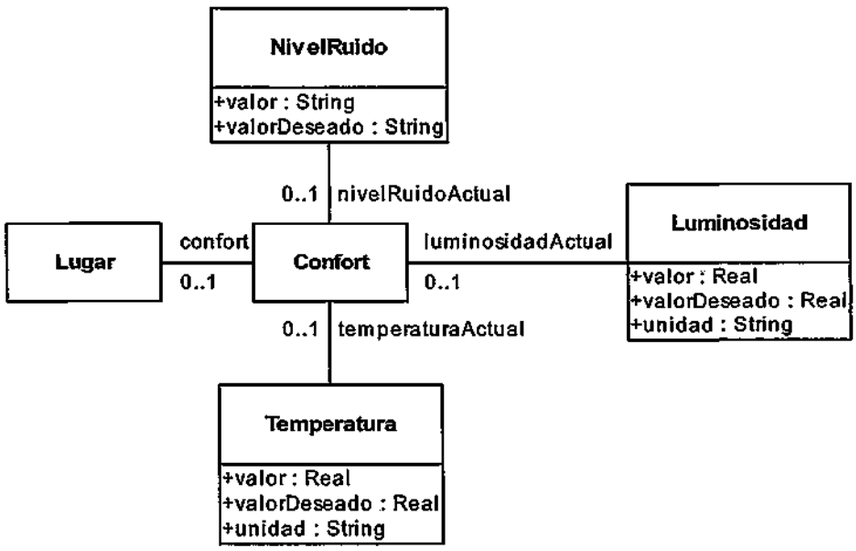


Figura 4.11: Modelo del confort en el hogar digital

Seguridad

Dentro del epígrafe de seguridad se incluyen tanto riesgos procedentes de actuaciones deliberadas como accidentales. Ejemplos del primer caso son una intrusión, un robo, un sabotaje... mientras que ejemplos de los segundos son una inundación, un escape de gas, un corte luz... Se denomina alerta a cualquier suceso que puede provocar una intervención de las aplicaciones encargadas de velar por la seguridad. Una posible clasificación de las alertas consiste en dividirlas según si son deliberadas o accidentales. Esta clasificación presenta el problema de que no siempre se pueden establecer con certeza las causas de una alerta, ya que ésta se puede haber producido por diversos motivos. Así, un corte de luz puede producirse por el efecto de una caída de tensión en la central eléctrica, o por un corte intencionado en la línea.

En cambio, otra posible clasificación divide las alertas en función qué o quien se encuentra en riesgo. Así, se dividen en:

- **Alertas del entorno.** Aquellas que afectan a todo el espacio físico del entorno. Éstas incluyen desde intrusiones en el entorno hasta catastrofes como incendios, fugas de agua, escapes de gas...

- **Alertas de personas.** Aquellas que afectan a la integridad física de la persona.

- **Alertas de un bien.** Aquellas que se generan por un malfuncionamiento de un recurso en particular. Posibles ejemplos son la rotura de un cristal, o una cerradura que ha dejado de funcionar.

- **Alertas del funcionamiento del entorno.** Son similares a las anteriores, pero en vez de afectar a un recurso afectan a una funcionalidad del entorno. Por ejemplo, una caída de tensión o un corte de la línea telefónica.

Hay que tener en cuenta que cuando el modelo se refiere al entorno puede tratarse tanto de todo el hogar como de una habitación en concreto.

Todas las alertas se representan mediante el concepto *Alerta* y se incluyen dentro del concepto *Seguridad*, el cual depende de cada entorno en particular. La seguridad de cada entorno guarda las alertas que se han producido mediante la relación *alerta*. Cada *Alerta* que se produce incluye las siguientes propiedades y relaciones:

- **descripición.** Propiedad que incluye una descripción textual de la alerta producida.

- **localizadaEn.** Relación que indica en qué entorno se ha producido la alerta.

Existen distintos tipos de alertas que, aparte de las anteriores propiedades, pueden incluir otras con información más particular. Éstas se detallarán posteriormente, agrupándolas según las cuatro categorías anteriores. Un tipo de alertas que se escapa del modelo son las relativas a violación de la información, como puede ser el conocimiento desautorizado del estado de alguna entidad del entorno. Éstas entrarían dentro de un modelo de privacidad y seguridad informática.

Las aplicaciones de seguridad tienen diferentes funcionalidades, que se pueden resumir en detección de alertas, respuesta a esas alertas y prevención de riesgos. Cualquier aplicación de seguridad en funcionamiento se espera que realice constantemente la función de detección. En cambio, las funciones de respuesta y prevención no tienen por qué estar activas todo el tiempo. Por el contrario, los habitantes pueden preferir desactivar alguna de las dos funcionalidades. La respuesta y la prevención se diferencian en cuanto al tipo de comportamiento de las aplicaciones. La funciones de respuesta siempre hacen referencia a un comportamiento pasivo de las aplicaciones. Es decir, éstas se disparan como reacción a la detección de una determinada incidencia. En cambio, las funciones de prevención implican un comportamiento activo, de manera que éstas se ejecutan automáticamente. Dependiendo del grado de seguridad se habilitan o deshabilitan las diferentes funcionalidades. La correspondencia entre los valores y los modos de funcionamiento se establece en la tabla 4.1. Las columnas representan los distintos valores que puede tomar el grado de seguridad; mientras que en las filas se encuentran las diferentes funcionalidades. Al cruzar una columna con un fila se obtiene si una función está activa para un determinado grado de seguridad.

	Bajo	Normal	Alto
Detección	SI	SI	SI
Respuesta	NO	SI	SI
Prevención	NO	NO	SI

Tabla 4.1: Relación entre el grado de seguridad del entorno y las funciones de seguridad activas

Para cada tipo de alerta se define el grado de seguridad como una propiedad del concepto *Seguridad*, de forma que se puede cambiar de forma individual el comportamiento del entorno para cada alerta. Además, las medidas concretas que se lleven a cabo —ya sea pasiva o activamente— también pueden depender de otras variables contextuales, aparte del grado de seguridad. Un ejemplo podría ser un hogar con los habitantes fuera de casa y un grado de seguridad alto. Ambos hechos podrían disparar aplicaciones que simulan que un persona se encuentra dentro, conectando y desconectando luces y aparatos electrónicos. En cambio, este tipo de prevención no tiene sentido si los habitantes se encuentran en el hogar.

Si una aplicación lleva a cabo una acción de respuesta que afecta a un entorno, el contexto del mismo cambia. Para reflejar el hecho de que un entorno está siendo sometido a una acción de seguridad se incluye la propiedad *estado* dentro del concepto *Seguridad*. Esta propiedad puede tomar los valores de normal o emergencia.

A continuación se analizan las cuatro categorías de alerta propuestas.

Alertas del entorno Dentro de esta categoría se incluyen cuatro tipos distintos:

- **Intrusión.** Una intrusión hace referencia a aquellas incidencias en las que una persona no identificada invade un entorno.
- **Incendio.** Una incidencia de incendio aparece cuando se detecta que se ha producido un fuego en alguna parte del entorno.
- **Escape de Gas.** Esta incidencia se produce cuando se ha detectado un escape de gas.
- **Inundación.** Similar a las dos anteriores pero en el caso de que se detecte una fuga de agua.

Alertas de las personas Una alerta sobre una persona se produce cuando se pone en peligro su integridad física. Se dividen en *Agresiones* y *AlertasMedicas*. Las primeras hacen referencia a acciones violentas de un tercero sobre la persona, mientras que las segundas incluyen problemas relacionados con la salud médica. Tanto unas como otras incluyen la siguiente propiedad:

- **personaAfectada.** Relación que indica qué persona ha sido objeto de la agresión.

Alertas de los bienes Las alertas que se pueden producir sobre un bien se dividen en daños y robos. Los daños sobre un bien se denominan *Desperfecto*. Por ejemplo, una rotura de un cristal o una cerradura. También incluyen malfuncionamiento de dispositivos como puede ser el corte en la señal video que emite una cámara, o el corte señal audio de una alarma. Los robos indican que un recurso físico ha sido desplazado por algo o alguien no identificado. Estos se representan mediante el concepto *Robo*. Ambos tienen la siguiente relación:

- **recursoAfectado.** Relación que indica qué recurso ha sido objeto del daño o robo.

Alertas del funcionamiento del entorno. Estas alertas afectan al correcto funcionamiento del entorno o de alguno de sus recursos. Se producen debido a que una parte crítica del entorno ha dejado de funcionar correctamente. Este tipo de alertas incluyen recursos que no están directamente modelados, como son los cortes en línea telefónica, o cortes en el suministro eléctrico.

Si el recurso afectado está modelado en la pizarra se indica con la siguiente relación:

- **recursoAfectado.** Relación que indica qué recurso está funcionando incorrectamente.

En la figura 4.12 se muestra un resumen del modelo de seguridad.

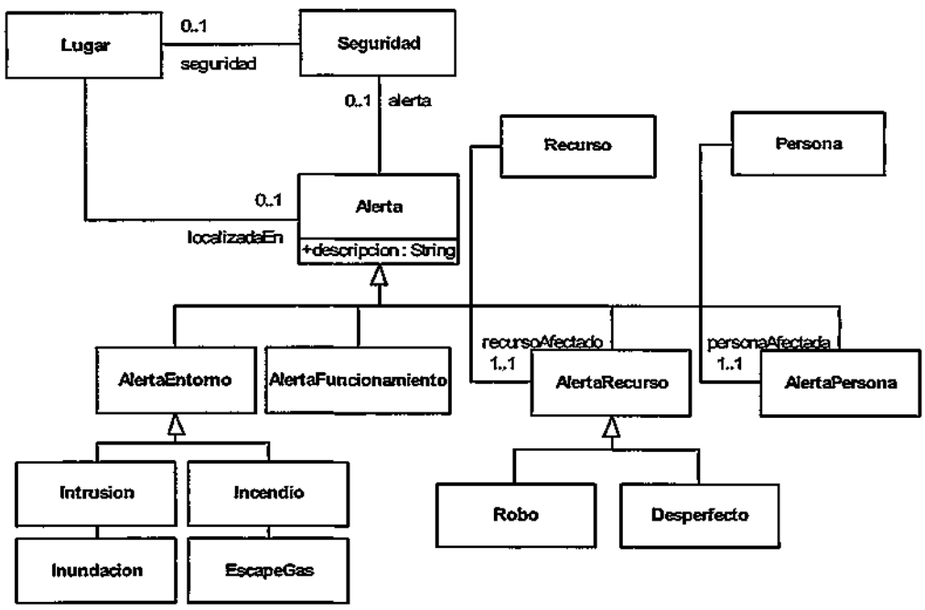


Figura 4.12: Modelo del confort en el hogar digital

Recursos

Finalmente, en un hogar se pueden encontrar también múltiples dispositivos y electrodomésticos comunes. El contexto se modifica dependiendo de los dispositivos que el usuario tenga a su alcance. Cada aparato del hogar queda representado por una instancia que contiene información sobre su estado y sus relaciones con el resto del entorno. En la figura 4.13 se puede observar cómo se particulariza el concepto *Recurso* en los distintos *Dispositivos* que se pueden encontrar en un hogar inteligente.

El concepto *Dispositivo* se ha dividido en dos categorías: *DispositivoEntrada* y *DispositivoSalida*. Los primeros representan dispositivos que consumen información mientras que los segundos incluyen aquellos que la producen. *DispositivoEntrada* incluye la subcategoría *Sensor*, que representa a un conjunto de dispositivos encargados de detectar cambios en el entorno (véase el apartado 3.2.1). El resto de dispositivos de entrada comprenden dispositivos capaces de capturar información audio-visual y dispositivos genéricos de entrada de datos como un ratón o un teclado. Análogamente, la categoría *DispositivoSalida* presenta una subcategoría denominada *Actuador*. Estos se llaman así porque son dispositivos capaces de actuar sobre el entorno (véase el apartado 3.2.3). La división en *Sensores* y los *Actuadores* viene motivada porque ambos constituyen la base para el control del entorno (véase el apartado 3.2.2).

Cada *Dispositivo* tiene al menos cuatro propiedades (véase la figura 4.14):

- **Estado.** Esta propiedad puede tomar, al menos, los valores encendido o apagado.
- **Valor.** Propiedad que al modificarse cambia el estado del dispositivo. Esta propiedad se separa de la propiedad **estado**, ya que al intentar cambiar el estado de un dispositivo, este cambio no siempre tiene que producir una modificación del estado.
- **Operativo.** Valor booleano que indica si el dispositivo se puede utilizar.
- **Potencia.** Característica de un dispositivo que indica la potencia máxima que consume. Esta propiedad modela cuánta energía requiere el dispositivo por unidad de tiempo. Como es posible que este valor no se conozca o no se pueda estimar, por defecto es cero. Mediante el consumo y el tiempo de uso de cada dispositivo se puede calcular el consumo del hogar, y tomar decisiones para rebajarlo dependiendo del contexto.

4.7.2. Contexto secundario para flujos de información continuos

Actualmente, el empleo del audio y del vídeo en los hogares se limita al uso de aparatos que operan aisladamente. Televisores, cadenas de música, altavoces, dvds, vídeos, teléfonos, cámaras de vigilancia... junto con un sinfín de dispositivos

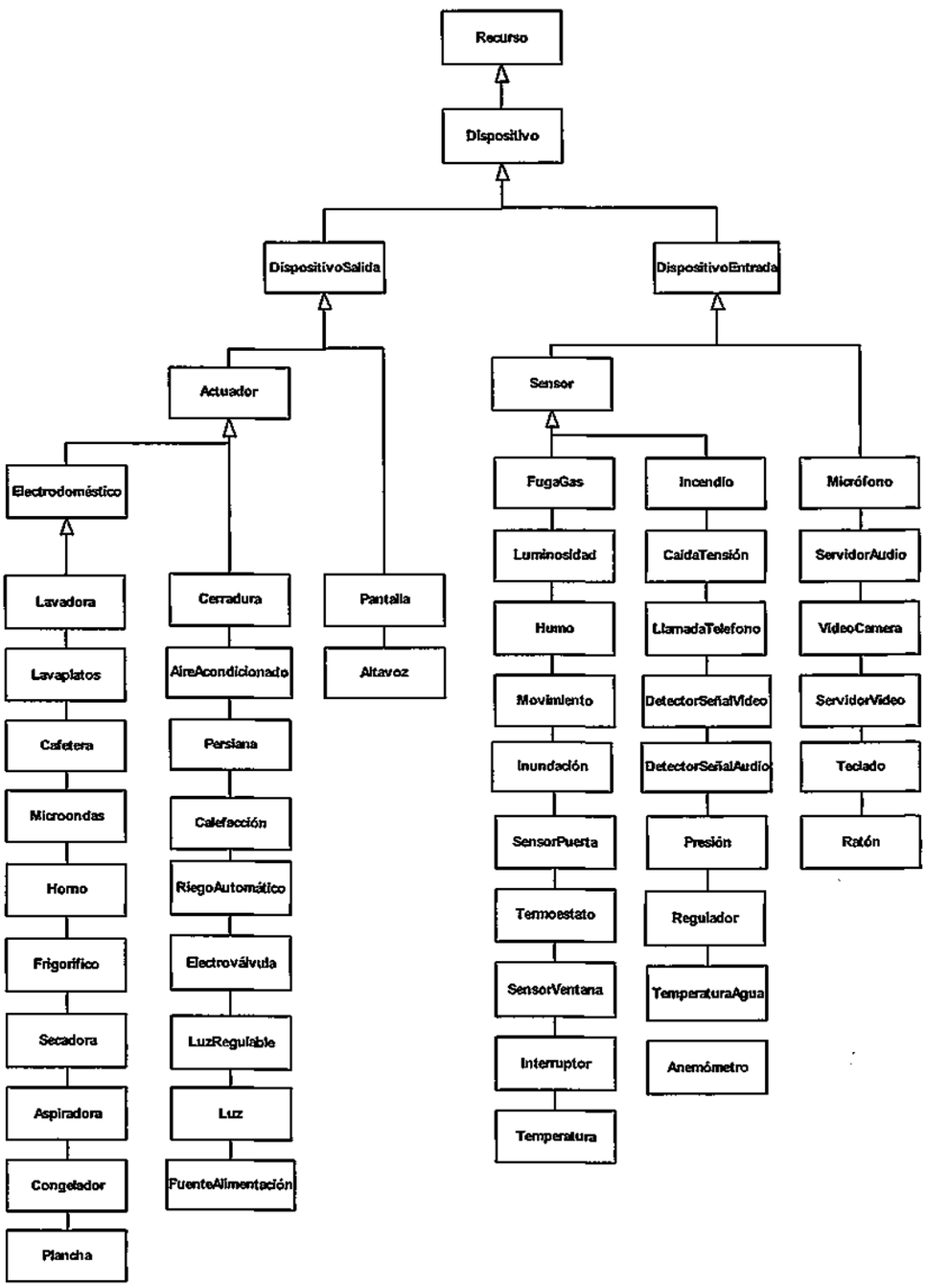


Figura 4.13: Jerarquía de dispositivos

Figura 4.14: Propiedades de un dispositivo

audio-visuales portátiles forman una extensa oferta donde el usuario elige el más idóneo para cada situación. Combinar varios de estos dispositivos suele ser una tarea ardua que requiere un esfuerzo adicional para el usuario, ya sea económico o temporal. La configuración de las interconexiones se basa en conexiones físicas realizadas a medida, que tienen que ser cambiadas manualmente cada vez que se quiere reconfigurar. Más allá de los dispositivos analógicos, que tienden a desaparecer, actualmente gran parte de los dispositivos multimedia tienen incorporados conectores digitales estándar que facilitan su interconexión (véase el apartado 3.2.2). Aun así, todavía se está lejos de una red multimedia completamente ubicua en la que cada dispositivo esté disponible para el resto en cualquier momento. En consonancia con otras iniciativas [99], creemos que en el futuro existirá un uso ubicuo de las tecnologías audiovisuales en el cual tendrán cabida nuevas aplicaciones y servicios creados a partir de la interconexión de diferentes clases de dispositivos. Para ello se precisa de una infraestructura que permita acceder a todos los aparatos a través de una única red multimedia. En el nivel lógico, se requiere que los dispositivos se puedan configurar automáticamente de forma que se posibiliten tanto aplicaciones proactivas, como a demanda de los usuarios.

Para facilitar un configuración de los dispositivos multimedia sensible al contexto, se precisa disponer de la representación de los flujos de información continua que se intercambian entre los distintos dispositivos audio-visuales. Estos flujos representan cómo los datos multimedia —imágenes, audio y vídeo— se distribuyen a través de la habitación.

Para modelar estos flujos se añaden dos nuevos conceptos *Productores* y *Consumidores*. Cada productor generará un tipo de datos multimedia que puede ser recibido por el consumidor correspondiente. Los distintos tipos de productores y consumidores se obtienen subclasificando los anteriores conceptos. En un primer nivel se encontrarían ocho tipos nuevos que se corresponderían a *ProductorAudio*, *ProductorVideo*, *ProductorImagen*, *ProductorAudioVideo*, *ConsumidorAudio*, *ConsumidorVideo*, *ConsumidorAudioVideo*, *ConsumidorImagen*. En un segundo nivel aparecería un nuevo tipo por cada formato específico, definido por la extensión MIME [104].

Cada dispositivo multimedia tiene al menos asociado un tipo de flujo multimedia. Esta asociación entre dispositivo y flujo se realiza mediante una relación denominada *maneja*. Dependiendo de si es un dispositivo de entrada o de salida tendrá asociado un consumidor o un productor respectivamente. Si el dispositivo es capaz de manejar varios flujos independientemente quedará reflejado con una relación por cada flujo.

La relación que expresa que un productor y un consumidor están conectados

se denomina *estaConectadoA*. Ambos tienen que coincidir en el tipo de flujos que son capaces de manejar. Por ejemplo, un *ProductorAudioMP3* se puede conectar a cualquier *ConsumidorAudioMP3*. Cada vez que se crea un relación *estaConectadoA* se refleja que se ha establecido un flujo de información continua entre un productor y consumidor en el mundo físico. De la misma manera, que cuando se destruye, se refleja que el flujo ha cesado.

Un problema que hay que asumir es la heterogeneidad de tecnologías disponibles para la difusión de audiovisual. Por un lado, hay que tener en cuenta que durante cierto tiempo convivirán dispositivos digitales y analógicos, lo cual implica que no todas las conexiones van a ser posibles siempre. Por otro lado, incluso en un entorno enteramente digital pueden existir distintas redes multimedia que no puedan interoperar. Para reflejar las distintas conexiones posibles se añaden dos relaciones: *conexionPermitida* y *conexionDenegada*. Cada relación indica que cierto dispositivo tiene la posibilidad o no de conectarse a otro. No es necesario que un *Productor* especifique todas sus posibles conexiones con todos los *Consumidores*. Las conexiones que no estén definidas toman un valor por defecto que se establece mediante la propiedad booleana *conexionPermitidaDefecto*. Esta propiedad pertenece a cada *Productor*. Si el valor es verdadero indica que por defecto se permiten todas las conexiones excepto aquellas que se indiquen como denegadas. En cambio, un valor falso indica que sólo se permiten aquellas conexiones que se definan explícitamente.

Por ejemplo, la conexión analógica entre una minicadena y dos altavoces, gobernada por un relé, se representaría por dos relaciones *conexionPermitida* entre la entidad *ProductorAudio* que modela la minicadena y las dos entidades de tipo *ConsumidorAudio* correspondientes a cada altavoz. Los extremos de la relación *conexionPermitida* pueden sustituirse por toda una clase de dispositivos, de tal forma que se pueden establecer relaciones entre tipos de dispositivos.

En la figura 4.15 se puede observar cómo queda el modelo completo.

4.8. Espacio de nombres

El modelo de información contextual que se ha desarrollado en los apartados anteriores plantea una separación del contexto en dos categorías. La primera engloba los conceptos y relaciones básicos para una *aplicación sensible al contexto*. La segunda se compone de un conjunto de esquemas que dependen del dominio de aplicación, y de los que se han presentado dos ejemplos. En este apartado se describe cómo se relacionan estas dos categorías a través del espacio de nombres. Esta relación se basa en que las entidades pertenecientes al contexto primario sirven como índices a cualquier parte del modelo, de forma que la información de contexto secundaria es accesible a través de las entidades de contexto primario.

El espacio de nombres propuesto define los mecanismos que permiten referenciar a un nodo del grafo para poder realizar posteriormente algún tipo de operación (por ejemplo cambiar el valor de una propiedad, borrar una entidad del grafo, añadir una relación, etc.)

Para localizar uno o más nodos en el grafo se precisa de una ruta. Una ruta es

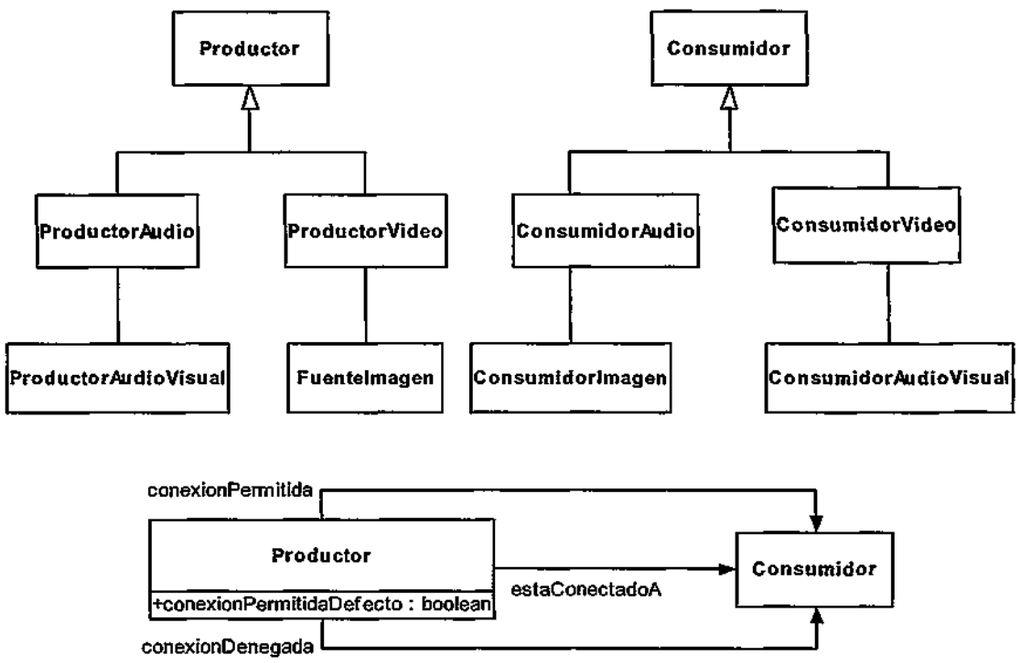


Figura 4.15: Contexto secundario añadido para modelar flujos de información continua

una serie de identificadores separados por el carácter /, donde cada identificador referencia a una parte del grafo. Cada ruta se puede construir de varias formas: el primer identificador que aparece en la ruta se emplea para diferenciar el mecanismo de nombres que se utiliza.

La ruta más sencilla que se puede construir permite localizar un nodo entidad. Para ello, el nombre de una entidad constituye un identificador único que le permite ser distinguido de cualquier otra instancia del modelo. Si se conoce este identificador, basta con emplear este nombre para referenciar unívocamente a la entidad. En este caso la ruta comienza con la palabra clave *name* seguida del nombre de la entidad. En el ejemplo de la figura 4.10 la ruta de la entidad *Lamp_1* es */name/lamp_1*.

Para referenciar una propiedad no se puede utilizar directamente el nombre de la propiedad, ya que éste no es único. La única posibilidad es extender la ruta de la entidad a la cual pertenece la propiedad. Para ello se añade a la ruta de la entidad la palabra clave *props* seguida del nombre de la propiedad. Siguiendo con el ejemplo de la figura 4.10 la ruta que le corresponde a la propiedad *Status* de la entidad *Lamp_1* es */name/lamp_1/props/status*

También se puede localizar un nodo de una entidad a partir de otra entidad con la que estuviera relacionado. Para ello se construye la ruta de la primera entidad y se le añade el nombre de la relación seguida del nombre de la entidad con la que se relaciona. En el ejemplo de la figura 4.10, la ruta */name/lab403/contiene/lamp_1* permite referenciar la entidad *lamp_1*. Esta forma de obtener la ruta no tiene mucho sentido en este ejemplo teniendo en cuenta que el nombre de la entidad lamp_1 es único, y que por tanto —como se ha explicado— es más sencillo construir la ruta anteponiendo la palabra *name*. Sin embargo, cobra importancia debido a que se permite el uso del carácter comodín *, que permite referenciar a más de una entidad simultáneamente. De esta forma, la ruta */name/lab403/contiene/** permitiría obtener todas las entidades relacionadas con la entidad *Lab403* mediante la relación *contiene*, es decir, todos los recursos y las personas de la habitación *Lab403*. En el caso del ejemplo de la figura 4.10 se correspondería con las entidades *Altavoz_1*, *Lamp_1* y *Dave*.

De igual forma, el carácter comodín se puede emplear en la ruta para referenciar a todas las propiedades de una entidad. Basándonos en el ejemplo anterior, la ruta */name/dave/props/** obtiene las propiedades *correoe*, *ocupado* y *noMolestar*. Además, si colocamos el comodín justo después del nombre de la entidad, se referencian tanto las propiedades como todas las entidades relacionadas. Así, */name/dave/** obtiene, aparte de las propiedades anteriores, las entidades *Altavoz_1*, *Lab403* y *A1*.

Finalmente, se proveen dos espacios de nombres complementarios a los anteriores:

- **Rutas a partir del tipo de la entidad.** Las entidades se pueden referenciar a partir de su tipo. El primer identificador de la ruta señala uno de los posibles conceptos, e indica cuál debe ser el tipo de la primera entidad de la ruta. Así, la ruta */person/*/props/correoe* apunta a los correos electrónicos de todas las instancias de las persona que se conozcan.

- **Rutas a partir de jerarquías predefinidas.** La ruta anterior permite referenciar a todas las entidades de un tipo dado. Sin embargo, no se tienen en cuentan las instancias pertenecientes a un tipo que herede del tipo contemplado. Por ejemplo, la ruta */lampara/** no incluye a las instancias del concepto *LamparaRegulable*, el cual hereda de *Lampara*. Durante el desarrollo de distintas aplicaciones (véase el capítulo 7) se observó que surgían un conjunto de consultas que se repetían frecuentemente, y que el espacio de nombres no permitía manejar adecuadamente. Éstas incluían los conceptos del contexto primario, y se producían cuando se precisaba conocer los recursos o las personas que se encuentran localizados en un lugar dado. Para facilitar el desarrollo de esta operaciones se introdujeron los espacios de nombres a partir de jerarquías predefinidas. Estos restringen la secuencia de los tipos de la entidades que van a aparecer en la ruta. Cada jerarquía predefinida especifica una secuencia de tipos distinta. Por ejemplo, la jerarquía *lugarpersona* define que la primera entidad de la ruta será de tipo Lugar y la segunda de tipo Persona. De esta forma, cuando se emplea un comodín, solo se referenciarán aquellos nodos cuyo tipo coincida con el especificado por la jerarquía. Aparte de la jerarquía anterior, se ha incluido la jerarquía *lugarrecursos*. Continuando con el ejemplo de la figura 4.10, la ruta */lugarrecurso/lab403/*/props/status* se interpretaría como sigue: el primer identificador determina la jerarquía que se está empleando, la cual establece que la primera entidad tiene que ser de tipo Lugar y que le tiene que seguir una entidad de tipo Recurso. Así, la ruta se leería como el valor de la propiedad *status* de todos los recursos relacionados con la habitación *Lab403*.

Capítulo 5

Una arquitectura de pizarra para la gestión de información contextual

5.1. Introducción

En el capítulo 4 se ha propuesto un modelo contextual básico. Este modelo funciona como punto de entendimiento de las aplicaciones sensibles al contexto que operan dentro de un entorno inteligente. En este capítulo se van a describir los mecanismos que permiten beneficiarse del anterior modelo. Este conjunto de mecanismos van a formar una capa intermedia[1] entre el entorno y las aplicaciones que se va denominar **capa de contexto**.

En el apartado 3.3.1 se describieron diversos ejemplos de capas intermedias que se utilizan actualmente. Una de las conclusiones que se extrajeron (véase el apartado 3.4) es la conveniencia de las arquitecturas pizarra para modelos de programación orientados a datos. En la misma dirección apunta el enfoque presentado en el capítulo 4, donde se propone una representación del contexto basada en las entidades y relaciones que conforman el entorno. En este capítulo se plantea como mecanismo de comunicación y coordinación una capa intermedia basada en la metáfora de pizarra. A diferencia de otras arquitecturas de pizarra basadas en tuplas, nuestra propuesta se basa en una representación en forma de grafo similar a las redes semánticas. Las operaciones principales de la capa de contexto permiten recuperar la información contextual, notificar cambios en el contexto, descubrir nuevas fuentes de información contextual y añadirlas al modelo, o eliminar del modelo información contextual que ha dejado de utilizarse.

Primeramente se va a explicar en qué consiste la metáfora que plantean las arquitecturas de tipo pizarra. A continuación se van a abordar las ventajas y desventajas que suponen este tipo de arquitecturas, lo que va a permitir motivar en el siguiente apartado la elección de esta arquitectura como componente principal de la capa de contexto. Una vez descritas las características principales que ofrece la capa intermedia propuesta, se va a terminar el capítulo describiendo la

[1] middleware

implementación de cada una ellas.

5.2. ¿Qué se entiende por la metáfora de pizarra?

Las arquitecturas de tipo pizarra constituyen un paradigma clásico [93] que ha demostrado su utilidad para implementar sistemas de control [135]. Esta metáfora fue concebida dentro del campo de la Inteligencia Artificial, de ahí que los primeros trabajos con arquitecturas de pizarra se aplicaron a la resolución de problemas no determinísticos [95]. La solución del problema se obtenía de forma cooperativa por varios módulos, cada uno especializado en una tarea concreta. Éstos iban dejando los resultados parciales en la pizarra, y un coordinador central se encargaba de elegir, rechazar o ensamblar las distintas soluciones parciales. Cada módulo sólo conoce de la existencia de la pizarra, y no necesita ninguna información del resto del sistema. De esta forma se adopta un punto de vista centrado en los datos en vez de centrado en el proceso. Los componentes del sistema, en vez de comunicarse directamente, envían sus peticiones a un repositorio central. Para recibir la información los componentes pueden, o bien suscribirse a los cambios en la pizarra, o bien acceder a la pizarra.

Las arquitecturas de pizarra se pueden aplicar a cualquier tipo de dominio, y han sido empleadas con éxito en otros proyectos en diversos campos de la *Computación Ubicua* como en los *entornos inteligentes* [103], [106], los agentes *software* [171], vestibles [82], o aplicaciones sensibles al contexto [122].

Actualmente, la tecnologías que se emplean para implementar las arquitecturas de tipo pizarra se basan en mecanismos de memoria compartida y distribuida basados en tuplas similares al sistema *Linda* [110]. En estos sistemas la información básica que se intercambia y que se guarda en la pizarra son tuplas. Una implementación de una pizarra basada en tuplas son los *T-Spaces* [297] de la compañía IBM. También se pueden encontrar implementaciones de pizarras basadas en objetos como *JavaSpace* [153] de Sun.

5.3. Criterios a considerar en la elección de la metáfora de pizarra

Según Terry Winograd [292] existen cuatro criterios claves para medir la bondad de una arquitectura distribuida: eficiencia, configurabilidad, robustez y simplicidad.

Eficiencia

En general, la eficiencia de cualquier sistema de computación viene dada por dos variables: el tiempo y el espacio. El tiempo que requiere para ejecutarse un proceso, el espacio que ocupa determinada representación de datos, o el tiempo que se tarda enviar determinada cantidad de información son medidas habituales para

comparar el rendimiento de un sistema. La principal crítica que se ha realizado a las arquitecturas de tipo pizarra es que no son eficientes en cuanto al tiempo. Estas arquitecturas se basan en un componente central del cual dependen todas las comunicaciones del sistema. Esto hace que puedan sufrir el efecto que se conoce como cuello de botella[2], que consiste en que el sistema ralentiza su rendimiento —pudiéndose llegar a saturarse— debido a que todas las comunicaciones tienen que pasar por un único componente. Otro problema de ineficiencia que presentan las pizarras es que toda comunicación requiere como mínimo dos accesos, uno del agente a la pizarra y otro de la pizarra al agente.

Teniendo en cuenta las características del problema, es necesario preguntarse si la tecnología disponible permite afrontar estas desventajas empleando una pizarra. En el apartado 4.5.2 se discutió la naturaleza de la información que se trata como contexto. En este sentido, se impusieron ciertas restricciones (por ejemplo, no se contempla información síncrona o con elevada velocidad de transmisión) que reducen la tasa de cambios en la pizarra. En algunos casos esto implicará que los datos que pasen por la pizarra tendrán que ser previamente filtrados. No tendría sentido que un sensor publicara en la pizarra un nuevo dato cada milisegundo, pero tampoco el tipo de escenarios contenplados requiere una resolución tan elevada. Por otro lado, el número de componentes que se espera encontrar en un entorno inteligente no suele ser muy alto. Revisando los ejemplos de que se describen en el capítulo 7, el número de entidades se encuentra en torno a la centena. A esto hay que añadir que las aplicaciones sensibles al contexto requieren tiempos de respuesta en la escala de la percepción humana, que es un requisito relativamente suave dentro de los sistemas de tiempo real.

Finalmente, otro de los problemas de eficiencia de las arquitecturas pizarra radica en la escalabilidad. Los sistemas basados en un espacio de datos compartido resultan difícilmente escalables, ya que se requiere de un mecanismo que permita replicar la información en todos los servidores de la arquitectura. Aunque la implantación de estas arquitecturas está limitada para redes de área global, se pueden adoptar soluciones que permitan mejorar la escalabilidad. Éstas pasan por mantener un modelo lógico centralizado, pero realizar una implementación distribuida. Las aplicaciones siguen percibiendo que el acceso a los datos se realiza como si hubiera un único componente; sin embargo, éstos se encuentran distribuidos en distintas pizarras organizadas jerárquicamente. Otra medida complementaria consiste en adoptar un mecanismo que mantenga los datos más usados en los servidores más cercanos.

Configurabilidad

Una de las ventajas de las pizarras es la flexibilidad que aportan para configurar el sistema, debido a que se basan en un mecanismo asíncrono de comunicación que permite un acoplamiento débil entre las fuentes y los consumidores de información. Este desacoplamiento se produce en tres niveles:

[2]bottleneck

- **Temporal.** Los dos participantes de la información no tienen por qué estar activos simultáneamente. El emisor puede dejar la información en la pizarra y terminar su ejecución. Posteriormente, el receptor se ejecuta y recupera la información.

- **Espacial.** La fuente no conoce quiénes son los consumidores. La fuente publica la información en la pizarra, y ésta se encarga de retransmitirla a los subscriptores sin que la fuente intervenga.

- **Funcional.** Los productores de la información no precisan conocer cuál es el uso que se le va dar a la información.

Este anonimato permite que la conexión o desconexión de un componente en el entorno se realice de forma transparente al resto de los componentes. Por tanto este tipo de comunicación facilita los cambios en la configuración del entorno, y es muy adecuada cuando estos cambios son frecuentes.

Las arquitecturas de pizarra se encuadran dentro de las capas intermedias orientadas a infraestructura (véase el apartado 3.3.1). Éstas se basan en un servidor central y público que es accedido por múltiples clientes ligeros. A este servidor se le asocia una dirección que no suele cambiar a lo largo del tiempo, lo que desde el punto de configuración supone una gran ventaja, ya que los clientes únicamente necesitan conocer dicha dirección.

Por otro lado, la implementación de un sistema basado en pizarra se realiza mediante una arquitectura orientada a datos (véase el apartado 3.3.1). Ésta consiste en una interfaz compuesta por un conjunto reducido de operaciones que permiten acceder a la información que se almacena en la pizarra. Se produce una separación clara entre los datos y la funcionalidad de la arquitectura, lo cual permite que ambas partes evolucionen por separado. Una ventaja directa de este enfoque reside en que cambiar la funcionalidad de los clientes es sencillo. Además, las tareas de búsqueda, compartición y comunicación de información se facilitan [118]. En contraposición, las arquitecturas orientadas a servicios ofrecen un extenso conjunto de servicios que los clientes deben conocer a priori o descubrir en tiempo de ejecución. En el primero de los casos, los clientes se encuentran fuertemente acoplados a los servicios de manera que la evolución del sistema se ralentiza. En el segundo, la configuración se complica al requerir de nuevos componentes que permitan localizar los servicios y componer nuevos servicios a partir de los existentes.

Robustez

La robustez de una arquitectura se mide en dos ejes: la tolerancia a fallos del sistema y la capacidad para recuperarse de éstos. Las pizarras son un posible punto de fallo. Esto supone una desventaja ya que el fallo de un único componente tiene como consecuencia la parada de todo el sistema. Por otro lado constituye una ventaja, ya que todos los esfuerzos por mejorar la robustez del sistema se centran en un único punto manteniéndose constante la complejidad de los clientes. Para mejorar la robustez de una arquitectura centralizada existen diversas soluciones que aportan una alta disponibilidad. La primera consiste en duplicar la

infraestructura del servidor empleando un *cluster* de servidores, de forma que si uno deja de funcionar, otro asume la funcionalidad. Esta solución da buenos resultados incluso con *clusteres* de dos equipos. La segunda solución sería distribuir la implementación en varios nodos manteniendo un vista lógica de pizarra de forma centralizada. Mientras que en la primera solución se dispone de un *hardware* que facilita la sincronización de los nodos, en la segunda solución se confía en una capa intermedia. Esta solución es más económica al no requerir de componentes *hardware* especializados.

Simplicidad

La simplicidad suele ser una de las claves para que una solución sea adoptada. Así lo demuestra la gran cantidad de protocolos o sistemas cuyos nombres contienen la palabra *Simple* (SNMP[3], SMTP [4], SOAP[5], SMRP[6], SAX[7] ...), y tantos otros —como HTTP— que sin exponerlo explícitamente en su título, son referentes de diseños simples y ampliamente difundidos. En este sentido, las arquitecturas de tipo pizarra proveen de un mecanismo sencillo de coordinación, ya que un cliente puede intercambiar información con cualquier otro cliente con sólo conocer la ubicación de la pizarra, en contraste con un sistema totalmente descentralizado en el que el grado de complejidad en la negociación puede ser de $O(n^2)$ donde n es el número de elementos.

La extensión de la arquitectura se facilita gracias a que es sencillo para el desarrollador buscar la información disponible ya que esta se encuentra en un único punto. Además como ya se ha comentado, la configuración de los nuevos componentes se simplifica ya que el acoplamiento entre ellos es débil.

5.4. Capa de contexto

Para uniformizar la interacción de los diferentes componentes del entorno se ha diseñado una capa intermedia que se ha denominado capa de contexto. Esta capa sirve de nexo de unión entre los distintos componentes que conforman el entorno. Para ello se ha elegido una arquitectura de tipo pizarra.

Las características más destacadas de la arquitectura de pizarra propuesta son:

- **Modelo de datos común.** La información que se almacena en la pizarra sigue un modelo común basado en un grafo. Los diferentes componentes del entorno se representan mediante entidades definidas por un conjunto de propiedades y un conjunto de relaciones con otras entidades. Este modelo ha sido descrito en el capítulo 4.
- **Un repositorio central.** La pizarra almacena toda la información contextual que se genera en el entorno. La información que se guarda puede ser

[3]Simple Network Management Protocol
[4]Simple Mail Transport Protocol
[5]Simple Object Access Protocol
[6]Simple Multicast Routing Protocol
[7]Simple API for XML

un cambio en una propiedad (por ejemplo, una puerta que se abre), o una nueva entidad que se añade o se elimina de la pizarra (por ejemplo, alguien que entra o sale de la habitación). Este repositorio funciona tanto para los componentes del mundo físico como para los del mundo virtual.

- **Un mecanismo de comunicación.** Éste se basa en dos tipos de protocolos: por sondeo, y por eventos asíncronos. El primero consiste en consultar la pizarra directamente para obtener la información que se requiere. El segundo consiste en un mecanismo de publicación-subscripción, de forma que las fuentes de información publican en la pizarra los cambios en el contexto, mientras que los consumidores se subscriben a esos cambios para recibirlos.

La flexibilidad y la simplicidad de las arquitecturas de pizarra las hacen una solución muy adecuada para entornos donde la configuración varía frecuentemente. Tal es el caso de los escenarios de la *Computación Ubicua* y los *entornos inteligentes*. Además, a esto hay que unir que el modelo de contexto explicado en el capítulo 4 requiere una estructura global de información compartida por todas la aplicaciones.

La capa de contexto se basa en una estructura global de información que se gestiona utilizando la metáfora de pizarra. A diferencia de otras arquitecturas de tipo pizarra donde la información se almacena en forma de tuplas, la capa de contexto se basa en un grafo dirigido donde los nodos consisten en entidades que representan el contexto del entorno y los vértices representan relaciones entre las entidades (véase el capítulo 4). Este grafo se encuentra a disposición de todas las aplicaciones.

La interacción con la capa de contexto se basa en un modelo cliente-servidor. La capa de contexto ofrece una serie de operaciones que permiten acceder al modelo del mundo (véase el apartado 5.5). Estas operaciones permiten comunicar cambios en el contexto, encontrar recursos disponibles, revelar si se añade o se elimina parte del contexto del entorno, o si las relaciones entre las entidades varían.

Además del repositorio central, dentro de un *entorno activo* se pueden encontrar diversos tipos de componentes. Cada uno de ellos emplea la pizarra para interaccionar con el resto. Estos componentes pueden estar muy cercanos al mundo físico, tales como sensores, dispositivos, interruptores, pantallas, micrófonos, etc., o pueden ser cualquier tipo de componente *software*, tales como gestores de diálogos, agentes inteligentes, interfaces de usuario, etc. Todos ellos utilizarán la capa de contexto como mecanismo de intercambio de información.

Una primera clasificación de los componentes se establece entre aquellos que pertenecen al mundo físico y aquellos que se encuentran en el mundo virtual. La pizarra almacena una representación de ambos mundos, de modo que los cambios en una y otra parte quedan reflejados en ella, y están disponibles a todo aquel que lo precise. Así se facilita el desarrollo de aplicaciones, ya que la comunicación con cualquier parte del modelo se realiza de forma transparente.

Los componentes, tanto del mundo físico como del virtual, se pueden clasificar en cinco grandes grupos:

- **Sensores.** Se denominan así a las fuentes de información contextual que

pertenecen al mundo físico. Los sensores miden el contexto directamente del mundo real, aportando un información contextual de alta resolución pero con poco grado de abstracción.

- **Intérpretes.** Están suscritos a la información que se publica en la pizarra. Existen de dos tipos distintos: por un lado, aquellos que procesan la información cruda de los sensores para generar información con un mayor grado de abstracción, deduciendo nuevas entidades y relaciones que se añaden a la pizarra. Por otro lado, están aquellos que interpretan los cambios que generan los productores, propagando nuevas acciones que pueden llegar a afectar a los actuadores del entorno.

- **Consumidores.** Son los receptores finales de la información contextual. Dentro de este grupo se incluyen las aplicaciones sensibles al contexto. Éstas acceden a la información contextual a través de los cambios en la pizarra y adaptan su comportamiento consecuentemente.

- **Productores.** Este grupo engloba a aquellas aplicaciones, módulos, agentes, etc. del mundo virtual que producen modificaciones en la información de la pizarra. Los nuevos cambios comunicados por los productores se pueden interpretar como una modificación del contexto, por ejemplo, una notificación de un nuevo mensaje, o como una acción a realizar sobre el mundo físico, por ejemplo, un cambio en la temperatura del entorno.

- **Actuadores.** Estos componentes se encargan de que los cambios en el modelo almacenado en la pizarra se propaguen al mundo físico. Al igual que los sensores, el grupo de los actuadores lo forman dispositivos físicos que —en general— son capaces de realizar acciones sencillas. Esto conlleva que comandos más complejos requieran ser interpretados para que tengan efecto en el mundo físico.

La figura 5.1 resume la interacción entre estos cinco grupos de componentes. Tal como se observa en la figura, la pizarra constituye un punto de central de entendimiento entre el resto de los componentes. Se ha dotado a la pizarra de controladores específicos que permiten la comunicación con los dispositivos físicos (véase el apartado 5.7), así como de una interfaz común de programación —que emplea protocolos estándar— para todas las aplicaciones del mundo virtual (véase el apartado 5.6.3). De esta forma, la pizarra habilita la interacción entre ambos mundos.

Volviendo a la figura 5.1, las flechas que llegan a la pizarra significan que los componentes realizan cambios en el modelo de la pizarra. Las flechas que salen de la pizarra se interpretan en dos sentidos: como notificaciones de las modificaciones en alguna parte del modelo, fruto de una subscripción previa, o como consultas directas del componente a alguna parte del modelo.

La pizarra guarda una fotografía de la información contextual del entorno que un momento dado se conoce. Esta fotografía se representa en forma de un grafo de entidades, propiedades y relaciones tal como se describió en el apartado 4.5.1.

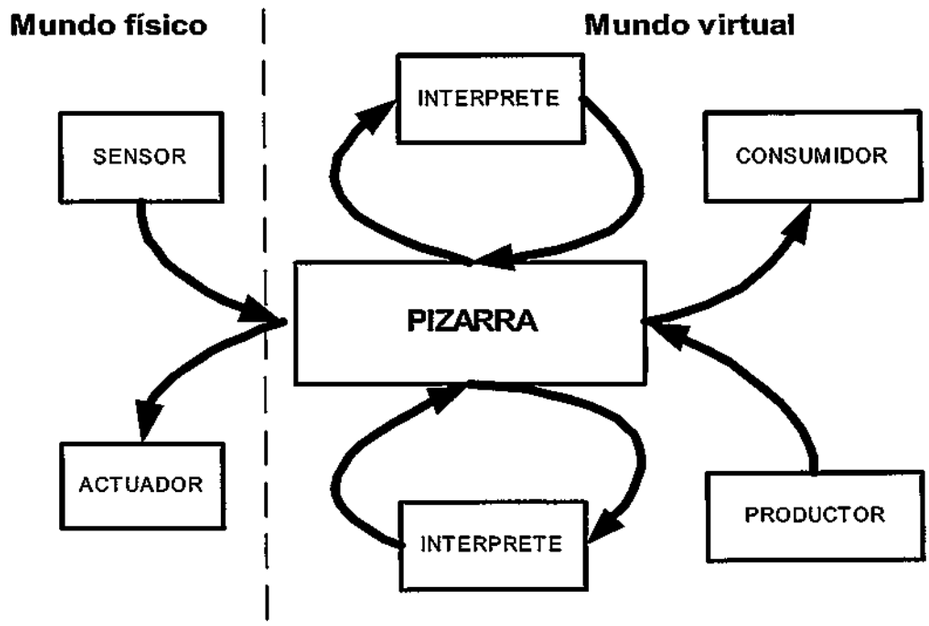


Figura 5.1: Esquema de interacción entre los diferentes tipos de componentes.

Cada entidad representa una instancia de alguno de los conceptos definidos en el contexto primario, o en alguno de los esquemas del contexto secundario. Esta fotografía del contexto se compone como si de un puzzle se tratara, donde las piezas se identifican con las entidades y sus propiedades, y las relaciones ligan las piezas del puzzle entre sí. Los distintos componentes actúan sobre las piezas del puzzle. Existen dos formas interaccionar con la fotografía: la primera consiste en cambiarla. Para ello los componentes pueden añadir nuevas piezas, ampliando la fotografía; pueden eliminar piezas de la fotografía, o pueden alterar el dibujo de las piezas. Todos estos cambios se rigen por el modelo de datos que determina dónde se pueden colocar las nuevas piezas, y cómo se pueden modificar. La segunda forma que tienen los componentes del entorno de interaccionar con el puzzle consiste en observar cómo se transforma. Se proporcionan dos opciones: o bien se registran a los cambios en las piezas de la fotografía, de forma que cuando se produzca uno les avisen; o bien consultan la fotografía por propia iniciativa.

Tal como se ha explicado anteriormente, una de las ventajas del paradigma propuesto radica en que cada componente no tiene por qué tener conocimiento de la existencia del resto. Cada componente únicamente conoce la ubicación de la pizarra, y qué parte del modelo le interesa, ya sea para cambiarla o para consultarla. Este enfoque conlleva un acoplamiento débil en tres sentidos: temporal, espacial y funcional. Por un lado, no se precisa que los componentes se encuentren sincronizados, esto es, un componente puede realizar un cambio en el modelo, y terminar su ejecución. Posteriormente, otro componente puede consultar a la pizarra y percatarse del cambio. Por otro lado, cuando un componente realiza una

modificación de la fotografía no tiene constancia de a quién afecta ese cambio. Cada componente interacciona con la pizarra como si estuviera sólo en el mundo, lo cual facilita su implementación.

El paradigma propuesto permite una gran flexibilidad a la hora de diseñar las aplicaciones que van a formar parte del entorno. Se pueden añadir y eliminar aplicaciones dinámicamente sin que afecte directamente a la configuración del resto. Estos beneficios se obtienen gracias a que el modelo de la información contextual constituye un contrato entre todos los componentes. Esto conlleva que el desarrollo de nuevos componentes, tanto en la capa física como en la capa de aplicación, sea un proceso que se realice de forma independiente. Cada nuevo componente se fija en una parte de la información, sin preocuparse de quién se encarga de obtenerla.

Ahora bien, un problema a tener en consideración surge cuando los componentes quieren cambiar simultáneamente la misma parte del modelo, o cuando un componente quiere bloquear una parte del modelo para que nadie la modifique durante un cierto tiempo. Al no conocer la existencia del resto de los componentes se precisa que la pizarra prevea mecanismos que permitan coordinar correctamente el acceso a las piezas del modelo. Esta parte se estudiará en los apartados 5.9 y 5.10.

5.5. Mecanismo de comunicación del contexto

En este apartado se explicará en detalle cómo se realiza la comunicación a través de la pizarra entre los clientes de la capa intermedia. La comunicación entre los productores y los consumidores se basa en una arquitectura de tres capas: capa física, capa de contexto y capa de aplicación. La capa física engloba a los componentes que proporcionan el valor de propiedades medidas directamente del mundo físico, mientras que la capa de aplicación alberga los componentes que deducen nuevas propiedades del mundo físico o que proporcionan valores relacionados con el mundo virtual. Tal como se ha explicado anteriormente, la información contextual generada, tanto en la capa física como en la de aplicación, se publica en un repositorio central accesible al resto del sistema.

El mecanismo de comunicación se resume en cuatro tipos de operaciones: consulta y modificación de propiedades, consulta y modificación de entidades, consulta y modificación de relaciones, y subscripción a cambios. La combinación de estas cuatro operaciones permite obtener la interacción requerida. Un escenario de ejemplo sería: alguien entra en una habitación vacía. Esto provoca que el agente de presencia notifique este evento añadiendo una relación entre la habitación y la persona. A continuación la pizarra avisa a una aplicación sensible al contexto que está subscrita a las modificaciones en la relación de inclusión entre la habitación y cualquier persona. Una vez recibido el cambio, la aplicación comprueba directamente el valor una propiedad que indica la luminosidad actual del entorno. Como recibe que está demasiado oscuro, el agente cambia la propiedad que aumenta la intensidad de las luces de la sala. Tanto la consulta del estado de la luces como el cambio en el valor de las mismas se transmite desde la pizarra hacia el

mundo físico a través de una pasarela SNMP (véase el apartado 5.7). Cuando la persona abandona la habitación, se apagan las luces automáticamente siguiendo un proceso de interacción similar.

Consulta y modificación de propiedades

La pizarra provee de un conjunto de funciones estándar (véase el apartado 5.6.3) que permiten obtener y modificar el valor de las propiedades. Por ejemplo, una de las propiedades del contexto es la cantidad de personas que hay en una habitación. Esta información se genera a partir de las lecturas de varios sensores, las cuales se almacenan en la pizarra. Un interprete procesa los valores leídos y obtiene un único valor, con el que modifica el valor de la propiedad *número de personas*. Finalmente, un consumidor subscrito a esa propiedad recibe y utiliza ese valor final sin ser consciente del número y naturaleza de las fuentes de contexto.

El proceso de interacción se puede resumir tal como sigue: los productores de contexto envían los cambios del contexto a la pizarra, en consecuencia la pizarra cambia el valor de los nodos correspondientes. Los consumidores de contexto (o los interpretes) se percatan de los cambios, o bien porque preguntan directamente a la pizarra, o bien porque se encuentran suscritos a los cambios.

Las propiedades que corresponden a valores que se miden directamente de un dispositivo físico reciben un tratamiento especial. En estos casos, el valor de la propiedad no se guarda en la pizarra, en cambio ésta actúa simplemente de intermediario. Siempre que se requiere el valor, la pizarra lo solicita directamente al dispositivo.

La figura 5.2 muestra cómo se realiza el proceso completo de interacción. Las líneas discontinuas representan notificaciones de cambios en el contexto, y las líneas sólidas peticiones de agentes para preguntar sobre el estado de un nodo, o para cambiarlo. Los nodos oscuros representan propiedades que no se almacenan en la pizarra directamente.

Consulta y modificación de relaciones

La pizarra posee un mecanismo que permite añadir y borrar relaciones entre entidades. Las relaciones se establecen entre dos entidades, que se denominan entidad fuente y entidad destino y son siempre unidireccionales. Así, si es necesario representar una relación bidireccional habrá que añadir dos relaciones a la pizarra. Cada relación se especifica mediante la entidad origen, la entidad destino, y el nombre de la relación. Igual que ocurre con las propiedades, cada vez que se produce una modificación en las relaciones existentes se notifica a los clientes suscritos.

Las relaciones son el mecanismo que permite establecer estructura entre las entidades reflejando en cada momento las conexiones que existen entre los diferentes elementos del entorno. Un ejemplo puede ser la relación *contiene* que refleja la relación espacial de inclusión entre un entorno y un objeto. Cuando se detecta que una persona entra en un entorno, se añade al modelo de la pizarra una relación de tipo *contiene* entre la entidad que representa al entorno y la entidad

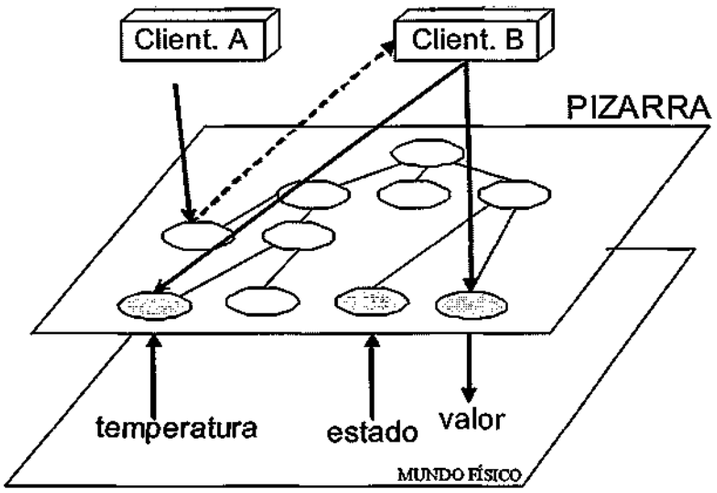


Figura 5.2: Esquema de interacción de dos componentes empleando la pizarra.

que representa a la persona. De la misma manera, cuando la persona abandona el entorno, la relación anterior desaparece.

Consulta y modificación de entidades

Al igual que las relaciones, las entidades se pueden consultar, añadir y borrar según entren a formar parte o desaparezcan de la información contextual. Cuando se añade un nueva entidad, se añade su representación a la pizarra, que incluye el nombre, el tipo y las propiedades que tuviera, así como cualquier información propietaria de las aplicaciones que estuvieran instaladas (véase el apartado 6.2.3). Si se borra una entidad, se elimina toda la información relativa a ésta, tanto las propiedades como las relaciones que tuviera. En el ejemplo del comienzo del apartado se explicaba cómo la entrada y salida del entorno conlleva la aparición y desaparición de la pizarra de la entidad que representa a la persona.

Subscripción a cambios

Finalmente, la comunicación del contexto se completa con un mecanismo de eventos asíncronos que permite suscribirse a los cambios que se produzcan. Éstos pueden ser debidos a modificaciones en una propiedad, en una relación o en una entidad. Las suscripciones plantean mayor complejidad al desarrollador ya que requiere implementar aplicaciones que estén constantemente activas. Por el contrario, en la comunicación directa es la aplicación quien decide cuándo preguntar a la pizarra, lo cual supone una ventaja cuando las consultas surgen esporádicamente. En cambio, cuando se quiere conocer en todo momento el valor de una parte del modelo, es preferible emplear el mecanismo de subscripción, en vez de consultar periódicamente a la pizarra.

5.6. Arquitectura de la capa de contexto

Cada pizarra es un servidor que implementa el conjunto de servicios definidos por la capa de contexto (véase el apartado 5.6.3). La comunicación con la pizarra sigue una arquitectura clásica de cliente-servidor. Dependiendo de si el cliente pertenece a la capa física o a la capa de aplicación se prevén dos formas distintas de acceso a los servicios ofrecidos por la pizarra. Se ha definido un mecanismo genérico de acceso para los clientes de la capa de aplicación, ya que se presupone que éstos tienen más recursos, mientras que los dispositivos de la capa física pueden tener capacidades de computación limitadas. Para estos segundos se prevé en la pizarra un mecanismo para poder añadir controladores específicos para cada dispositivo, que se comunican de manera *ad-hoc*. En la primera parte de este apartado se van describir los protocolos y la interfaz para la capa de aplicación, en la segunda parte se explicará un ejemplo de cómo la pizarra integra un controlador de dispositivos.

La comunicación entre la pizarra y los clientes de la capa de aplicación se realiza empleando el protocolo HTTP [8] basado en TCP/IP. Es un protocolo que surgió para intercambiar páginas de hipertexto, pero que dada su sencillez de implementación se ha extendido a otros dominios de aplicación. Los comandos enviados a la pizarra se codifican empleando la sintaxis definida por el estándar XML. Tanto HTTP como XML permiten usar herramientas estándar que facilitan la integración en la pizarra de nuevos desarrollos, así como la documentación y mantenimiento del sistema.

5.6.1. Implementación de la pizarra

La implementación de la pizarra sigue el esquema que se muestra en la figura 5.3.

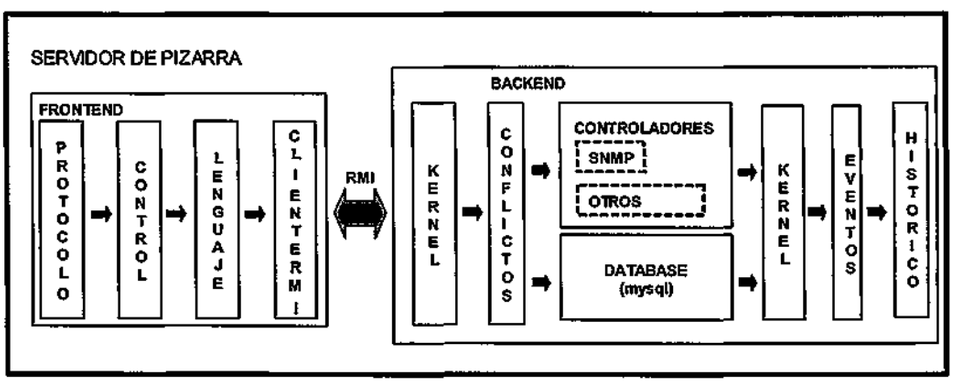


Figura 5.3: Esquema de la arquitectura de la pizarra. Módulos principales.

La pizarra tiene dos componentes principales:

- **Lógica de aplicación** o *backend*. Consta de los siguientes módulos.

[8] HyperText Transfer Protocol

Núcleo. Este módulo implementa los servicios de la capa de contexto: *Obtener Contexto, Establecer Contexto, Añadir Contexto, Eliminar Contexto, Añadir Relación, Borrar Relación, Subscribir* y *Desubscribir.*

Base de Datos. Permite almacenar de forma persistente las entidades, las propiedades, los parámetros, y las relaciones entre entidades que conforman el contexto. El núcleo se comunica con la base de datos mediante la interfaz *ODBC.* Se ha utilizado el motor de bases de datos MySQL. La arquitectura permite cambiar de manera sencilla el mecanismo de almacenamiento, bien por otro gestor de bases de datos, bien por un sistema distribuido de almacenamiento, según fuera necesario.

Módulos extras. Implementan funcionalidad adicional que puede ser integrada sin modificar el resto de la arquitectura. Es un mecanismo que prevé la extensibilidad del sistema. Por el momento se han desarrollado dos módulos: un mecanismo de memoria contextual (véase el apartado 5.8) y un mecanismo para solucionar conflictos (véase el apartado 5.10).

Controladores. Éstos se implementan a medida para poder comunicar a la pizarra con los diferentes dispositivos del entorno. En el caso del entorno inteligente *Interact* se dispone de un controlador SNMP[9], ya que la mayor parte de los dispositivos se comunican mediante este protocolo. Ésta es la capa que aísla a la pizarra de las particularidades del *hardware.*

- **Interfaz de comunicación** o *Front-end*: esta parte implementa la conexión y comunicación con el cliente. La comunicación entre el *front-end* y el *back-end* se realiza empleando RMI[10] de forma que se pueden añadir varios *front-ends* a la arquitectura ejecutándose en paralelo, cada uno con un protocolo y un lenguaje de representación distintos. En la actualidad se soporta HTTP, como capa de transporte y XML, como lenguaje de intercambio de información. Para la implementación del servidor HTTP se emplea *Tomcat Apache.*

5.6.2. Parámetros de configuración

Tal como se explicó en el apartado 4.5.2, el modelo de entidades y propiedades se puede extender para incluir información propietaria de cada aplicación. Estos elementos de información se denominan parámetros, y se agrupan en conjuntos de parámetros. La pizarra incluye un conjunto de parámetros denominado *default* asociado a cada entidad y a cada propiedad que permite personalizar la configuración de la pizarra.

En el caso de las entidades se define por defecto un único parámetro:

[9]Simple Network Management Protocol
[10]Remote Method Invocation

- **Descripción.** Este parámetro almacena una descripción textual de la entidad. Se emplea por motivos de documentación y su inclusión es opcional. También está definido este parámetro en cada propiedad.

```
<entity name="Lamp_1" type="device">

  <property name="Status">
   <paramSet name="default">
    <param name="location">1</param>
    <param name="access">0</param>
   </paramType>
  </property>

  <property name="Value">
   <paramSet name="default">
    <param name="location">1</param>
    <param name="initialValue">0</param>
    <param name="access">1</param>
   </paramType>
  </property>

  <paramSet name="default">
   <param name="desc">Luces del salón</param>
  </paramType>
</entity>
```

Figura 5.4: Ejemplo de parámetros de configuración de la pizarra

Para las propiedades, aparte del parámetro *Descripción* se definen:

- **Localización.** Distingue si la propiedad es interna o externa. En el primer caso el valor de la propiedad se almacena en la pizarra mientras que en el segundo los cambios o lecturas del valor se redireccionan al dispositivo directamente (véase el apartado 5.5). Este parámetro es opcional, si no se establece se toma como defecto que la propiedad es interna.
- **Valor inicial.** Permite establecer un valor inicial la primera vez que se añade la propiedad a la pizarra.
- **Acceso.** Establece el tipo de acceso a la propiedad: lectura (0), escritura (1) o lectura y escritura (2).

En el ejemplo 5.4 se define un dispositivo de tipo *Luz* que tiene una propiedad denominada *Status* que es de sólo lectura e indica el estado de la lámpara y una propiedad *Value*, de sólo escritura que permite cambiar el estado de la lampara, y cuyo valor inicial es apagado (0 - apagado, 1 - encendido).

```
ruta_pizarra    = hostport [ '/' hpath ]
hostport        = host [ ':' port ]
host            = hostname | hostnumber
hostname        = *[ domainlabel '.' ] toplabel
domainlabel     = alphadigit | alphadigit *[ alphadigit | '-' ] alphadigit
toplabel        = alpha | alpha *[ alphadigit | '-' ] alphadigit
alphadigit      = alpha | digit
hostnumber      = digits '.' digits '.' digits '.' digits
port            = digits

hpath           = hsegment *[ '/' hsegment ]
hsegment        = *[ uchar | ';' | ':' | '@' | '&' | '=' ]

alpha           = lowalpha | hialpha
digit           = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' |
                  '8' | '9'
safe            = '$' | '-' | '_' | '.' | '+'
extra           = '!' | '*' | ''' | '(' | ')' | ','
national        = '' | '' | '|' | '\' | '^' | '~' | '[' | ']' | '`'
punctuation     = '<' | '>' | '#' | '%' | <'>

reserved        = ';' | '/' | '?' | ':' | '@' | '&' | '='
hex             = digit | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' |
                  'a' | 'b' | 'c' | 'd' | 'e' | 'f'
escape          = '%' hex hex

unreserved      = alpha | digit | safe | extra
uchar           = unreserved | escape
xchar           = unreserved | reserved | escape
digits          = 1*digit

lowalpha        = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' |
                  'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' |
                  'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' |
                  'y' | 'z'
hialpha         = 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' |
                  'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' |
                  'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'
```

Figura 5.5: Símbolos terminales y no terminales extraídos del RFC1738

5.6.3. Operaciones básicas que soporta la pizarra

Este apartado detalla cuáles son las operaciones básicas que soporta la Interfaz de Programación de la pizarra. Para cada operación se describe el efecto que produce en la pizarra y el formato de los mensajes intercambios. Se ha empleado una variante de la notación BNF[11] definida en [35] para definir el formato de los mensajes. Los símbolos que se utilizan en esta notación se resumen como sigue: los corchetes cuadrados [ ] delimitan partes opcionales; los paréntesis ( ) permiten agrupar distintos símbolos; una barra vertical | indica que hay que elegir una de las posibles opciones que se presentan; un asterisco indica que el símbolo que viene a continuación se repite cero o más veces; finalmente, los textos literales se denotan mediante comillas simples ' ' para evitar confusión con las dobles comillas que se emplean en XML.

Se parte de algunas definiciones sintácticas ya conocidas. En primer lugar, de la definición de URL[12] que se recoge en la RFC1738 [35] y se muestra en la figura 5.5. Cada vez que se envía un mensaje a la pizarra mediante el protocolo HTTP, se necesita especificar el URL donde se encuentra localizada la pizarra. Así, cada vez que aparezca el símbolo *ruta_pizarra* habrá que referirse a la anterior figura. Además, también se reutilizan otros símbolos que se definen en el RFC1738, por ejemplo, *host*, que describe el nombre de una máquina tanto con su dirección IP como por su nombre DNS, incluyendo el dominio.

En segundo lugar, en la figura 5.6 se detallan los símbolos necesarios para definir una *ruta* de un nodo de la pizarra, donde el nodo puede referirse a una entidad o a una propiedad. Este símbolo, también, aparece con relativa frecuencia, por lo que se define aparte.

```
ruta              = raiz '/' ( resto | '*' )
raiz              = 'name' | tipo | jerarquia
tipo              = identificador
jerarquia         = 'roomresource' | 'roomperson'
resto             = ruta_entidad [ruta_propiedad]
ruta_entidad      = '/' ( identificador | '*' ) *[ relacion ]
relacion          = '/' identificador '/' ( identificador | '*' )
ruta_propiedad    = '/props/' ( identificador | '*' )
identificador     = alpha | alpha *[ alphadigit | '-' ] alphadigit
```

Figura 5.6: Representación en BNF de la ruta de un nodo de la pizarra

A continuación se muestra la sintaxis de las operaciones básicas: ObtenerContexto, CambiarContexto, Subscribirse, Desubscribirse, AñadirContexto, BorrarContexto, AñadirRelaciones y BorrarRelaciones.

- **Obtener Contexto** *[GetContext]*

[11] Backus-Naur Form
[12] Uniform Resource Locator

El cliente suministra la ruta (véase el apartado 4.8) de una entidad o una propiedad que se encuentre en la pizarra. La respuesta de la pizarra contendrá el nodo referenciado codificado en XML (véase el apartado 6.2.2). Dependiendo de si se accede a una entidad o a una propiedad se obtendrán distintos resultados. En el caso de la entidad, la pizarra devolverá el identificador y el nombre de la entidad, y en el caso de la propiedad además se añadirá su valor. Tal como se explicó (véase el apartado 5.5), éste puede estar almacenado en la pizarra, o puede ser obtenido de un dispositivo externo, pero esto es transparente al cliente que realiza la petición.

Formato de la petición HTTP:

```
peticion     = 'PUT' ruta_pizarra '/get HTTP/1.0\n\r' mensaje
mensaje      = '<GetRequest><path>' ruta '</path></GetRequest>'
```

Formato del mensaje de respuesta XML:

```
respuesta    = '<GetResponse>' nodo '</GetResponse>'
```

La representación del nodo se puede consultar en la figura 6.8.

Esta operación se puede simplificar codificándola entera en la URL de la petición HTTP. La ruta del nodo se concatena con la ruta de la operación y la operación queda completamente especificada. En este caso, no es necesario enviar ningún mensaje XML adicional, por lo que se emplea el método GET del protocolo HTTP.

Formato de la petición HTTP:

```
peticion = 'GET' ruta_pizarra '/get' ruta 'HTTP/1.0'
```

Seguidamente se muestran dos ejemplos de peticiones, una para una entidad y otra para una propiedad, y las respuestas que produce la pizarra.

Ejemplo de petición HTTP:

```
(Entidad)    GET /interact/bb/get/device/lamp_1 HTTP/1.0
(propiedad) GET /interact/bb/get/device/lamp_1/props/status HTTP/1.0
```

Ejemplo de mensaje de respuesta XML:

```
(Entidad)
<GetResponse>
    <entity name="lamp_1"/>
</GetResponse>

(Propiedad)
<GetResponse>
    <property name="status">1</property>
</GetResponse>
```

La operación **Obtener Contexto** admite el uso del carácter comodín * en la ruta del nodo, de tal forma que se puede recuperar la información de más de un nodo a la vez. La ruta se lee de izquierda a derecha: cada nodo que se encuentre se añade al árbol de respuesta. Cuando aparece un comodín se incluyen todos los nodos referenciados, y para cada uno de los nodos se aplica la parte restante de la ruta. El pseudocódigo de la operación es el siguiente:

Pseudocódigo:

```
nodos = recuperar_nodo_inicial(ruta)
MIENTRAS QUE exista_token(ruta) HACER
token = siguiente_token(ruta)
nodos_aux = nodos.extraerTodos()
PARA CADA nodo_actual EN nodos_aux HACER
    SI token = "*" ENTONCES
    nodos.añadir(recuperar_nodos(nodo_actual))
    SINO
    nodos.añadir(recuperar_nodo(token,nodo_actual))
    FIN SI
    resp.añadir(nodos)
FIN PARA CADA
FIN MIENTRAS
```

La variable *nodos* se inicializa al primer nodo de la ruta. Se extraen todos los nodos de *nodos* (en la primera iteración sólo contendrá el inicial), y para cada nodo se extrae el nodo que indique en el token que corresponda de la ruta. En el caso de que se encuentre un asterisco, se extraerán todos los nodos relacionados con el nodo. La función *recuperar_nodos* se comporta de forma distinta dependiendo de la jerarquía que se haya establecido y la posición del token. Así, por ejemplo, (tal como se describió en el apartado 4.8), en la ruta /roomdevice/lab_b403/*, la jerarquía *roomdevice* indica que el primer token de la *lab_b403* es una habitación y que el * se refiere a un dispositivo. En este caso la función *recuperar_nodos* únicamente obtendrá aquellos nodos de tipo dispositivo que estén relacionados con el nodo que representa a la habitación *lab_b403*.

En el ejemplo que se muestra a continuación se obtienen todas las propiedades de *value* de los dispositivos que se encuentren en la habitación *lab_b403*.

Ejemplo de petición HTTP:

```
GET /interact/bb/get/roomdevice/lab_b403/*/props/value/
```

Ejemplo de mensaje de respuesta XML:

```
<GetResponse>
 <entity name="lab_b403">
  <entity name="lampv2">
   <property name="value">30</property>
  </entity>
  <entity name="lampv1">
   <property name="value">40</property>
  </entity>
 </entity>
</GetResponse>
```

- **Cambiar Contexto** *[SetContext]*

La pizarra recibe la ruta de la propiedad sobre la cual se quiere actuar y su nuevo valor. El comando de modificar se almacena en el *montón*[13] de órdenes hasta que se convierte en activo (véase el apartado 5.10). Entonces, se ejecuta el comando y se modifica el valor de la propiedad. Esta operación puede implicar o bien cambiar el valor de un nodo de la pizarra o bien acceder a un dispositivo. La petición del comando se codifica empleando únicamente la cabecera del protocolo HTTP, añadiendo el nuevo valor de la propiedad a la ruta del nodo. La respuesta será un mensaje XML en el cual se indica si la operación se ha realizado con éxito o si ha habido algún error.

Formato de la petición HTTP:

```
peticion  = 'PUT' ruta_pizarra '/set HTTP/1.0\n\r' mensaje
mensaje   = '<SetRequest>' et_ruta et_valor ' </SetRequest>'
et_ruta   = '<path>' ruta '</path>'
et_valor  = '<value>' valor '</value>'
valor     = 1*uchar
```

Al igual que la operación anterior, se permite codificar la petición en la URL directamente, sin tener que enviar el mensaje XML.

Formato de la petición HTTP:

```
peticion  = 'GET' ruta_pizarra '/set' ruta '?value' valor 'HTTP/1.0'
valor     = 1*uchar
```

[13] heap

Formato del mensaje de respuesta XML:

```
respuesta = '<SetResponse>' ( resp_ok | resp_err ) '</SetResponse>'
resp_ok   = '<ok></ok>'
resp_err  = '<error number="' digits '" msg="' causa '" />'
causa     = 1*uchar
```

En el ejemplo que se muestra seguidamente se utiliza el comando de **Cambiar Contexto** para cambiar encender la luz.

Ejemplo de petición HTTP:

```
GET /interact/bb/set/device/lamp_1/props/value?value=1 HTTP/1.0
```

Ejemplo de mensaje de respuesta XML:

```
<SetResponse>
    <ok><ok/>
</SetResponse>
```

- **Subscribirse** *[SubscribeContext]*

La pizarra incorpora un mecanismo de subscripciones mediante eventos. Los eventos que se pueden producir son:

- Cambiar del valor de una propiedad.
- Añadir una entidad a la pizarra.
- Borrar una entidad de la pizarra.
- Añadir una relación a una entidad.
- Borrar una relación de una entidad.

Para cada entidad y propiedad la pizarra guarda una lista de subscriptores, de forma que cuando se produce uno de los anteriores eventos la pizarra se encarga de informar a los clientes de la lista correspondiente. La retransmisión de los eventos se realiza utilizando TCP/IP. Por ello, cada cliente definirá una máquina y un puerto donde se encuentra escuchando los posibles eventos que se produzcan.

Cuando una aplicación quiere suscribirse a un cambio, envía un mensaje XML utilizando el protocolo HTTP en el que se codifica la siguiente información:

- **Máquina.** Dirección IP o nombre DNS de la máquina donde se ejecuta el cliente.
- **Puerto.** Puerto al cual se encuentra ligado el cliente para recibir los eventos.
- **Entidad.** Identificador de la entidad a la que el cliente se quiere subscribir.

- **Propiedad.** Identificador de la propiedad a la que la entidad se quiere subscribir.
- **Tipo.** Define qué tipos de entidades son del interés del cliente.
- **Comando.** Nombre del evento al que se quiere subscribir (cambiar propiedad, añadir entidad, borrar entidad ...)
- **Callback.** Cadena de caracteres que permite identificar o agrupar las subscripciones que realiza un mismo cliente.

Seguidamente se muestra el formato que se sigue para realizar una subscripción.

Formato de la petición HTTP:

```
peticion  = 'PUT' ruta_pizarra '/subscribe HTTP/1.0\n\r' mensaje
mensaje   = '<SubscribeRequest>'
                    maquina
                    puerto
                    entidad
                    propiedad
                    tipo
                    comando
                    callback
              '</SubscribeRequest>'

maquina   = host
puerto    = digits
entidad   = identificador
propiedad = identificador
tipo      = identificador
comando   = 'SET' | 'ADD' | 'REMOVE' |
            'ADDRELATION' | 'REMOVERELATION'
callback  = identificador

identificador = alpha | alpha *[ alphadigit | '-' ] alphadigit
```

La respuesta que recibiría el cliente tras realizar la subscripción tiene el siguiente formato:

Formato del mensaje de respuesta XML:

```
respuesta = '<SubscribeResponse>'
              ( resp_ok | resp_err )
            '</SubscribeResponse>'
resp_ok   = '<ok></ok>'
resp_err  = '<error number="' digits '" msg="' causa '" />'
causa     = 1*uchar
```

En el siguiente ejemplo un cliente que se ejecuta en la máquina 150.244.57.89 y escucha en el puerto 9000 solicita una subscripción que le avise de los campos de la entidad *lamp_1* en la propiedad *status* que es de tipo *device*, y establece como *callback* el texto *Ejemplo_Tesis*

Ejemplo de petición HTTP:

```
PUT /interact/bb/subscribe HTTP/1.0

<SubscribeRequest>
    <host>150.244.57.89</host>
    <port>9000</port>
    <entity>lamp_1</entity>
    <property>status</property>
    <type>device</type>
    <command>SET</command>
    <callback>Ejemplo_Tesis</callback>
</SubscribeRequest>
```

Todos los campos son obligatorios. Pero para los campos de entidad, propiedad, tipo y comando se puede establecer el valor -1. Esto indicaría que la subscripción se aplica a todos los elementos de ese campo. Así, una aplicación se puede subscribir a todos los eventos que se produzcan en la entidad *lamp_1* con sólo cambiar los campos propiedad, tipo y comando por el valor -1.

El formato de los eventos recibidos por los subscriptores es el siguiente:

Formato del mensaje de respuesta XML:

```
evento          = '<Event>' ( set | add | remove |
                              add_relation |remove_relation )
                  '</Event>'
set             = '<Set>' nodo '</Set>'
add             = '<Add>' nodo '</Add>'
remove          = '<Remove>' nodo '</Remove>'
add_relation    = '<AddRelation>' nodo '</AddRelation>'
remove_relation = '<RemoveRelation>' nodo '</RemoveRelation>'
```

donde el cambio producido en la pizarra varía según el tipo de evento. En el siguiente ejemplo se muestra un evento recibido por la modificación en una propiedad.

Ejemplo de evento recibido XML:

```
<Event>
    <set>
        <entity name="lamp_1">
            <property name="status">1</status>
        </entity>
   </set>
</Event>
```

- **Desubscribirse** *[UnsubscribeContext]*

Esta operación permite eliminar las subscripciones que tuviera un cliente. Para ello se envía un mensaje XML que contenga el *callback* de las subscripciones que se quieren borrar.

Formato de la petición HTTP:

```
peticion      = 'PUT' ruta_pizarra '/unsubscribe
                HTTP/1.0\n\r' mensaje
mensaje       = '<UnsubscribeRequest><callback>'
                     callback
                 '</callback></UnsubscribeRequest>'
callback      = identificador
```

El mensaje de respuesta tiene una estructura similar al de la operación *Subscribirse*

Formato del mensaje de respuesta XML:

```
respuesta     = '<UnsubscribeResponse>'
                    ( resp_ok | resp_err )
                  '</UnsubscribeResponse>'
resp_ok       = '<ok></ok>'
resp_err      = '<error number="' digits '" msg="' causa '" />'
causa         = 1*uchar
```

- **Añadir Contexto** *[AddContext]*

Las fuentes de información contextual aparecen y desaparecen dinámicamente del entorno; se requiere un mecanismo que permita reflejar estos cambios en el modelo del mundo. Ésta y las siguientes tres operaciones se encargan de esta tarea. El comando de *Añadir Contexto* permite incorporar nuevas entidades a la pizarra. Para ello, se envía un mensaje XML con la definición de la nueva entidad.

Formato de la petición HTTP:

```
peticion      = 'PUT' ruta_pizarra '/add
                HTTP/1.0\n\r' mensaje
mensaje       = '<AddContextRequest>' nodo '</AddContextRequest>'
```

La pizarra informará del resultado de la operación en el mensaje de respuesta.

Formato del mensaje de respuesta XML:

```
respuesta      = '<AddContextResponse>'
                      ( resp_ok | resp_err )
                  '</AddContextResponse>'
resp_ok        = '<ok></ok>'
resp_err       = '<error number="' digits '" msg="' causa '" />'
causa          = 1*uchar
```

En el siguiente ejemplo se añade a la pizarra la información contextual relativa a una persona que ha pasado a formar parte del entorno.

Ejemplo de petición HTTP:

```
PUT /interact/bb/add HTTP/1.0

<AddContextRequest>
<entity name="xavier" type="Person">
        <property name="Nombre">
            <paramSet name="default">
                <param name="initialValue">Xavier</param>
             </paramSet>
        </property>
        <property name="Apellidos">
             <paramSet name="default">
                <param name="initialValue">Alamán</param>
             </paramSet>
        </property>
        <property name="NumTarjeta">04494583</property>
        <property name="Correoe">Xavier.Alaman@uam.es</property>
        <relation name="tieneReunion" destination="ruth" />
        <relation name="tieneReunion" destination="german" />
        <relation name="tieneReunion" destination="pablo" />
    </entity>
</AddContextRequest>
```

- **Borrar Contexto** *[RemoveContext]*

Esta operación permite eliminar una entidad de la pizarra. Cuando se borra la entidad también desaparecen todas las relaciones que tuviera, tanto si la entidad actúa como fuente o como destino. Esta operación únicamente requiere establecer la ruta de la entidad que se desea eliminar.

Formato de la petición HTTP:

```
peticion      = 'PUT' ruta_pizarra '/remove
                HTTP/1.0\n\r' mensaje
mensaje       = '<RemoveContextRequest><path>'
                     ruta
                '<path></RemoveContextRequest>'
```

Esta operación también se puede agrupar en la codificación de la petición HTTP.

Formato de la petición HTTP:

```
peticion = 'GET' ruta_pizarra '/remove' ruta 'HTTP/1.0\n\r'
```

El siguiente ejemplo refleja que Xavier abandona el entorno, por lo que se elimina la entidad que lo representa del contexto.

Ejemplo de petición HTTP:

```
GET /interact/bb/remove/persons/xavier HTTP/1.0
```

- **Añadir Relaciones** *[AddRelationship]*

Este comando establece una relación unidireccional entre dos entidades. Cada relación vendrá definida por la entidad origen, la entidad fuente y el tipo de relación. Las dos entidades que conforman la relación tienen que especificarse utilizando la ruta absoluta.

Formato de la petición HTTP:

```
peticion      = 'PUT' ruta_pizarra '/addrelation' ruta
                'HTTP/1.0\n\r' mensaje
mensaje       = '<AddRelationRequest>
                     entidad
                     tipo
                '</AddRelationRequest>'
entidad       = '<entity name="' ruta '></entity>'
tipo          = '<type name="' identificador '></type>'
```

Por ejemplo, para establecer una relación entre un habitación y un dispositivo que ha sido incorporado recientemente se tendría que enviar el siguiente mensaje.

Ejemplo de petición HTTP:

```
PUT /interact/bb/addrelation/rooms/lab_b403 HTTP/1.0

<AddRelationRequest>
    <entity name="/devices/lamp_1"></entity>
    <type name="contiene"></type>
</AddRelationRequest>
```

- **Borrar Relaciones** *[RemoveRelationship]*

Recíprocamente a la operación anterior, **Borrar Relaciones** elimina una relación entre dos entidades de la pizarra. De nuevo es necesario determinar cuáles son la entidad origen y destino, y el tipo de relación que se quiere eliminar.

Formato de la petición HTTP:

```
peticion      = 'PUT' ruta_pizarra '/removerelation' ruta
                'HTTP/1.0\n\r' mensaje
mensaje       = '<RemoveRelationRequest>
                    entidad
                    tipo
                '</RemoveRelationRequest>'
entidad       = '<entity name="' ruta '></entity>'
tipo          = '<type name="' identificador '></type>'
```

En un ejemplo que eliminara la relación que se añadió anterior el mensaje quedaría:

Ejemplo de petición HTTP:

```
PUT /interact/bb/removerelation/rooms/lab_b403 HTTP/1.0

<RemoveRelationRequest>
    <entity name="/devices/lamp_1"/>
    <type name="contiene"/>
</RemoveRelationRequest>
```

5.7. Controladores de dispositivo

No todos los componentes del entorno tienen por qué tener unas capacidades de computación suficientes como para poder interaccionar empleando HTTP y XML. Por otro lado existen multitud de componentes con estándares de comunicación diferentes. Por estos motivos se ha añadido un mecanismo de pasarela entre la pizarra y otros dispositivos que no soporten HTTP o XML.

En el caso concreto del entorno sobre el cual se ha probado la capa intermedia la gran mayoría de los dispositivos físicos son controlados mediante el protocolo SNMP[174]. La interconexión entre la pizarra y la capa SNMP se realiza mediante un controlador específico que se encuentra integrado dentro de la arquitectura de la pizarra.

Para cada entidad se pueden distinguir las propiedades que son externas a la pizarra de las que son almacenadas por ésta. Para ello se emplea el parámetro *localización* (véase el apartado 5.6.2). Este parámetro indica si una propiedad es interna a la pizarra, o en caso contrario cuál es el nombre del controlador que se utiliza para accederla. Este controlador requiere información adicional de la entidad y la propiedad, como puede ser el nombre de la entidad en el espacio de

nombres externo. Esta información se añade mediante los parámetros extendidos que proporciona el lenguaje BBXML (véase el apartado 6.2.3)

La comunicación entre el controlador y el dispositivo se realiza mediante una pasarela intermedia que transforma los comandos del protocolo SNMP en comandos del protocolo de comunicación propietario del dispositivo. Tal como se describe en el apartado 7.3.1, los dispositivos de control del entorno están conectados al bus domótico estándar EIB.

En la red SNMP los dispositivos se encuentran divididos por áreas. Cada área tiene su propia pasarela. En la pizarra se definen las áreas que tiene el entorno, qué dispositivos pertenecen a cada área, y para cada área se definen la dirección IP y el puerto donde se encuentra ubicado el servidor que actúa de pasarela. Por otro lado, para cada dispositivo se añade la dirección dentro de la red SNMP, que consiste en un identificador único para cada área.

Cuando la capa de contexto recibe una petición que afecta a una propiedad externa realiza los siguientes pasos para obtener el valor de la propiedad.

1. Empleando la información del parámetro *localización* se lanza el controlador pertinente.

2. El controlador recorre las áreas definidas en el entorno hasta que encuentra el área donde se encuentra el dispositivo.

3. Recupera la información de localización de la pasarela: parámetros *AreaIPAddress* y *AreaPortAddress*

4. Reenvía la petición a la pasarela añadiendo la dirección del dispositivo (parámetro *DeviceAddress*).

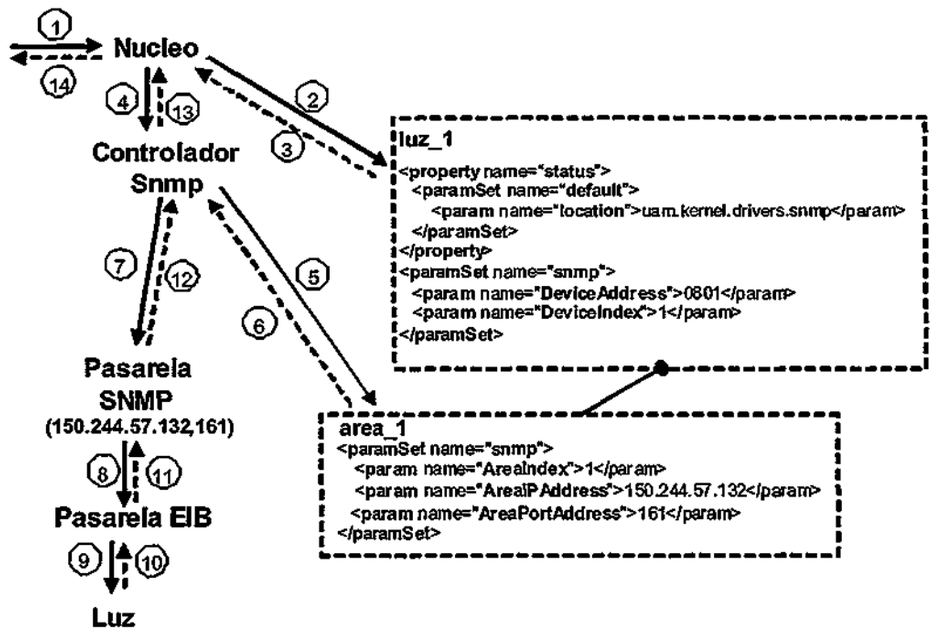
5. Recibe la respuesta a la petición y la reenvía al cliente de la capa de contexto.

El controlador incluye una memoria *cache* que le permite reducir el tiempo de búsqueda del paso 2, en las propiedades más consultadas. Además también se incluye un temporizador para que el controlador no espere indefinidamente la respuesta de la pasarela en el paso 5.

En la figura 5.7 se puede visualizar un resumen del proceso de interacción entre el controlador y la pasarela.

5.8. Recordando el pasado

Las *aplicaciones sensibles al contexto* requieren de mecanismos que les permitan recordar los cambios en el contexto (véase el apartado 2.4.1). Una funcionalidad importante que debe estar presente una capa intermedia es poder almacenar y etiquetar con información contextual los cambios que se produzcan en el contexto. Esta funcionalidad se suele implementar con un mecanismo de registro de eventos similar al que se puede encontrar en otro tipo de aplicaciones. Este mecanismo puede ser:



1- Petición: consultar estado de la luz
2- El núcleo comprueba la localización de la propiedad.
3- Respuesta que incluye el nombre del controlador y la dirección del dispositivo
4- Se ejecuta el controlador
5- El controlador busca el área a la que pertenece el dispositivo
6- Respuesta con la dirección IP y puerto de la pasarela SNMP
7- Envío de la petición a la pasarela
8- La pasarela reenvía la petición a la pasarela EIB
9- La pasarela EIB consulta al dispositivo luz conectado al bus EIB
10- Respuesta del microcontrolador que acciona la luz
11- Reenvío de la respuesta a la pasarela SNMP
12- Reenvío de la respuesta al controlador
13- Reenvío al núcleo
14- Respuesta a la petición.

Figura 5.7: Descripción de los pasos que se siguen cuando se recibe una petición sobre una propiedad externa a la pizarra

- **Centralizado.** Todos los cambios en el contexto los recibe un único modulo que se encarga de registrarlos. Esta estrategia es muy adecuada para una arquitectura centralizada como es una pizarra, pero también funciona con arquitecturas distribuidas, ya sean objetos distribuidos como la Context Toolkit [86] o agentes *software*.

- **Distribuido.** Los cambios se almacenan en cada uno de los componentes de la arquitectura. Esta solución permite repartir el almacenamiento de los registros, y reduce las comunicaciones, a costa de complicar la recuperación de los datos.

El diseño de la pizarra propuesto facilita la implementación de un módulo de registro centralizado. El mecanismo de comunicación obliga a que todos los mensajes que se intercambien pasen por la pizarra. Así, implementar un registro de los cambios es tan sencillo como almacenar los mensajes que llegan a la pizarra. El módulo provee de una operación que permite recuperar los cambios. Esta operación admite filtrar la información según el tipo de entidad, propiedad y un intervalo de tiempos.

De cada evento se almacena una información común a todos los eventos y otra que varía según el comando que se haya realizado. De esta forma, en cada registro siempre está la fecha cuándo se produjo y el tipo de comando realizado. Si la operación consiste en cambiar el valor de una propiedad se incluirá también el identificador y el tipo de la entidad, el nombre de la propiedad y el nuevo valor. Si la operación es un cambio en una relación aparecerá el nombre de la relación y los nombres de las entidades fuente y destino.

En el ejemplo de la figura 5.8 se puede observar una secuencia de eventos registrados correspondientes a la entrada de una persona en una habitación. La persona se identifica con la entidad que tiene por nombre *xavier* mientras que el entorno se denomina *labB403*. Al entrar al entorno la persona tiene que emplear una tarjeta: la lectura de esa tarjeta es el primer evento que se produce. Si la tarjeta leída tiene permiso para poder acceder se produce la apertura de la cerradura de la puerta. Un sensor colocado en la puerta detecta la apertura. También se aprecia cómo la luz del salón *lamp_1* como la luz de ambiente *lampv1* se encienden automáticamente. Finalmente, se añade el nuevo ocupante a la pizarra, y se añaden las relaciones correspondientes para indicar que esa persona está dentro de la habitación.

5.9. Acceso a los dispositivos

En un entorno compartido por numerosos componentes distribuidos surgen diversos problemas en el control de acceso a los recursos. Uno de los problemas que aparece con más frecuencia es decidir quién puede acceder a cada uno de los recursos. Dentro del área de los *entornos inteligentes* se han propuesto diferentes soluciones que se resumen en dos enfoques: basadas en políticas [156] y basadas en listas de control de acceso [220]. Este último enfoque es el que se ha seguido en nuestro caso.

```
2006-01-23 18:25:15,219 Cmd:set Name:tarjeta Type:Sensor Prop:tarjeta_leida Val:0102ac5584
2006-01-23 18:25:15,399 Cmd:set Name:tarjeta Type:Sensor Prop:tarjeta_leida Val:0
2006-01-23 18:25:17,402 Command:set Name:lamp_1 Type:Lamp Prop:status Val:1
2006-01-23 18:25:17,933 Command:set Name:lampv1 Type:DimmableLamp Prop:value Val:45
2006-01-23 18:25:18,113 Command:set Name:puerta1 Type:Lock Prop:status Val:1
2006-01-23 18:25:18,984 Command:set Name:detector_puerta Type:sensor Prop:value Val:1
2006-01-23 18:25:19,164 Cmd:add Name:xavier Type:Person
2006-01-23 18:25:20,276 Command:set Name:lab_b403 Prop:habitants Value:1
2006-01-23 18:25:21,167 Cmd:addRelation Name:located-in Src:xavier Srk:labB403
2006-01-23 18:25:27,146 Cmd:addRelation Name:contains Src:labB403 Srk:xavier
```

Figura 5.8: Ejemplo de sucesión de eventos registrados por la capa de contexto

El diseño de la lista de acceso se ha realizado manteniendo el modelo de datos descrito en el capítulo 4. De igual manera a como se añadió el contexto secundario (véase el apartado 4.7) se ha extendido el modelo para gestionar el acceso a los dispositivos.

Se han añadido los conceptos *Agente* y *Grupo*. Un agente es cualquier recurso que trabaje dentro del ámbito del entorno, y que tenga capacidad de decisión por sí mismo para realizar acciones que afecten al mundo físico o a otros agentes. En este sentido, no se encuentran incluidos dentro de esta categoría todos aquellos recursos que se encarguen de transmitir la entrada del usuario directamente, o que se limiten a recibir órdenes. Así, una interfaz de usuario o un interruptor no son agentes, sino herramientas en manos de la voluntad del usuario. A efectos del control de acceso, los comandos transmitidos por estos recursos es como si hubieran sido enviados por una persona.

Un *Grupo* es una colección de agentes compuesto de al menos un agente. Un grupo se crea añadiendo una entidad a la pizarra que lo representa. Esta acción la tiene que llevar acabo un agente, al que se le designa como propietario del grupo, y que tiene los derechos de administración del mismo. Únicamente el propietario del grupo tiene capacidad para añadir o borrar nuevos miembros, o para eliminar definitivamente el grupo. La inclusión de un nuevo agente dentro de un grupo se representa mediante una relación bidireccional *miembroDe/tieneMiembro*. No se limita el número de grupos a los que puede pertenecer un agente.

La gestión de los dispositivos se completa mediante cinco relaciones adicionales y una propiedad que se incluye en el recurso. Dependiendo de las relaciones establecidas se va a decidir si un agente puede o no manejar un dispositivo.

- **esPropietario/tienePropietario.** Esta relación define quién tiene los derechos de administración. Cada dispositivo tiene un único propietario que puede ser un agente o un grupo. El propietario del dispositivo es el único que tiene derecho para cambiar las relaciones que controlan el acceso al dispositivo.

- **esAgentePermitido/tieneAccesoPermitido.** Por cada agente que pueda controlar un dispositivo se establecen dos relaciones, una entre el agente

y el dispositivo de tipo *esAgentePermitido*, y otra entre el dispositivo y el agente del tipo *tieneAccesoPermitido*.

- **esAgenteRestringido/tieneAccesoRestringido.** De forma similar a la anterior se establecen estas dos propiedades cuando un agente tiene el acceso denegado a un dispositivo.
- **esGrupoPermitido/tieneGrupoPermitido.** Esta relación es equivalente a que cada miembro del grupo tuviera una relación *esAgentePermitido/tieneAccesoPermitido* con el dispositivo. De esta forma, se permite agrupar diferentes agentes en una sola relación reduciendo el número de éstas en la pizarra.
- **esGrupoRestringido/tieneGrupoRestringido.** Igual que la anterior pero con la diferencia de que en este caso todos los miembros del grupo tienen el acceso denegado.
- **permitidoDefecto.** Esta propiedad de los recursos indica, cuando el valor es verdadero, que por defecto se permiten todas las conexiones excepto aquellas que se indiquen como denegadas. En cambio, si el valor es falso indica que sólo se permiten aquellas conexiones que se definan explícitamente.

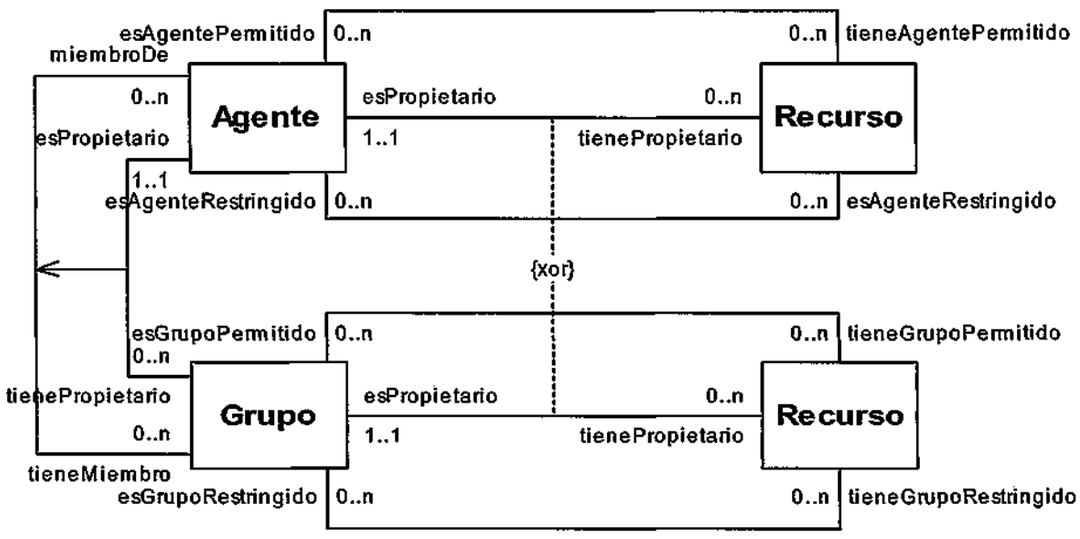


Figura 5.9: Relaciones entre agentes y recursos

Cuando un agente pretende acceder a una propiedad de un dispositivo (véase el apartado 5.5) la capa intermedia determina en función de cómo estén establecidas las relaciones anteriores si el acceso tiene validez o no. Para ello se emplea el siguiente algoritmo:

Pseudocódigo:

```
FUNCION permitirAcceso(id_dispositivo, id_agente)
DEVUELVE Verdadero ó Falso

    SI tienePropietario(id_dispositivo, id_agente)
        DEVUELVE Verdadero
    FIN SI

    SI permitidoDefecto ENTONCES
        // Por defecto está admitido el acceso
        SI tieneAgenteRestringido(id_dispositivo,id_agente)
        ENTONCES
            DEVUELVE Falso
        FIN SI

        PARA CADA id_grupo EN miembroDe(agente) HACER
            SI tienePropietario(id_dispositivo, id_grupo)
            ENTONCES
                DEVUELVE Verdadero
            FIN SI
            SI tieneGrupoRestringido(id_dispositivo, id_grupo)
            ENTONCES
                DEVUELVE Falso
            FIN SI
        FIN PARA CADA
        DEVUELVE Verdadero
    SINO
        // Por defecto está denegado el acceso
        SI tieneAgentePermitido(id_dispositivo,id_agente)
        ENTONCES
            DEVUELVE Verdadero
        FIN SI

        PARA CADA id_grupo EN miembroDe(agente) HACER
            SI tienePropietario(id_dispositivo, id_grupo)
            ENTONCES
                DEVUELVE Verdadero
            FIN SI
            SI tieneGrupoPermitido(id_dispositivo, id_grupo)
            ENTONCES
                DEVUELVE Verdadero
            FIN SI
        FIN PARA CADA
        DEVUELVE Falso
    FIN SI
FIN FUNCION
```

Por defecto, el acceso está concedido o restringido dependiendo del valor de la propiedad *permitidoDefecto*. Para comprobar si cumple las condiciones de acceso al recurso el agente tiene que satisfacer los siguientes requisitos. Lo que primero se comprueba es si el agente es el propietario del recurso. En caso afirmativo se le permite el acceso, y no se realizan más comprobaciones. En caso negativo, se comprueba el valor de la propiedad *permitidoDefecto*. Si el valor es verdadero significa que se parte de que el permiso está concedido y que hay que revisar si existe alguna relación que se lo deniegue. En el caso de que la propiedad sea falsa, se parte de que el permiso está denegado y hay que verificar las relaciones que se lo puedan conceder.

En el primer caso se comprueba si el agente se encuentra en alguna de las listas que restringen el acceso. Si no estuviera en ninguna de las listas, se pasaría a comprobar cada grupo a los que el agente pertenece. En cuanto alguno de los grupos sea propietario del recurso, se le permite el acceso y se deja de comprobar el resto. En cambio, en cuanto uno de los grupos tenga el acceso vetado, el agente tendrá automáticamente el acceso restringido. Si se ha terminado con la comprobación de todos los grupos, y no se ha dado ninguna de las circunstancias anteriores, entonces se le concede el permiso.

En el segundo caso se procede de manera similar pero cerciorándose de si alguno de los grupos a los que pertenece tiene el acceso permitido. Si es así, se le permite el acceso al recurso, si no se le deniega.

Un ejemplo de cómo funciona el mecanismo se encuentra en la figura 5.10. Se parte de que la propiedad *permitidoDefecto* tiene el valor falso. *Dave* puede modificar el altavoz y la lampara. El primero por ser el propietario de ese recurso y el segundo por estar incluido en la lista de agentes de confianza. Por el contrario, *Dave* no puede abrir la cerradura *Cerradura1* ya que está en la lista de agentes restringidos. Esta prohibición se mantiene aún a pesar de pertenecer al grupo *Niños* que sí tiene acceso a la cerradura.

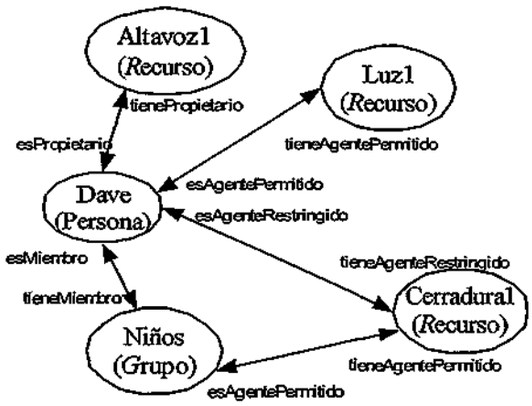


Figura 5.10: Ejemplo de mecanismo de acceso empleando listas control de acceso

Un problema por resolver es realizar una heurística eficaz para asociar dispositivos de entrada a las personas que los están empleando. En una interfaz

de usuario gráfica es sencillo establecer dicha relación si la aplicación solicita un usuario y una contraseña al iniciarse. De esta manera todas las acciones realizadas desde la interfaz se asociarían a esa persona. Cuando se emplean dispositivos físicos esta asociación se complica por dos motivos. Primero, la identificación del usuario puede que no se pueda realizar de una manera tan directa. Dependiendo de la tecnología disponible está podrá ser más o menos sencilla. Si el entorno dispone de algún tipo de control de entrada, sería al ingresar cuando se realizará la identificación, ya sea mediante tarjetas RFID, reconocimiento de huellas, llaves digitales, técnicas de identificación del locutor, reconocimiento de rostros, etc. Una vez identificado, cada acción sobre un actuador del entorno llevará asociado al ocupante del entorno en ese instante. La segunda complicación surge cuando hay más de una persona en la habitación y se debe determinar cuál de los ocupantes es el que ha accionado un determinado dispositivo. Una posible solución sería dotar a los objetos con etiquetas RFID y a los ocupantes con antenas detectoras de las etiquetas. Un usuario al acercarse a un dispositivo para accionarlo, leerá su etiqueta, y la distribuirá junto con su identificación a la capa intermedia. De esta forma se realizaría la asociación correspondiente entre usuario y actuador.

En nuestro caso, se ha optado por una solución más sencilla de implementar pero eficaz. En el caso de existir varios usuarios, las acciones que se realizan sobre un dispositivo se asocian siempre a la persona que tenga más privilegios sobre el dispositivo. Así, por ejemplo, si hay tres personas en el entorno de los cuales dos tienen prohibido el acceso a la televisión y otro lo tiene habilitado, se permite a todas las personas el acceso a ésta. Para ello, todas las acciones que se realicen sobre el televisor se asocian al tercer usuario independientemente de quién la haya realizado realmente. Nuestra aproximación se basa en delegar la responsabilidad sobre el usuario con más acceso. Se entiende que en estos casos rigen las normas sociales que existen entre las personas. Si la persona con privilegios está de acuerdo en encender la televisión, no importa quién de los tres sea el que lleve a cabo la acción. Cómo lleguen a un acuerdo las tres personas, o cómo ejercerá su autoridad el tercer ocupante en el caso de que no quiera encender la televisión, es un problema que se traslada al plano social.

5.10. Resolviendo conflictos

Una vez establecido el mecanismo de control de acceso hay que afrontar el problema que surge cuando dos o más componentes pretenden controlar de manera simultánea un mismo recurso (por ejemplo un dispositivo, digamos, una lámpara). La solución al conflicto pasa por elegir a qué componente se le concede el permiso para acceder primero al recurso. Un problema relacionado con éste ha sido estudiado dentro del dominio de los sistemas distribuidos y se ha denominado exclusión mutua distribuida.

Las soluciones más cercanas a los entornos inteligentes se encuentran en los sistemas multiagente. La Open Agent Architecture [173] o Hive (véase el apartado 3.3.1) son dos plataformas que han abordado este problema de forma distribuida. Otra aproximación es realizar una implementación centralizada [105]. Para ello

se requiere de un componente coordinador que se encarga de recibir todas las peticiones y que decide a qué componente se otorga el permiso para acceder al recurso. Los algoritmos distribuidos aportan frente a la solución centralizada una mayor robustez, al no depender de un componente central. Por contra, son menos eficientes en cuanto al número de mensajes que se tienen que transmitir entre los procesos, y adolecen de una mayor complejidad de implementación.

Nuestra propuesta para solventar conflictos dentro de un *entorno inteligente* aboga por la simplicidad y la eficiencia. La capa de contexto que sirve de pegamento entre los componentes del entorno inteligente descansa en un repositorio común de información, lo cual facilita la implementación de un algoritmo centralizado. Por ello, se ha elegido una solución centralizada.

En los escenarios que se plantean en un entorno inteligente, la priorización entre las peticiones se torna esencial. En general, es frecuente que los usuarios del entorno tengan diferentes roles (padre/hijo, propietario/invitado, profesor/alumno, administrador/usuario...), y que se quiera establecer distinción según la acción la realice uno u otro. De esta forma a un usuario se le da preferencia en el acceso en función de rol que desempeñe. Ahora bien, en un entorno inteligente se plantea otra cuestión a resolver que los mecanismos anteriormente citados no tienen en cuenta. Son frecuentes las situaciones en las que un usuario requiere asumir el control de un recurso que está siendo empleado por otro. Teniendo en cuenta los distintos roles y la situación del entorno, la decisión a tomar puede ser la de arrebatar el control al componente que actualmente tiene en su posesión el recurso. Este tipo de decisiones son críticas en escenarios como el hogar; por ejemplo, los módulos de seguridad (gas, inundación, antirrobo...) deben tener preferencia frente al resto de los módulos en caso de emergencia.

Nuestra propuesta consiste en un mecanismo de colas centralizado en el que se prioricen las acciones que se quieren realizar sobre los recursos. Las peticiones se almacenan en la cola teniendo en cuenta la prioridad, de forma que la que se extrae primero es la que más prioridad tiene. Se ha decidido emplear colas de prioridades preferentes. Estas colas se caracterizan por que si la petición que llega tiene más preferencia que la que está en curso, le arrebata el control del recurso. En contraposición, en las colas no preferentes la petición que llega tiene siempre que esperar a que termine la petición en curso. Los requerimientos de seguridad y confort obligan a que las peticiones más urgentes puedan obtener el control del recurso en cualquier momento. Una acción del usuario debe imponerse sobre cualquier acción automática. Por ejemplo, el riego automático ha sido encendido por un agente *software* que ha determinado que para el cuidado del jardín tiene que estar encendido una hora. La acción de encender el riego se mantendrá activa durante todo ese tiempo, quedando el recurso bajo el control de ese agente. Ahora bien, el usuario decide apagar el riego antes de que se cumpla la hora programada. La acción del usuario se envía a la cola asociada al aspersor. Si la cola no es preferente, debería esperar a que pasara el tiempo que resta hasta la hora para poder apagar el aspersor, mientras que en caso contrario la orden se ejecutaría instantáneamente.

La elección de colas preferentes obliga a tener en cuenta consideraciones adicionales. Los agentes deben contemplar la posibilidad que su petición puede que

no sea atendida. Esto es así porque las peticiones de baja prioridad pueden permanecer indefinidamente en la cola, si llegan continuamente peticiones de mayor prioridad. Así, hay que atender el caso en el que un componente con elevada prioridad se adueñe de forma indefinida del recurso. Un método simple y efectivo de solventar la espera indefinida es establecer un tiempo límite para la reserva de un recurso. Por ejemplo, en la capa intermedia Jini[90] se plantea un mecanismo que consiste en permitir a un componente reservar el acceso de un recurso durante un tiempo fijo. Para seguir manteniendo el control del recurso el componente tiene que renovar la concesión antes de que caduque. De forma similar, para evitar que los procesos se queden perpetuamente esperando a que se cumpla la petición, se incorpora un tiempo de caducidad a las peticiones de forma que pasado este tiempo dejan de ser válidas. Este límite temporal es establecido por el proceso que emite la petición, de forma que puede controlar si ha caducado o no.

5.10.1. Descripción del funcionamiento de las colas de prioridades

Cuando un agente pretende realizar una operación, envía un comando que es recibido por la capa de contexto. Las operaciones de Modificar propiedad y de Añadir/Borrar relación cambian el estado de las entidades y la configuración del entorno, por lo que son susceptibles que se produzcan conflictos si varios agentes envían una operación que afecta a la misma propiedad o relación.

Para solventar esta situación, se provee de una cola por cada propiedad de una entidad y por cada relación entre dos entidades. Cuando se envía un comando de los anteriormente descritos se le asigna una prioridad, además de otra serie de parámetros. A continuación el comando se almacena en la cola correspondiente. La cola de prioridades a la cual se envía un comando se determina en función de la propiedad o relación sobre la cual se quiere actuar.

En cada instante de tiempo, cada cola se compone de un conjunto de comandos ordenados según su prioridad; aquel que tiene mayor prioridad se denomina comando activo. Cuando llega un nuevo comando se almacena en la cola de forma que si su prioridad es mayor que la prioridad del comando activo, el recién llegado pasa a ser el comando activo. En caso contrario, se guardará en el lugar correspondiente de la cola. Siempre que un comando se activa, éste se ejecuta una vez y se aplican los cambios pertinentes en la pizarra.

Para evitar el problema de que un comandos con la máxima prioridad bloquee la cola indefinidamente se establece un tiempo de caducidad para cada comando. De esta manera, una vez que se ha ejecutado el comando permanecerá como comando activo hasta que caduque o hasta que otro comando con mayor prioridad le reemplace. Un comando se ejecutará tanta veces como se active, es decir, tantas veces como acceda a la primera posición de la cola y no haya caducado.

Una restricción que se impone es que para cada cola sólo puede existir un comando por cada agente. Si un agente envía un comando a una cola, se elimina previamente el comando anterior, en el caso de que existiera. Por otro lado, los agentes tienen la capacidad de anular un comando de una cola determinada, o

todos los comandos enviados a todas las colas.

Las colas se crean y se destruyen según las necesidades. Cuando llega un comando se comprueba si ya existe la cola correspondiente, y en caso contrario se crea. Cuando una cola se queda vacía de comandos se elimina. De esta forma podrá haber tantas colas de prioridades como propiedades y relaciones haya en la pizarra, pero sólo ocuparan recursos aquellas colas que tengan una o más órdenes.

En la figura 5.11 se muestra el comportamiento de una cola según llegan diferentes comando con diferentes prioridades y tiempo de caducidad. Se representa una cola de tamaño cuatro a la que llegan acciones —relacionadas con cambios en la sintonización de una televisión— durante cinco instantes de tiempo. Cada comando se representa por su prioridad, el nombre del agente emisor y la caducidad. En el instante (a) un usuario (agente 1) decide silenciar el altavoz de la habitación. Esto se traduce en el envío de un comando para apagar el altavoz con prioridad 25 y caducidad 7. En el instante (b) una aplicación de mensajería (agente 2) tiene que distribuir un mensaje vocal urgente. Esto produce que se envíe una nueva orden con una prioridad mayor pidiendo que se suba el volumen del altavoz (además se enviarán otros comandos que permitan distribuir el mensaje). Dado que la prioridad es superior, pasa a ocupar el primer puesto en la cola desplazando al comando anterior. Así, el volumen del altavoz sube cerca del máximo. En el instante (c) entra en la habitación un nuevo usuario (agente 3). A este usuario le gusta escuchar la música alta, de modo que las preferencias del usuario disparan un comando automático —que pretende ajustar el altavoz a un volumen alto— con un prioridad de 15. Como es la menor de todas se coloca al final de la cola. En el instante (d) el comando que envío la aplicación de mensajería ha caducado con lo que desaparece de la cola. Asciende el comando de mayor prioridad que corresponde al agente 1. De nuevo el volumen del altavoz se silencia. Finalmente, en el instante (e) se elimina de la cola el comando del agente 3 al caducar. Este comando no se ha llegado a ejecutar ya que no ha podido llegar a colocarse en el primer puesto de la cola. El comando del agente 1 permanecerá dos instantes más en la cola, y también dejará de estar activo. Hay que tener en cuenta que aunque deje de estar activo el altavoz permanecerá silenciado ya que no hay ningún otro comando en la cola que modifique el volumen.

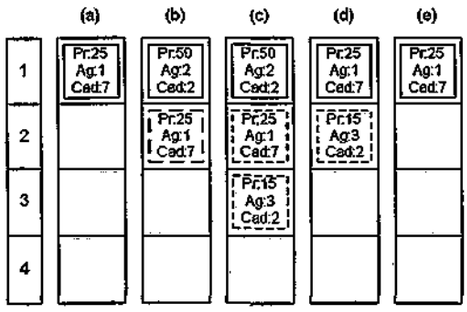


Figura 5.11: Ejemplo simplificado del funcionamiento de la cola de prioridades.

Parámetros de un comando

Se establecen una serie de parámetros para cada comando. Éstos son los siguientes:

- **Propietario del comando.** Identifica qué agente ha enviado el comando.
- **Prioridad.** Un entero positivo que indica la prioridad en la ejecución. Cuanto mayor sea el número, mayor será la prioridad. Las órdenes con prioridad cero se descartan directamente.
- **Modo.** Determina cómo se calcula caducidad del comando (véase el tabla 5.1).
- **Tiempo de caducidad.** Indica a partir de qué momento el comando deja de tener validez. Este parámetro sólo tiene validez si el modo elegido es *fecha de caducidad* (véase la tabla 5.1).
- **Nombre.** Asocia un nombre al comando. Dado un propietario, este parámetro permite distinguir entre los comandos emitidos. Este nombre permite al agente referenciar al comando para eliminarlo de la cola. Varios comandos en distintas colas se pueden agrupar bajo el mismo nombre, de tal forma que se pueden borrar todos a la vez.

El propietario del comando, modo, tiempo de caducidad y nombre son establecidos por el agente que envía el comando, mientras que la prioridad la decide la capa de contexto en función de la política de prioridades (véase el apartado 5.10.2).

La tabla 5.1 describe los distintos modos en que el emisor puede elegir la caducidad de un comando.

5.10.2. Políticas para establecer las prioridades de los comandos

El mecanismo de prioridades decide el orden en que se ejecutaran un conjunto de comandos que actúan sobre un recurso. Este orden se establece en función de la prioridad del comando y del tiempo de validez de esa prioridad. Sin embargo, no se ha definido en el mecanismo anterior cómo asignar la prioridad a un comando. Para ello se dispone de un módulo que se encarga de asignar las prioridades de los comandos. Cada política es una función que decide, dado un comando, cuál es la prioridad que se debe asignar. Se podrían definir tantas políticas como recursos se quieran controlar, aunque por motivos prácticos todos los comandos se tratan siguiendo una única política por defecto; y aquellos casos particulares que no se ajustan a la política por defecto se tratan por separado mediante políticas específicas. El administrador del entorno es el encargado de decidir para cada recurso cuál es la política que se quiere aplicar.

A continuación se describe la política que se emplea por defecto. El diseño de esta política viene afectado por dos consideraciones. Por un lado, la política

Modo	Descripción
DESCONOCIDO	Se utiliza cuando se desconoce el modo de caducidad que se desea. Se emplea por defecto en caso de que no se establezca ninguno, de esta forma se consigue la compatibilidad hacia atrás con aplicaciones existentes. Cuando se emplea este modo la caducidad de la orden se calcula cómo si el modo fuera DESPUÉS DE EJECUTARSE.
FECHA DE CADUCIDAD	La caducidad de la orden se establece mediante el tiempo en milisegundos que indique el parámetro *fecha de caducidad.*
DESPUÉS DE EJECUTARSE	El comando desaparece de la cola de prioridad una vez que se haya ejecutado. Por defecto se establece un tiempo de caducidad máximo de modo que si el comando no se ejecutó en ese tiempo se elimina.
INSTANTÁNEA	La orden no se almacena en la cola: si no existe ninguna orden con mayor prioridad en la cola, se ejecuta y desaparece.

Tabla 5.1: Distintos modos de establecer la caducidad de un comando

tiene que ser lo suficientemente flexible y genérica para abarcar el mayor número posible de casos. Por otro lado, la implementación tiene que ser sencilla para que no suponga una sobrecarga excesiva.

Para sopesar la prioridad del comando se tienen en cuenta distintas variables. Éstas se pueden dividir en tres grupos:

1. Aquellas relativas al emisor de la orden.
2. Aquellas relativas al comando.
3. Aquellas relativas al entorno.

Variables relativas al emisor de la orden

Dentro del primer grupo se incluyen quién es el emisor de la orden y el rol con el cual está actuando. La primera variable puede tener dos valores, o bien que el comando provenga de un **usuario** (U) o bien de una **aplicación** (A). Esta variable indica realmente quién es el que ha decidido emitir la orden, ya que los comandos provenientes de los usuarios siempre lo son a través de algún tipo de interfaz, ya sea una componente *software* (una interfaz gráfica, textual, vocal...) o un componente físico (un interruptor, un panel de control, un regulador...). En contraposición, se entiende que el emisor es una aplicación cuando la decisión del comando la toma una aplicación sin intervención directa del usuario, como por ejemplo, un agente que se encarga de establecer las preferencias del usuario cuando cambia el contexto. Por otro lado, el **emisor del comando**, independientemente

de que sea un usuario o una aplicación, puede asumir distintos roles dependiendo de la relación entre él y el recurso que se quiere modificar. El emisor puede ser **propietario** (O) del recurso, o puede ser **invitado** (G). Ambos roles se indican según exista o no una relación en la pizarra del tipo *esPropietario* entre la entidad que representa al emisor y el recurso. Además, si el emisor es **esPropietario** del entorno, se considera que es propietario de todos los recursos que se encuentren dentro de éste.

Variables relativas al comando

El segundo grupo está formado por una única variable que indica el **tipo de orden**. Ésta puede tomar distintos valores según quién emita el comando. Si el emisor es un usuario, el tipo de orden puede ser un **comando directo** (C) o una **preferencia de usuario** (P). En el primer caso, la orden se ha generado como resultado de una acción directa del usuario, como por ejemplo, pulsar un interruptor. En el segundo caso, la orden se origina automáticamente tras cumplirse una condición que el usuario previamente había definido. En el caso de que el emisor sea una aplicación, se definen dos tipos de comandos según que correspondan a **comandos ordinarios** (L) o **comandos de seguridad** (H). Estos últimos se definen como aquellos comandos que provienen de aplicaciones que se encargan de la seguridad del entorno.

Variables relativas al entorno

Finalmente, el último grupo engloba dos variables relativas al estado del entorno (véase el apartado 4.7.1). La primera define el **nivel de alerta** que puede ser **normal**, de **emergencia** o **intrusión**. El valor *emergencia* se reserva para situaciones graves como incendio o inundación. El valor *intrusión* corresponde con intrusiones en la vivienda. La segunda variable define el **nivel de seguridad**. Se definen tres niveles de seguridad: **bajo**, **normal** y **alto**. Una habitación con nivel de seguridad bajo correspondería a una localización pública y compartida por varios usuarios, un nivel normal se podría establecer para localizaciones privadas y pertenecientes a un usuario, y un nivel alto, para entornos que tuvieran restricciones de seguridad importantes.

Cálculo de la prioridad

Cuando se recibe un comando, se recupera el valor de cada una de las variables anteriores. Cada posible emisor tiene un código único que le autentifica y que se envía junto con la orden. Este código permite averiguar a partir de la definición del emisor en la pizarra quién es el emisor de la orden y qué tipo de órdenes emite. Mediante la relación entre el emisor y el recurso que se quiere modificar se halla el rol de emisor. Finalmente, se obtienen el estado de alerta y de seguridad del entorno a partir de las propiedades de la entidad entorno. A continuación se forma una tupla con las variables correspondientes a los grupos (a) y (b), de tal forma que el primer elemento defina el emisor, el segundo el rol, y el tercero el tipo de orden. Por ejemplo, un usuario propietario del recurso que ha enviado una orden

directa se define mediante la tupla (UOC), mientras que una aplicación invitada sería la tupla (AGL). La prioridad se calculará según la posición que ocupa la tupla en una lista. El orden de la lista es función del estado de alerta del entorno y del nivel de seguridad. Tal como se muestra en la Tabla 5.2, se han definido cuatro listas donde se ordenan las tuplas de mayor a menor prioridad.

Tipo Lista	Orden de preferencia							
Bajo	UOC	UGC	AOH	UOP	AOL	UGP	AGH	AGL
Normal	UOC	AOH	UGC	UOP	AOL	UGP	AGH	AGL
Alto	AOH	UOC	AOL	UOP	UGC	UGP	AGH	AGL

Tabla 5.2: Listas correspondientes a los estados de alerta y seguridad ordenadas según las prioridades de las tuplas

Una vez se encuentra la posición que ocupa la tupla en la lista correspondiente, se calcula la prioridad según la formula:

$$Prioridad = pos * ((MAXPRIO - MINPRIO)/8) + MINPRIO \quad (5.1)$$

donde MAXPRIO y MINPRIO son dos constantes que determinan la máxima y mínima prioridad que se pueden asignar, y pos es la posición que ocupa en la lista.

Por regla general, se sigue el patrón de dar mayor prioridad a los comandos directos que a las preferencias de usuario. También se establecen con mayor prioridad los comandos de los propietarios frente al de los invitados, tanto si se trata de usuarios como de aplicaciones.

Para una situación normal del entorno y un nivel de seguridad bajo, las acciones de los usuarios, tanto propietarios como invitados, serán las que tengan máxima prioridad. Se entiende que en este estado prima la confianza entre los habitantes del entorno. Esto puede ser debido a que el entorno es un sitio público al que se accede con relativa frecuencia, o un entorno en el que son frecuentes las visitas.

En el caso de un nivel de seguridad normal, la diferencia radica en que las decisiones de prevención que tomen las aplicaciones de seguridad se anteponen a los invitados, de ahí que se le otorgue mayor prioridad a los comandos enviados por las aplicaciones pertenecientes al entorno.

Por último, la lista de nivel de seguridad alto define como comandos más prioritarios aquellos enviados por módulos de seguridad del entorno, por delante de los comandos emitidos por usuarios propietarios o por el resto de aplicaciones del entorno. En este caso, las acciones de usuarios o aplicaciones que no tienen relación con el entorno se les otorga menor prioridad.

Este mecanismo permite realizar cambios fácilmente, y adaptar rápidamente la política general de prioridades a nuevas necesidades simplemente cambiando el orden de las listas.

5.10.3. Políticas particulares de asignación de prioridades

Pueden existir propiedades de recursos que no se adapten al mecanismo de asignación anteriormente descrito. Por ejemplo, porque la prioridad dependiera del valor de la propiedad, lo cual no está contemplado en la política anterior. Así, el volumen de unos altavoces localizados en el entorno tiene que tener en cuenta unas convenciones sociales que no están recogidas en la política genérica. En este caso, y suponiendo que el estado de alerta sea normal, tiene más sentido emplear una política más *educada* para decidir cuál de los posibles valores para el volumen del altavoz es el que se establece. Si dos o más agentes quieren subir o bajar el volumen, las normas sociales, por lo general, determinan que se ponga el volumen más bajo. Para poder reflejar este tipo de comportamiento es necesario añadir un mecanismo que permita asociar políticas definidas *ad-hoc* que sustituyan a la política por defecto en determinadas propiedades.

El diseñador del entorno puede introducir en la definición de la entidad un parámetro denominado *ad-hoc* que se incluye dentro del conjunto de parámetros *policy*. El valor de este parámetro será una cadena de texto con el nombre de la política especial que se aplicará a esa propiedad. Esta política podrá ser escogida dentro de un conjunto predeterminado de políticas especiales. Este conjunto se ampliará a medida que se necesiten nuevas políticas. En el ejemplo de la figura 5.12 se muestra cómo se define la política particular para las propiedades del altavoz volumen izquierdo y volumen derecho. La política se denomina *polite_speaker_volume* y consiste en determinar la prioridad de forma inversamente proporcional al valor que se intente establecer. Así, aquellos agentes que quieran subir el volumen tienen menos prioridad que aquellos que pretenden bajarlo. La sintaxis que se emplea se explicará en el capítulo 6.

```
<class name="speaker">
    <property name="Left_Volume">
        <paramSet name="policy">
            <param name="particular">polite_speaker_volume</param>
        </paramSet>
    <property name="Right_Volume">
        <paramSet name="policy">
            <param name="particular">polite_speaker_volume</param>
        </paramSet>
    </property>
</class>
```

Figura 5.12: Definición de la política de asignación de prioridades de una entidad altavoz empleando XML

Nótese que en el ejemplo se ha utilizado un parámetro de clase, ya que se requiere que la política se emplee para todas las instancias de la clase altavoz. También, se podría haber asociado para un único altavoz.

Actualmente se incluyen tres políticas especiales:

- **Volumen altavoz.** La prioridad es inversamente proporcional al volumen qué se quiera establecer en altavoz.
- **Primar apagar/cerrar.** Si la orden es de apagar (o cerrar) el recurso, está tiene la máxima prioridad. Sería también el caso del altavoz. Si un emisor decide apagarlo completamente tiene más prioridad que los que deciden mantenerlo encendido.
- **Primar encender/abrir.** Éste sería el caso contrario al anterior. Se prima a aquellos comandos que enciendan (o abran) el recurso. Por ejemplo, sería la política a aplicar para el control de la puerta.

Además se han definido dos políticas específicas que afectan a todos los recursos:

- **Estado de emergencia.** Este caso se da cuando ha ocurrido una catastrofe en el entorno. En estos casos, los habitantes requieren el mayor control del entorno ya que su vida puede depender ello. Así, todas los comandos directos tendrán máxima prioridad independientemente de donde vengan. A continuación se primarán las aplicaciones de seguridad. El resto de los casos se les otorgará prioridad nula.
- **Estado de intrusión.** Ocurre cuando existe una intrusión en el entorno. Sólo se atienden los comandos provenientes de propietarios del entorno, tanto usuarios como aplicaciones de seguridad.

Capítulo 6

BBXML: Lenguaje de representación del entorno

6.1. Introducción

En los capítulos previos se ha descrito un modelo de información contextual y una capa intermedia que permite gestionar este modelo. Este capítulo define un lenguaje de representación que permite describir los elementos del modelo. De esta forma se facilita la labor a los desarrolladores de aplicaciones del entorno, ya que permite especificar las características de cualquier entidad —independientemente de su naturaleza— mediante un lenguaje de representación común. El lenguaje se ha definido utilizando la sintaxis que proporciona XML.

Las ventajas de XML como *lingua franca* son:

- Existen numerosas herramientas de libre distribución que permiten reconocer y generar XML.
- Es muy flexible ya que XML no es un lenguaje en sí, sino un conjunto de reglas para crear lenguajes.
- Se puede transformar fácilmente para poder mostrarse en un navegador web.
- Es independiente de la plataforma escogida.
- Es excelente para representar información estructurada pero irregular, gracias a que tiene un tipado débil.

Como desventaja se podría destacar la ineficiencia de XML en cuanto a la relación entre los datos representados y el tamaño de las estructuras de datos utilizadas para representarlos.

El lenguaje definido se ha denominado BBXML. Éste permite describir por un lado las clases de entidades que existen y las propiedades de cada clase, incluyendo un sencillo mecanismo de herencia. Por otro lado, se describen las instancias que inicialmente conforman el entorno y sus relaciones. A partir de la definición de clases e instancias iniciales se provee una herramienta que automáticamente genera

una implementación inicial de la pizarra. También se ha realizado una herramienta que realiza la documentación de la pizarra a partir de la representación anterior.

La última parte del capítulo describe una extensión del lenguaje BBXML que permite añadir información específica para cada aplicación. Esta información se incorpora al modelo BBXML y se almacena también en la pizarra. Así, las diferentes instancias de una aplicación distribuida pueden acceder a la misma configuración común. Se presentan dos ejemplos de interfaces de usuario que se crean automáticamente a partir de la definición que se incorpora a la representación de las entidades. El primero de los ejemplos es un *applet Java* que se despliega en un navegador web y el segundo es una interfaz vocal que emplea lenguaje natural.

6.2. Lenguaje de Representación

El lenguaje de representación de la pizarra se ha denominado BBXML, acrónimo de *Blackboard XML*. Este lenguaje permite definir las posibles clases de entidades que se pueden encontrar en el entorno (véase el apartado 6.2.1). Cada clase se representa mediante un conjunto de propiedades que pueden ser heredadas de otras clases. Una vez definidas las clases disponibles, se pueden especificar las entidades que existen en el entorno (véase el apartado 6.2.2) como instancias de algunas de estas clases. Por último, el lenguaje permite establecer relaciones entre las distintas instancias.

6.2.1. Definición de clases

Cada clase define las propiedades comunes a todas las instancias de la clase. La definición de la clase incluye el nombre de la clase, la clase a la que extiende, la lista de propiedades que conforman la clase y la lista de posibles relaciones que puede tener. En la figura 6.1 se muestra el formato XML que contiene la definición de una clase.

```
xml      = '<?xml version="1.0"?>' classes
classes  = '<classes>' 1*class '</classes>'
class    = bclass *property *relation eclass
bclass   = '<class name="' nombre '" extends="' nombre '">'
property = '<property name="' nombre '" type="' nombre '"/>'
relation = '<relation name="' nombre '"/>'
eclass   = '</class>'
nombre   = identificador
```

Figura 6.1: Esquema de definición de una clase en BBXML

Se ha definido una clase raíz denominada *root* de la cual heredan el resto de las clases (véase la figura 6.2).

```
<?xml version='1.0'?>
<classes>
    <class name="root"/>
</classes>
```

Figura 6.2: Definición en BBXML de la clase *root*

En el apéndice A se describen todas las clases que se proponen en los capítulos 4 y 5. Este conjunto de clases, propiedades y relaciones conforma el esquema básico de conceptos del cual se parte para el diseño de un *entorno inteligente*. Este conjunto se puede extender fácilmente incorporando nuevos dispositivos a partir de los ya existentes. En el ejemplo de la figura 6.3 se describe un nuevo dispositivo denominado *LuzAmbiente*. Éste consta de una lampara regulable y una luz de lectura. Para definir este dispositivo se extiende al dispositivo *Luz*. De esta forma se obtienen las propiedades de *Luz*, *Actuador*, *DispositivoSalida*, *Dispositivo* y *Recurso* tal como muestra la jerarquía de la figura 4.13. Como la *LuzAmbiente* tiene además una luz regulable que no está contemplada en las anteriores, se añaden dos propiedades *estadoVariable* y *valorVariable*. La primera permite consultar el estado de la luz regulable, y la segunda cambiarlo. Ambos son un número entero entre 0-100 que indica el porcentaje de luminosidad de la lampara.

```
<?xml version='1.0'?>
<classes>
    <!-- Define una Luz Ambiente
     -->
    <class name="LuzAmbiente" extends="Luz">
        <property name="estadoVariable" type="int"/>
        <property name="valorVariable" type="int"/>
    </class>
</classes>
```

Figura 6.3: Definición en BBXML de la clase *LuzRegulable* en función de la clase *Luz*

6.2.2. Definición de instancias

Cada instancia se corresponderá con una entidad que inicialmente se encuentre en el entorno. A partir de las instancias definidas, una herramienta crea una implementación de la pizarra, la cual provee de mecanismos que permiten añadir o borrar nuevas instancias modificando la definición inicial (véase el apartado 5.5).

La definición de cada instancia incluye la clase a la que pertenece y la lista de relaciones iniciales que tiene establecidas. El formato que describe la instancia se muestra en la figura 6.4.

```
xml       = '<?xml version="1.0"?>' instances
instances = '<instances>' 1*inst '</instances>'
inst      = bentity *relation eentity
bentity   = '<entity name="' nombre '" type="' nombre '">'
relation  = '<relation name="' nombre ' destination="' nombre '"/>'
eentity   = '</entity>'
nombre    = identificador
```

Figura 6.4: Esquema de definición de una instancia en BBXML

En el ejemplo de la figura 6.5 se muestra la definición del laboratorio B-403 y se informa de la existencia de una lámpara regulable en dicho laboratorio:

```
<?xml version='1.0'?>
<instances>
    <entity name="lab_B403" type="Habitacion">
        <relation name="contiene" destination="luzambiente1" />
    </entity>
    <entity name="luzambiente1" type="LuzAmbiente">
        <relation name="estaEn" destination="lab_B403" />
    </entity>
</instances>
```

Figura 6.5: Ejemplo de definición de instancias usando BBXML

6.2.3. Extensión del lenguaje para información propietaria

Las propiedades de las entidades que se describen con el lenguaje BBXML son aquellas que describen las características intrínsecas de la entidad. Es decir, aquellas características que definen la naturaleza de la entidad. Por otro lado, es necesario incluir un mecanismo que permita extender la definición de entidad añadiendo información extrínseca a ésta. Por ejemplo, información usada por la capa intermedia para la política de permisos (véase el apartado 5.10.2).

Esta información se añade en forma de pares atributo-valor que se denominan parámetros (*param*). A su vez los parámetros se agrupan en conjuntos de parámetros (*paramsets*), de forma que cada aplicación puede definir sus propios parámetros sin interferir con los de otras aplicaciones. Cada parámetro tiene un nombre que es único dentro de cada conjunto.

Los conjuntos de parámetros se pueden definir asociados a una clase o a una entidad, y dentro de cada ella se pueden referir a toda la clase/entidad o sólo a una propiedad en particular. En la figura 6.6 se muestra cómo queda la definición de una clase cuando se le añaden uno o más conjuntos de parámetros.

```
xml         = '<?xml version="1.0"?>' class
class       = bclass 1*prop *paramSet eclass
bclass      = '<class>'
prop        = bprop 1*paramSet eprop
bprop       = '<property name="' nombre '">'
paramSet    = bparamSet 1*defparam eparamSet
bparamSet   = '<paramSet name="' nombre '">'
defparam    = '<param name="' nombre '"/>'
eparam      = '</paramSet>'
eprop       = '</property>'
eclass      = '</class>'
nombre      = identificador
```

Figura 6.6: Esquema de definición de clase añadiendo uno o más conjuntos de parámetros

6.2.4. Generando la pizarra

Se han desarrollado una serie de herramientas que facilitan la reutilización de las clases y la distribución de la especificación de las entidades en el entorno. Para el diseño de las herramientas se han considerado los diferentes tipos de desarrollador que pueden estar trabajando en el entorno. En este sentido se distinguen tres roles:

- **Diseñador del recurso.** Se encarga de desarrollar los recursos que luego formarán parte del entorno, como por ejemplo los dispositivos. El diseñador de recursos aporta por cada tipo de recurso una definición de clase.
- **Administrador del entorno.** Su responsabilidad es configurar los diferentes recursos del entorno. Por cada recurso de que disponga tendrá que definir una instancia y establecer las relaciones necesarias con el resto de los recursos.
- **Diseñador de aplicaciones.** Los diseñadores de aplicaciones se encargan de aportar funcionalidad al entorno. Para ello pueden requerir añadir información propietaria a las entidades del entorno, en cuyo caso, para cada entidad o clase de entidades deberá definir los parámetros necesarios.

Además, para cada uno de los roles se supone que puede existir más de un desarrollador: por ejemplo podría haber un diseñador de recurso por cada recurso.

Cada uno de los desarrolladores genera un fichero BBXML donde define la información que necesita. Para generar la pizarra se crea un único fichero a partir de los anteriores. Este fichero se utilizará como entrada de la herramienta *bbbuilder*, cuya salida es la creación de las entidades del entorno en la representación interna de la pizarra. Para crear este fichero único de descripción del entorno se proveen varias herramientas que se encargan de combinar los diferentes ficheros.

En la figura 6.7 se expone cómo se realiza el flujo completo de desarrollo[1]. A continuación se detalla este proceso.

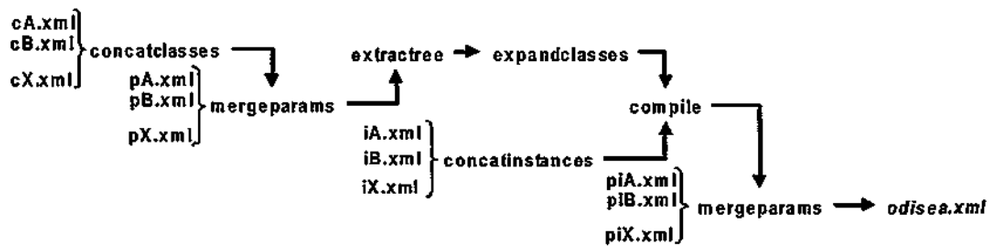


Figura 6.7: Esquema del flujo de desarrollo

Primer paso: Generación de un archivo de clases

Cada desarrollador de recursos provee de un archivo de recurso en el cual añade la definición y relación jerárquica de los recursos que implementa. Hay que tener en cuenta que ya existen definiciones de clase provistas por el propio sistema, y que se pueden reutilizar las definiciones que ya existieran de otros desarrolladores. De esta forma cada desarrollador sólo debe aportar aquellas definiciones de clase nuevas. Por ejemplo, el desarrollador de una lámpara que tuviera dos tipos de luz, una regulable y otra de lectura, aportaría junto con el dispositivo físico, el fichero donde se describe la nueva clase de lámpara.

A partir de los archivos apartados por cada desarrollador, se genera un único archivo (*classes-raw.aux.xml*) con todas las clases mediante la herramienta *concatclasses*.

Por otro lado, los desarrolladores de aplicaciones, a su vez, pueden aportar por cada clase o conjunto de clases un archivo con los parámetros que requieran añadir. Normalmente las aplicaciones necesitarán modificar sólo algunas de las clases del entorno.

Utilizando la herramienta *mergeparams* se fusionan las clases del archivo (*classes-raw.aux.xml*) con los parámetros requeridos por cada aplicación dando lugar al archivo *classes.aux.xml*. De este archivo se extrae la jerarquía de clases utilizando la aplicación *extracttree*. Esta jerarquía junto con el archivo *classes.aux.xml* permite generar un único archivo *classes.xml* que contiene la definición completa de todas clases del entorno.

[1]La ejecución de todos los pasos de este proceso se automatizan mediante un *script*

Segundo paso: Generación de instancias de clases

La definición de las clases, como se ha podido comprobar, es realizada por los desarrolladores de dispositivos y de *aplicaciones sensibles al contexto*, de forma independiente. Las instancias, en cambio, son definidas por el instalador del entorno inteligente concreto, que decide cuántas lámparas, altavoces, interruptores, etc. se instalan en el entorno.

Para ello, al igual que se realiza con las clases, se recopilan todas las instancias en un único archivo (*instances.aux.xml*). Este paso lo realiza la herramienta *concatinstances*.

A partir de la definición de las clases (*classes.xml*) y de la declaración de las instancias (*instances.aux.xml*) se obtiene mediante la herramienta *compile* un nuevo archivo dónde a cada instancia se le ha añadido la información de clase. Finalmente, se agregan los parámetros particulares de cada instancia suministrados por los diseñadores de aplicaciones utilizando de nuevo la herramienta *mergeparams*, y dando como resultado el archivo *blackboard.xml*

Tercer paso: Generación de la pizarra

El archivo *blackboard.xml* contiene toda la información relativa a las entidades que conforman inicialmente el entorno. Cada entidad posee tanto la información definida en la clase como la definida en la instancia. Ésta incluye propiedades, relaciones con otras entidades, y los parámetros específicos asociados tanto a la entidad como a cada propiedad en particular. El formato XML que sigue una entidad se define en la figura 6.8. Éste es el mismo formato que se emplea cuando se recupera la información de un nodo de la pizarra mediante la operación *GetContext* (véase el apartado 5.6.3).

```
xml       = '<?xml version="1.0"?>' entity
entity    = bentity *prop *relation *paramSet eentity
bentity   = '<entity name="' nombre '" type="' nombre '">'
prop      = bprop *paramSet eprop
bprop     = '<property name="' nombre '">'
paramSet  = bparamSet 1*defparam eparamSet
bparamSet = '<paramSet name="' nombre '">'
defparam  = '<param name="' nombre '"/>'
eparam    = '</paramSet>'
eprop     = '</property>'
relation  = '<relation name="' nombre ' destination="' nombre '"/>'
eentity   = '</entity>'
nombre    = identificador
```

Figura 6.8: Esquema de definición de una entidad en BBXML

El archivo *blackboard.xml* se emplea como entrada de la herramienta *builder* que se encarga de transformar la representación XML en el formato interno que

emplea la pizarra. Hay que tener en cuenta que este archivo contiene únicamente la información estática del entorno, aquella que establece la configuración inicial del mismo. Mediante la herramienta de construcción se permite recuperar un estado conocido. Por otro lado, la capa intermedia provee de operaciones que permiten agregar y eliminar entidades y modificar los valores de las propiedades de forma dinámica (véase el apartado 5.5).

6.3. Interfaces de usuarios dinámicamente generados

Un área de investigación necesaria dentro de la computación ubicua es el diseño de interfaces de usuario que se adapten a los cambios del entorno. En esta sección se presentan la principales ideas de cómo el lenguaje BBXML se adapta para definir interfaces de usuario de distinta naturaleza [134]. Se han implementado dos ejemplos.

La primera de la dos interfaces de usuario desarrolladas ha sido una interfaz basada en web que permite controlar los dispositivos del entorno. Esta interfaz ha recibido el nombre de *Jeoffrey*. La segunda interfaz se denomina *Odisea*, y consiste en un gestor de diálogos que interacciona con el entorno mediante lenguaje natural. Ambas interfaces son generadas dinámicamente en función de la información descrita mediante BBXML sobre las entidades almacenadas en la pizarra [14].

Ambas han sido implementadas por desarrolladores independientes, la primera como parte de una beca asociada al proyecto Interact, y la segunda como tesis doctoral, defendida en el 2005 [182].

6.3.1. *Jeoffrey*

Jeoffrey ha sido diseñado pensando en un entorno enfocado al hogar. *Jeoffrey* despliega una vista parcial de la información contextual almacenada en la pizarra, ya que únicamente muestra los dispositivos y su relación con el entorno en el cual están ubicados. Cada entidad que representa a un dispositivo, además de la información propia de la entidad, posee un conjunto de parámetros que se emplean en la creación de la interfaz gráfica.

La interfaz se compone de tres partes estructuradas jerárquicamente:

- **Ventana principal.** Es la ventana principal de la aplicación. Despliega una lista que contiene las habitaciones que conforman el entorno.
- **Ventana de habitación.** Cuando el usuario elige una habitación en la ventana principal, se despliega una ventana que muestra un mapa de la habitación. Este mapa incluye la disposición física de los dispositivos y del mobiliario. El mapa se construye mediante la superposición de un fondo fijo que representa a la habitación y de las imágenes individuales que conforman la aplicación. Este mapa se genera dinámicamente cada vez que se inicia la interfaz.

- **Panel de control.** Cada dispositivo contiene un panel de control que se muestra cuando el usuario pulsa en la imagen que representa el dispositivo. Este panel permite interaccionar con el dispositivo físico.

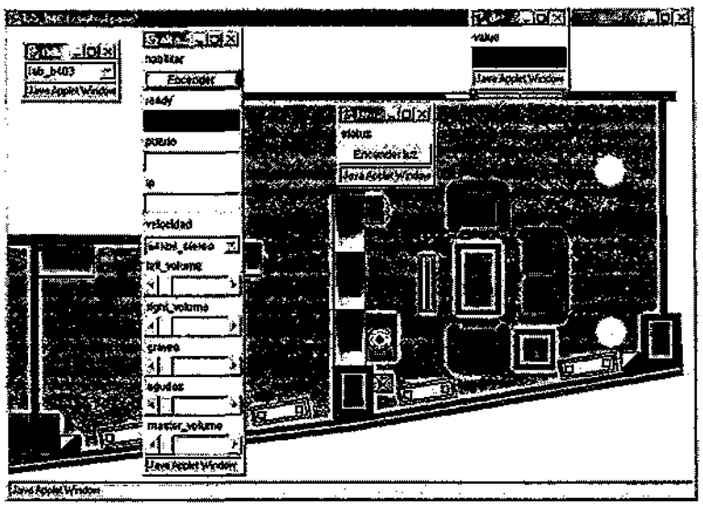


Figura 6.9: Interfaz de usuario desplegada por *Jeoffrey*

La figura 6.9 muestra la interfaz de usuario. La ventana más a la izquierda es la ventana principal. La ventana que se encuentra en el fondo de la imagen corresponde a la ventana de la habitación seleccionada. Finalmente, la tercera ventana que aparece es un panel de control de uno de los dispositivos del entorno.

Jeoffrey se invoca desde un navegador web, y la interfaz se genera dinámicamente en función de la información almacenada en la pizarra. Esto ocurre cada vez que se carga la interfaz. Si se producen cambios en la configuración del entorno (ej. un dispositivo que aparece o desaparece), quedan reflejados en la pizarra se transmiten automáticamente cada vez que se arranca un nuevo interfaz.

Creación de la ventana principal

El grafo de la pizarra incluye una entidad por cada habitación de tal forma que *Jeoffrey* puede preguntar por todas las habitaciones que contiene la casa. Con la referencia de cada habitación construye una lista desplegable que permite al usuario seleccionar la habitación que quiere gestionar.

Creación de la ventana de habitación

La interfaz de la ventana de la habitación también se crea dinámicamente. Por cada dispositivo localizado en la habitación existirá una relación de tipo *contiene* entre la habitación y el dispositivo. Una vez seleccionada la habitación, *Jeoffrey* puede extraer toda la información de los dispositivos. En la descripción de cada

instancia se ha empleado un conjunto de parámetros denominado *Jeoffrey* que añade a la entidad parámetros propios que se emplean para mostrar la representación gráfica de la entidad.

De esta forma, cada entidad habitación tiene conjunto de parámetros *Jeoffrey* (véase el ejemplo 6.10). El primer parámetro (*background*) indica el nombre del archivo que se emplea como fondo del mapa. Este archivo contiene un plano de la habitación que incluye las partes que no están representadas en la pizarra, como el mobiliario. Los otros dos parámetros indican el tamaño del mapa a desplegar (*width* y *height*).

```
<entity name="lab_B403" type="room">
    <paramSet name="jeoffrey">
        <param name="background">fondof.jpg</param>
        <param name="width">764</param>
        <param name="height">513</param>
    </paramSet>
</entity>
```

Figura 6.10: Ejemplo de un conjunto de parámetros *Jeoffrey* para la definición de una entidad de tipo Habitacion

Además cada dispositivo incluye también un conjunto de parámetros *Jeoffrey*. Esta información puede estar asociada a todo el dispositivo o a una de sus propiedades. Para generar la ventana de la habitación se emplean los parámetros asociados a toda la entidad que son los que indican qué imagen es la que representa al dispositivo y cuáles son las coordenadas donde se va a dibujar en el mapa. En el ejemplo 6.11 se muestra el conjunto de parámetros correspondientes a un fluorescente de la habitación. Este fluorescente se representa con la imagen del archivo *fluoresecente.gif* y se va a dibujar en las coordenadas (460, 247).

```
<entity name="Lamp_1" type="fluorescent">
    <paramSet name="jeoffrey">
        <param name="image">fluorescente.gif</param>
        <param name="x">460</param>
        <param name="y">247</param>
    </paramSet>
</entity>
```

Figura 6.11: Ejemplo de un conjunto de parámetros *Jeoffrey* para la definición de una entidad de tipo Luz

Cuando se selecciona la habitación, *Jeoffrey* recupera de la pizarra la entidad que representa la habitación y la información propietaria asociada. De esta forma

es capaz de pintar el fondo del mapa. A continuación obtiene todos los dispositivos de esa habitación y los dibuja en el mapa según los parámetros establecidos en cada dispositivo.

Creación del panel de control

El segundo conjunto de parámetros *Jeoffrey* que aparece en la descripción de un dispositivo se emplea para generar el panel de control. Hay un conjunto de parámetros por cada propiedad del dispositivo, que define las características del control gráfico. Se definen seis tipos genéricos de control gráfico:

- **Áreas de texto.** Permiten cambiar un valor que se expresa en forma de cadena de caracteres.
- **Botones.** Despliegan un botón que cada vez que se pulsa envía un valor fijo.
- **Conmutadores.** Son botones que envían como posibles valores encendido y apagado. Cada vez que se pulsan envían un valor alternativo.
- **Barras de desplazamiento.** Modifican propiedades que admiten valores dentro de un intervalo.
- **Listas desplegables.** Definen la lista de valores que puede tomar la propiedad, y sobre la cual el usuario puede elegir uno.
- **Alarmas.** Son etiquetas que cambian de color según el valor de una propiedad de la pizarra. Por ejemplo, una alarma puede ser el estado de la puerta que se refleja en una etiqueta negra si está cerrada o rojo si está abierta.

Tal como se aprecia en la tabla 6.3.1 cada tipo de control gráfico tiene un conjunto de parámetros distintos.

Se definen dos parámetros que son comunes a todos los controles. Estos parámetros permiten establecer dependencias entre los controles de tal forma que se puede indicar que un control se encuentre deshabilitado hasta que se establezca un determinado valor en otro control.

- **Dependences.** El valor de este parámetro es un identificador que se emplea para referenciar a la propiedad.
- **Enable.** Este parámetro indica la propiedad de la cual depende y el valor que tiene que tomar la propiedad para que el control se habilite. Para cualquier otro valor el control se deshabilita.

El ejemplo de la figura 6.12 muestra cómo se emplean los parámetros *dependences* y *enable*. En el ejemplo, la propiedad *Puerto* estará habilitada cuando la propiedad *Habilitar* tenga el valor 0. Para ello, a la propiedad *habilitar* se le asigna

Tipo de control	Parámetro	Descripción
Conmutadores	text_off	Texto del botón cuando el valor de la propiedad es apagado.
	cmd_off	Valor que se establece en la propiedad cuando se pulsa el botón en estado de apagado.
	text_on	Texto del botón cuando el valor de la propiedad es encendido.
	cmd_on	Valor que se establece en la propiedad cuando se pulsa el botón en estado de encendido.
	color_on	Color del botón encendido.
Botones	text	Texto del botón
	cmd	Valor que se establece en la propiedad cuando se pulsa el botón
	color	Color del botón
Lista desplegable	entry	Uno o más parámetros por cada entrada de la lista
Barra de desplazamiento	lo	Indica el valor mínimo de la barra
	hi	Indica el valor máximo de la barra
	unit	Indica el incremento
Alarma	entry	Una entrada por cada color de la forma <Valor> <Color_HEX>. Indica qué color establecer para cada valor de la propiedad.

Tabla 6.1: Tipos de control gráfico definidos en *Jeoffrey*

```
<!-- Entidad Altavoz 1-->
<entity name="Altavoz">
    <property name="Habilitar">
        <paramSet name="jeoffrey">
            <param name="dependences">1</param>
                ...
        </paramSet>
    </property>
            ...
    <property name="Puerto">
        <paramSet name="jeoffrey">
            <param name="enable">1.0</param>
            <param name="type">entry</param>
        </paramSet>
    </property>
            ...
 </entity>
```

Figura 6.12: Ejemplo de uso de los parámetros *dependences* y *enable*

el identificador 1 mediante el parámetro *dependences*[2]. Por otro lado, a la propiedad *Puerto* se le añade el parámetro *enable*[3]. Este parámetro sigue el formato <identificador>.<valor>, en el ejemplo el valor es de *1.0*. Esto indica que la propiedad *Puerto* estará habilitada cuando la propiedad cuyo parámetro *dependences* coincida con 1 —en este caso la propiedad *habilitar*— tenga el valor 0.

En la figura 6.13 se ha ampliado el ejemplo mostrado en la figura 6.11 añadiendo la propiedad *status* y los parámetros correspondientes a esta propiedad.

Se observa que se le ha asociado un conmutador como control[4], y se ha configurado la presentación del componente empleando los parámetros definidos en la tabla 6.3.1 para el control del conmutador. Éstos permiten modificar las siguientes características del botón:

- **Texto del botón**. Este texto cambia según el estado de la propiedad. El parámetro texto apagado[5] establece el mensaje que se muestra en el botón cuando la luz está apagada, mientras que el parámetro texto encendido[6] aparece cuando la luz está encendida.

- **Color del botón**. Este parámetro establece el color del botón cuando el fluorescente está encendido. Por defecto, si está apagado el color es gris. El color se codifica mediante tres números hexadecimales de 0 a FF precedidos

[2] <param name="dependences">1</param>
[3] <param name="enable">1.0</param>
[4] <param name="type">*switch*</param>
[5] <param name="text_off">Encender</param>
[6] <param name="text_on">Apagar</param>

```
<entity name="Lamp_1" type="fluorescent">
    <property name="Status">
        <paramSet name="jeoffrey">
            <param name="type">switch</param>
            <param name="text_off">Encender</param>
            <param name="text_on">Apagar</param>
            <param name="cmd_on">0</param>
            <param name="cmd_off">1</param>
            <param name="color_on">0x00FF00</param>
        </paramSet>
    </property>
    <paramSet name="jeoffrey">
        <param name="image">reflectante.gif</param>
        <param name="x">460</param>
        <param name="y">247</param>
    </paramSet>
</entity>
```

Figura 6.13: Ejemplo completo de definición de un entidad añadiendo el conjunto de parámetro *Jeoffrey*

del prefijo *0x*. El primer número indica la cantidad de color rojo, el segundo de color verde, y el último de color azul.

Finalmente, los parámetros *cmd_apagado* y *cmd_encendido* definen los valores que se establecen en la propiedad estado cuando se presiona el botón.

La figura 6.14 ilustra cómo queda el panel de control de fluorescentes tanto cuando está encendido como apagado. La imagen más a la derecha de la figura es la que se utiliza para representar el fluorescente en el mapa de la habitación. Esta imagen se obtiene a partir del parámetro imagen[7].

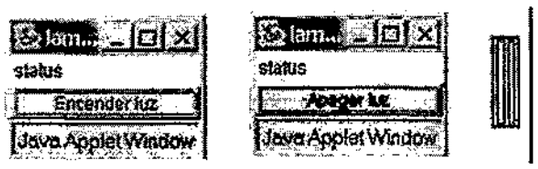


Figura 6.14: Panel de control para el dispositivo fluorescente

[7] <param name="image">reflectante.gif</param>

6.3.2. *Odisea*

La interacción mediante el habla es uno de los mecanismos preferidos por el ser humano. La construcción de entornos de *Inteligencia Ambiental* necesariamente tiene que contemplar tal tipo de interfaz. Por este motivo, se ha aplicado el lenguaje BBXML para generar una interfaz vocal que permite interaccionar con las entidades del entorno. Los diferentes diálogos disponibles se crean automáticamente a partir de la información de la pizarra, y se describen mediante dicho lenguaje [183].

La filosofía de diseño es similar a la que se describió con *Jeoffrey*. De hecho, parte de la información —la relativa a las propiedades de cada dispositivo— es compartida entre ambas interfaces. Cada dialogo está asociado a una entidad, de tal forma que cuando una entidad se añade a la pizarra, se crea un nuevo dialogo que permite interaccionar con ella. Además, para cada entidad se puede personalizar su diálogo, de tal forma que se puedan distinguir una entidad del resto del mismo tipo.

Para cada entidad se define un conjunto inicial de partes lingüísticas que tratan de cubrir todas las posibles formas de interacción entre la entidad y el usuario. Este conjunto está formado por:

- **Parte Verbal (VP).** Corresponde a las acciones que el usuario puede realizar sobre la entidad.
- **Parte Objeto (OP).** Incluye la lista de nombre que el usuario puede asociar a la entidad.
- **Parte Ubicación (LP).** Especifica su localización dentro del entorno
- **Parte Modal (MODALP).** Indica el modo en que se debe realizar la acción.
- **Parte Cuantificadora (QP).** Define una cantidad que se aplica sobre la acción a realizar.
- **Parte Modificadora (MP).** Añade información calificativa a alguna de las partes anteriores.

Definición de un diálogo para una clase

La información lingüística se asocia a cada entidad mediante un conjunto de parámetros denominado *dialogue*. Por cada una de las posibles acciones se define un parámetro que toma como nombre *actionI* donde *I* es un número que permite distinguir las posibles acciones. Por cada parámetro *action* se definen uno o más parámetros *skeletonIJ* que definen los diferentes esqueletos de las oraciones que se pueden utilizar para invocar la acción, donde *IJ* corresponde al esqueleto j-ésimo de la acción i-ésima. El esquema completo de una clase una vez agregados los parámetros *dialogue* se muestra en la figura 6.15

Cada parámetro acción identifica cada una de las acciones que se pueden realizar sobre la propiedad, mientras que cada esqueleto describe cada de una de las

```
classes     = bclasses 1*class eclasses
bclasses    = <classes>
class       = bclass 1*property eclass
bclass      = '<class name="' nombre '" extends=' nombre '">'
property    = bproperty paramSet eproperty
bproperty   = '<property name="' nombre '">'
paramSet    = bparamSet 1*params eparamSet
bparamSet   = '<paramSet name="dialogue">'
params      = paramaccion 1*paramskel
paramaccion = '<param name="' accion '">' 1*uchar '</param>'
accion      = 'action' digits
paramskel   = '<param name="' skeleton '"> 1*parte '</param>'
parte       = mod 1*palabra
palabra     = 1*uchar
mod         = ('VP' | 'OP' | 'LP' | 'IOP' | 'MODALP' | 'QP' | 'MP')
skeleton    = 'skeleton' digits
eparamSet   = '</paramSet>'
eproperty   = '</property>'
eclass      = '</class>'
eclasses    = '</classes>'
nombre      = identificador
```

Figura 6.15: Ejemplo de definición de una clase genérica añadiendo el conjunto de parámetro *dialogue*

posibles oraciones que se pueden utilizar para designar la acción. Cada oración se divide en una o más partes lingüísticas empleando las siguientes abreviaturas (VP, OP, LP, IOP, MODALP, QP ó MP). A continuación de cada abreviatura se especifican las palabras que pueden aparecer en esa parte de la oración.

En el figura 6.16 se muestra la definición de las acciones que admite el gestor de diálogo para todos los dispositivos de tipo (*light*). Todos los dispositivos de tipo *Luz* tienen una propiedad denominada *status* que refleja cómo se encuentra la luz. Se definen sobre esa propiedad dos acciones: encender y apagar. La acción de encender tiene asociado el esqueleto *skeleton11* que especifica que la oración que invoca la acción tiene tres partes: Parte verbal (VP), Parte Objeto y Parte Modal. Para cada una de las partes se especifican los sinónimos que pueden aparecer para referirse a esa parte. Por ejemplo, en la parte verbal de la oración puede aparecer el verbo encender o el verbo poner. Hay que resaltar que el gestor de diálogos admite tanto el infinitivo como cualquiera de sus conjugaciones. En el caso de la Parte Objeto se puede emplear o la palabra *luz* o la palabra *lámpara* (de nuevo se admiten todas las posible formas, luz, luces, lámpara, lámparas). En función de este esqueleto frases como *pon la luz,encenderías la lámpara, enciende las luces...* serían válidas para establecer el valor de encendido en la propiedad de *status*. La última parte del esqueleto (Parte Modificadora), es optativa y permite añadir calificadores a las oraciones anteriores que ayuden al reconocimiento, así las frases *pon la luz de ambiente* ó *enciende la lámpara halógena* también serían válidas.

```
<classes>
<class name="Light" extends="device">
    <property name="status">
        <paramSet name="dialogue">
            <param name="action1">encender</param>
            <param name="skeleton11">VP encender poner
                                     OP luz lampara
                                     MP ambiente halógena</param>
            <param name="action2">apagar</param>
            <param name="skeleton21">VP apagar quitar
                                     OP luz lampara
                                     MP ambiente halógena</param>
        </paramSet>
    </property>
</class>
</classes>
```

Figura 6.16: Ejemplo de clase tipo *Luz* una vez añadidos el conjunto de parámetros *dialogue*

Definición de un diálogo para una instancia

El ejemplo anterior describe cómo definir la interacción hablada con todas las entidades de tipo *Light* del entorno. Ahora bien, se necesita un mecanismo que permita particularizar la interacción para cada entidad de forma que el usuario puede referirse a una entidad de tipo *Light* en concreto. Para ello se definen los parámetros del tipo *add_(all|ij)*. El parámetro *add_all* añade nueva información a todos los esqueletos definidos para la propiedad, mientras que los parámetros *add_ij* se refieren a un esqueleto en concreto (*skeletonIJ*). Estos parámetros se adjuntan con la definición de la entidad agrupados por un conjunto de parámetros *dialogue* tal como se indica en la figura 6.17

```
instances   = binstances 1*entity einstances
binstances  = '<instances>'
entity      = bentity 1*property eentity
bentity     = '<entity name="' nombre ' type="' nombre '">'
property    = bproperty paramSet eproperty
bproperty   = '<property name="' nombre '">'
paramSet    = bparamSet 1*params eparamSet
bparamSet   = '<paramSet name="dialogue">'
params      = paramall | 1*paramskel
paramall    = '<param name="add_all">' parte '</param>'
paramskel   = '<param name="' add_ij '"> parte '</param>'
add_ij      = 'add' digits
eparamSet   = '</paramSet>'
eproperty   = '</property>'
eentity     = '</entity>'
einstances  = '</instances>'
nombre      = identificador
```

Figura 6.17: Definición en BBXML de un diálogo para una instancia

De esta forma si en el entorno existen dos luces, una en el techo y otra en la pared, se añadiría a todos los esqueletos una *Parte de Ubicación* a cada una de ellas dónde se especificaría. Así, para la luz del techo habría que añadir *LP techo* mientras que para la otra luz sería *LP pared*. En la figura 6.18 se muestra la definición completa de la entidad que representa a la luz de techo.

En la tesis *Estudio e integración de un sistema de diálogos dinámico en un entorno inteligente* de Germán Montoro [182] puede encontrarse una descripción mucho más detallada de la interfaz oral ODISEA, y de su integración en la arquitectura propuesta en esta tesis.

```
<instances>
<entity name="lamp1" type="light">
    <property name="status">
        <paramSet name="dialogue">
            <param name="add_all">LP techo</param>
        </paramSet>
    </property>
</entity>
</instances>
```

Figura 6.18: Ejemplo de definición de un diálogo para una instancia de tipo *Luz*

6.4. Ejemplo de proceso de desarrollo

El lenguaje propuesto permite un desarrollo modular y factible de las aplicaciones de *Inteligencia Ambiental*. Esto se puede apreciar en el siguiente ejemplo concreto de desarrollo.

Primeramente, se parte de las definiciones básicas de habitaciones y dispositivos más habituales que proporciona esta tesis (véase el apéndice A). Un desarrollador independiente diseña una luz de ambiente y proporciona el fichero de clase que contiene información sobre las propiedades y relaciones del dispositivo. Dado que una luz de ambiente es una clase particular del dispositivo *Luz*, la mayor parte de la información necesaria para definir la luz de ambiente se obtendrá de la definición de la clase *Luz*.

Por otro lado, dos desarrolladores independientes realizan una aplicación de control web y una interfaz oral respectivamente. Cada uno proporcionan los ficheros de clase que permiten parametrizar la luz de ambiente, de modo que se pueda integrar en ambas interfaces.

Un instalador despliega una infraestructura EIB con una serie de dispositivos (interruptores, luces de ambiente, sensores...) para cada dispositivo genera un fichero de instancia donde define las propiedades de cada dispositivo particulares a la instalación. Además, proporciona para cada interfaz un fichero de instancia. De esta forma, el instalador puede particularizar aspectos concretos de la interfaz como por ejemplo el nombre propio de cada luz, la posición dentro de la habitación...)

Finalmente, se genera la pizarra a partir de todos los ficheros provistos, y el entorno automáticamente dispone de una interfaz web y otra interfaz oral que permiten controlar los dispositivos EIB instalados en el entorno.

PARTE III

Resultados, conclusiones y trabajos futuros

Capítulo 7

Demostradores

7.1. Introducción

En la visión general de un *entorno inteligente* que se exponía en la introducción (véase el apartado 3.1) se distinguieron las siguientes capas: la capa física, que se encarga de la interacción con el entorno físico; la capa intermedia o de contexto; y la capa de aplicación, donde está incluida la interfaz con el usuario. En los capítulos previos se han descrito las ideas fundamentales e implementación de la capa intermedia. Para probar la funcionalidad de la capa de contexto se ha integrado la arquitectura en un entorno real. En este capítulo se van a presentar el entorno experimental (véase el apartado 7.3) y los demostradores desarrollados como validación de la propuesta de esta tesis (véase el apartado 7.5). La capa física consiste en dos redes de comunicación y un conjunto heterogéneo de dispositivos. La capa de aplicación consta de una serie de aplicaciones distribuidas construidas sobre las bases de diseño expuestas en los capítulos previos. Las interacciones entre los componentes de estas dos capas se armonizan gracias a la capa de contexto.

En la fase del diseño del sistema *Interact* se pensaron dos escenarios, un entorno de oficina y un entorno orientado al hogar, en los que las necesidades del usuario son sensiblemente distintas, para probar que el método de diseño y la arquitectura del sistema fueron lo suficientemente flexibles como para ser reutilizados. Así, el entorno físico que constituye el laboratorio queda ha dividido en dos áreas (véase el apartado 4.6): el salón y la oficina. En esta última localización se guardan los ordenadores personales necesarios para tareas pesadas de computación y el control de los diferentes dispositivos.

Se han desarrollado una serie de demostradores para comprobar la idoneidad del enfoque propuesto en esta tesis. Estos demostradores varían desde simples pruebas de concepto a sistemas completos. Para el desarrollo de cada demostrador se ha seguido un modelo de programación común. Éste se basa en el esquema de interacción que se explica en el apartado 5.4, y que queda resumido en la figura 5.1. Este esquema permite desarrollar cada demostrador como un conjunto de agentes diseñados e implementados de forma independiente.

7.2. Notación

Durante este capítulo se van utilizar diagramas de secuencia en UML para representar gráficamente las operaciones que realizan los diversos agentes que conforman una aplicación sobre la pizarra. Para que la exposición de los diagramas sea más clara se va a seguir un notación específica que permita asociar a cada operación de la pizarra una descripción textual.

Así, la operación de *GetContext* que permite obtener información almacenada en la pizarra se representa por un mensaje *obtiene: <info>*, donde <info> puede ser el nombre de una entidad o el valor de una propiedad. En el primer caso la operación equivaldría a obtener toda la información de la entidad incluyendo el valor de sus propiedades. En el segundo se accedería únicamente al valor de la propiedad.

La operación *SetContext* se representa por un mensaje *cambia: <valor>*, donde <valor> indica una entidad y un nombre de propiedad.

Las operaciones de *AddContext* y *RemoveContext* se transforman en *añade: <entidad>* y *borra: <entidad>* respectivamente.

De forma similar las operaciones de *AddRelation* y *RemoveRelation* se representan con *añade: rel <relacion> <relacion>→<relacion>* y *borra: rel <relacion> <relacion>→<relacion>* respectivamente.

Los eventos que se reciben se transcriben con *evento: <operacion>*, donde <operacion> hace referencia a la última operación que se ha producido.

7.3. Entorno experimental

El componente principal del entorno experimental consiste en un laboratorio de investigación que ha sido amueblado como el salón de una casa (véase la figura 7.1). Este salón ha sido equipado con un conjunto heterogéneo de dispositivos y máquinas (véase el apartado 7.3.3). Tal como se expuso en el capítulo 3, existen múltiples opciones a la hora de elegir la red o redes que van a interconectar los distintos dispositivos. Se ha optado por realizar la separación lógica que se propone en ese capítulo.

Por un lado, el entorno consta de una red de control que permite distribuir información con tasa binaria reducida. Esta red permite conectar los sensores (presencia, temperatura, luminosidad...) y actuadores (conmutadores, termostatos...) del entorno, así como cualquier dispositivo que requiera recibir comandos de control (véase el apartado 3.2.2). La red de control hay que entenderla desde un punto de vista lógico, ya que en la implementación se emplearón diversas tecnologías.

Por otro lado, se dispone de una red multimedia que permite distribuir el tráfico audio-visual. A esta red se conectan dispositivos capaces de emitir o recibir este tipo de tráfico.

A continuación se explica en detalle la arquitectura de estas dos redes.

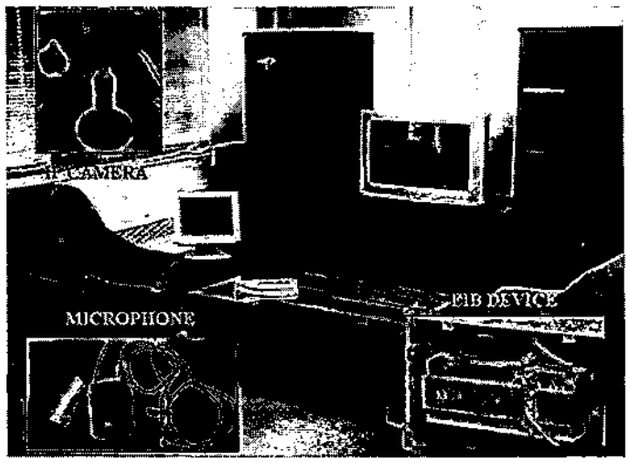


Figura 7.1: Fotografía del entorno físico empleado en el sistema *Interact*

7.3.1. Red de Control

En la implementación de la red de control intervienen tres tipos de tecnologías: el bus EIB, Phidget y Ethernet. El bus EIB constituye la columna vertebral de la red control, y es donde se conectan la mayor parte de actuadores y sensores.

La elección del bus EIB viene motivada por diversos factores:

- Es un bus que está dentro de los posibles candidatos a hacerse con el mercado europeo.
- Gracias a su difusión en Europa es fácil conseguir dispositivos.
- La oferta de dispositivos disponibles es bastante amplia.
- Es un bus bastante versátil, ya que permite poder construir dispositivos propios al tener la interfaz con el bus claramente separada.

EIB constituye el camino de conexión con los actuadores y sensores distribuidos en el entorno. Por un lado permite controlar los distintos dispositivos y mecanismos eléctricos que se conectan al entorno. Se dispone de un componente denominado salida binaria que actúa como un relé controlado por el bus EIB. Se ha habilitado un tipo especial de tomas de corriente que constan de una salida binaria, de forma que se puede controlar el encendido y apagado de los electrodomésticos que se conecten a estas tomas, sin necesidad de modificar el dispositivo en sí. También se emplean las salidas binarias para actuar sobre mecanismos eléctricos, como el portero automático que permite abrir la puerta.

Por otro lado, el bus EIB permite distribuir la información capturada por los sensores. Para ello se ha instalado otro tipo de componente denominado entrada binaria que permite detectar cuándo un circuito eléctrico se encuentra abierto o

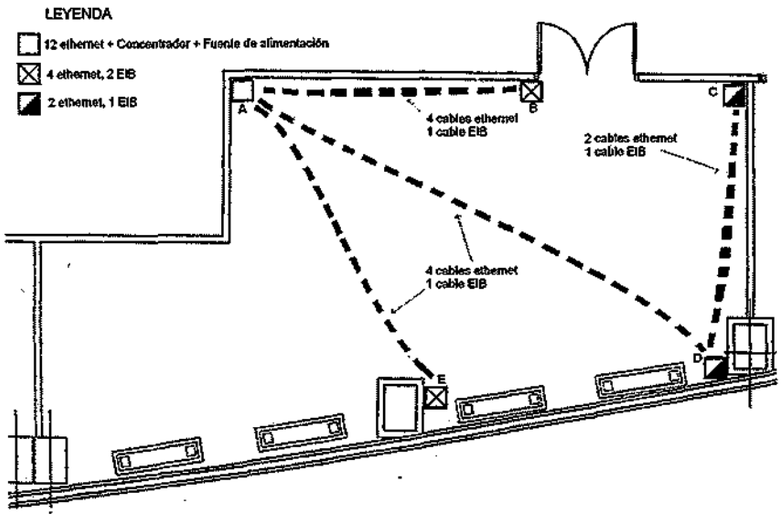


Figura 7.2: Plano del cableado de red del laboratorio de entornos inteligentes

cerrado. Éste es el caso, por ejemplo, del sensor que avisa si la puerta está abierta o cerrada.

En la figura 7.2 se muestra la distribución y número de tomas EIB que se han incorporado al laboratorio.

En una instalación típica de EIB, los componentes se encuentran configurados de manera que cada uno conoce con qué dispositivo tiene que comunicarse. De esta forma, un interruptor EIB conoce a qué luces tiene enviar los comandos de apagado y encendido. Ésto permite que cada dispositivo sea autónomo, de forma que la red es completamente distribuida. Ahora bien, la configuración inicial de los dispositivos es un proceso lento, que requiere manipularlos físicamente, lo que dificulta los cambios en la configuración. Por el contrario, en la configuración que se ha optado en esta tesis todos los dispositivos EIB se comunican a través de la pizarra. Ésto permite una mayor flexibilidad del bus ya que se puede cambiar dinámicamente la funcionalidad de los dispositivos. Por ejemplo, en una configuración estándar un interruptor tendría almacenado qué luces enciende o apaga. Un cambio en esta configuración implica emplear una aplicación para descargar la nueva configuración, deteniendo el dispositivo momentáneamente. En cambio, cuando la comunicación se realiza a través de la pizarra el interruptor no requiere ser reconfigurado. Ahora son los distintos agentes que están interesados en los cambios del interruptor quiénes deciden qué otros dispositivos controla. Así, el interruptor puede pasar de controlar las luces a controlar las persianas sin tener que modificar su configuración. Otra ventaja adicional radica en que se facilita la integración de la red EIB con otros tipos de redes. De esta forma el interruptor puede también controlar otros tipos de dispositivos que no sean EIB, como por ejemplo, el encendido y apagado de una cámara IP.

Además de EIB, también está en la red de control una antena RFID sumi-

nistrada por la empresa Phidget. Esta antena permite detectar quién accede al laboratorio o quién lo abandona. Cada usuario del entorno está provisto de un tarjeta pasiva que dispone de un identificador único. Cuando se acerca la tarjeta a la antena se captura el identificador, de manera que se puede extraer la identidad del usuario.

Finalmente, la red Ethernet que se describe en el siguiente apartado se emplea también para cursar los comandos de control que actúan sobre los dispositivos audio-visuales. Tal es el caso, por ejemplo, de los cambios en el volumen de los altavoces o en la emisora de la radio.

7.3.2. Red Multimedia

Esta red será la que se encargue de transportar el tráfico multimedia. Para implementarla se ha optado por la tecnología Ethernet/100 Mbps (véase el apartado 3.2.2). Varios son los factores claves en esta elección:

- **Soporta tráfico multimedia.** La tecnología Ethernet no dispone de un mecanismo que permita reservar ancho de banda, es más, al resolver el acceso al medio mediante un algoritmo de contienda, las prestaciones de la red se degradan con el aumento del terminales. En un principio este motivo debería ser suficiente para desacreditar la elección de Ethernet como red multimedia. Sin embargo, gracias a las elevadas tasas binarias (hasta 1 Gbps) que puede llegar a soportar Ethernet, se pueden inyectar en la red los canales multimedia necesarios en un hogar digital. A modo de ejemplo de las posibilidades de esta tecnología en el hogar, actualmente ya existen pasarelas que permiten convertir canales de televisión digital en canales multidifusión IP que se distribuyen a través de una Ethernet.

- **Disponibilidad.** La red Ethernet se encuentra ampliamente difundida tanto en el entorno universitario como en el empresarial. Así, resulta fácil encontrar componentes en el mercado.

- **Experiencia.** En relación con el punto anterior, también se puede afirmar que hay bastante experiencia en el despliegue de soluciones con este tipo de redes. La red Ethernet constituye una tecnología madura que ha sido ampliamente probada a lo largo de su existencia.

- **Futuro.** Tal como se mostró en el apartado 3.2.2, el triunfo de la tecnología Ethernet en las redes de área local es una realidad, lo cual lleva a pensar que este éxito probablemente se traslade al mundo residencial.

En la figura 7.2 se muestra la disposición de cableado y las diferentes tomas colocadas en el prototipo. Consta de doce tomas distribuidas en los mismos puntos que las tomas EIB. En la sala de máquinas se ha colocado un conmutador de 24 puertos que distribuye el tráfico entre todas las tomas. Éste se emplea además como pasarela a la red externa del Departamento.

7.3.3. Dispositivos

Cada uno de los dispositivos del entorno se encuentra conectado a una (o a ambas) de las dos redes descritas anteriormente, dependiendo de su naturaleza. Independientemente de la red a la cual estén conectados, el control de los dispositivos se ha uniformizado empleando el protocolo SNMP [174]. La jerarquía de las diferentes redes se puede consultar en la figura 7.3

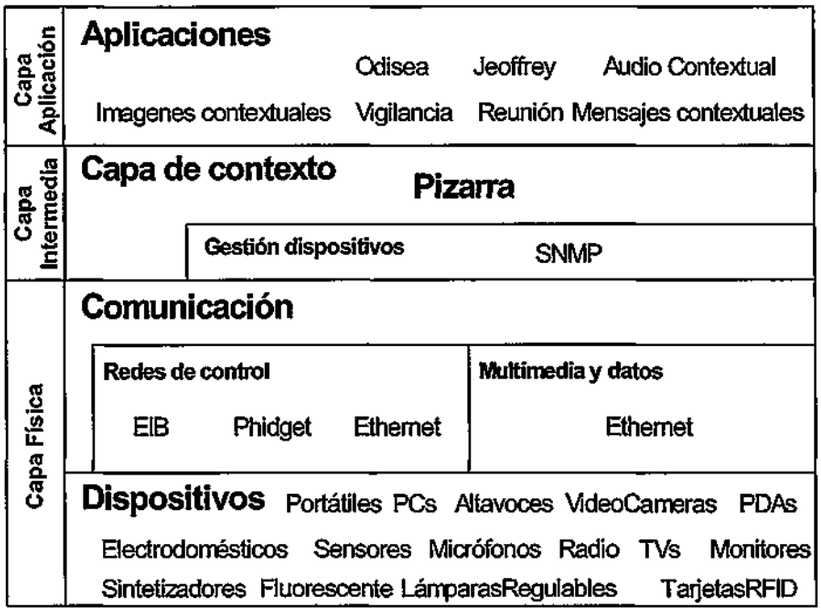


Figura 7.3: Jerarquía de las diferentes redes de control y multimedia instaladas en el laboratorio, y su conexión con la pizarra.

Los dispositivos instalados caen dentro de una de las tres siguientes categorías:

- **Domóticos.** Se componen de varios sistemas independientes.
  - **Sistema de control automático de entrada.** Tiene tres partes (a) Un sensor electromagnético que informa cuándo la puerta está abierta o cerrada. (b) Un sistema de tarjetas por radio frecuencia que identifica al usuario que intenta acceder al entorno. (c) Una cerradura electrónica que permite abrir la puerta.
  - **Sistema de iluminación.** Se compone de dos fluorescentes que aportan la iluminación principal del entorno; dos lámparas de pie, cada una de ellas consistente en una luz de ambiente regulable y en una lámpara de lectura; y finalmente dos interruptores que permiten controlar manualmente la iluminación principal. Todos estos componentes están conectados al bus EIB.

- **Electrodomésticos.** Actualmente se dispone de una cafetera conectada a una toma de corriente del bus EIB, de modo que se puede automatizar la preparación del café.

- **Información Audio-Visual.** Cada dispositivo del sistema audio-visual genera y recibe información multimedia que se distribuye mediante la red ethernet al resto de los dispositivos. El control se realiza directamente mediante el protocolo SNMP.

  - Un generador de música. Éste se compone de dos dispositivos: una radio USB y un reproductor de CDs, conectados al mismo ordenador. Este generador se encarga de realizar la difusión del audio que se captura —bien de la radio o bien del CD— a través de la red Ethernet. El generador emite en un canal de difusión configurable a través de la pizarra. También se puede controlar el volumen en que emite.
  - Dos altavoces de alta fidelidad. Estos altavoces se conectan a un ordenador personal que recibe el audio que se emite por la red multimedia. Si hubiera distintos generadores de música se podría configurar cada altavoz para que captara a un generador distinto. Para ello, a través de la pizarra se puede cambiar el canal de difusión que se desea escuchar. Además se puede controlar el volumen tanto del generador como de cada altavoz por separado.
  - Tres ordenadores de sobremesa que se emplean como puestos de trabajo. Cada ordenador personal dispone de un sintetizador de audio conectado a los altavoces que permite enviar mensajes vocales a la persona que se encuentre enfrente.
  - Una televisión de alta definición. Esta televisión se conecta a una toma de corriente del bus EIB de modo que se puede controlar su encendido y apagado.
  - Varios monitores planos, cada uno conectado a un ordenador personal.
  - Dos cámaras de vídeo, que se conectan directamente a la red Ethernet.

- **Interacción vocal.** La interfaz en lenguaje natural se compone de dos tipos de micrófonos:

  - Micrófonos inalámbricos que permiten al usuario mayor libertad de movimientos y que tienen una alcance suficiente para todo el laboratorio.
  - Micrófonos de cabeza que mejoran la tasa de reconocimiento, a cambio de limitar la movilidad del usuario.

En el apéndice B puede encontrarse la descripción en BBXML del laboratorio, su división en áreas y descripción de los dispositivos que conforman el laboratorio.

7.4. Capa de contexto

La capa de contexto se despliega sobre la infraestructura de dispositivos y redes de comunicación que conforman el entorno experimental. Esta capa consiste en un servidor que implementa la arquitectura de pizarra, incluyendo los módulos del controlador de dispositivos SNMP y otros módulos adicionales.

La capa de contexto ha sido extensamente descrita en los capítulos 4, 5 y 6 por lo que en este apartado sólo se van a tratar ciertas consideraciones de implementación que quedaron fuera de los capítulos anteriores.

7.4.1. Interacción con la pizarra

La interacción con la pizarra se realiza a través de la interfaz de comunicación descrita en el capítulo 5. Ésta presenta a los módulos de la capa de aplicación una interfaz común que abstrae la diversidad de dispositivos del entorno físico. De tal forma que cualquier tipo de red de dispositivos que implemente una pasarela a nuestra capa de contexto puede integrarse dentro del sistema.

Hay que tener en cuenta que los módulos no tienen por qué implementar todas la operaciones que proporciona la pizarra. Cada uno elegirá el subconjunto de operaciones que necesite, lo que facilita la implementación de los mismos.

El acceso se realiza mediante protocolos de comunicación y lenguajes de representación estándar, lo que consigue una implementación que es independiente del lenguaje de programación elegido. Aunque todas las aplicaciones se han desarrollado en Java resulta sencillo integrar nuevas aplicaciones empleando otros lenguajes.

Por ejemplo, en la figura 7.4 queda reflejado cómo empleando herramientas estándar se puede construir un *guión*[1] que permite enviar mensaje de voz sobre un sintetizador.

```
#!/usr/bin/bash

msg=$(echo $1 | sed -e 's/ /%20/g')
wget http://odisea.ii.uam.es:8080/interact/bb/set/device
    /syntb349/props/say?value=$msg -q -O - > /dev/null
```

Figura 7.4: Ejemplo envío de un mensaje vocal emplean un *guión* de *bash*

7.4.2. Rendimiento de la pizarra

Una de las preocupaciones en la implementación de la pizarra ha sido el número de entidades que ésta es capaz de manejar. Al ser el modelo de programación orientado a datos se puede pensar que incluso en entornos de tamaño medio —un

[1] *script*

edificio, por ejemplo— el número de entidades pueda ser elevado. Dado que en este primer prototipo se ha empleado un único servidor, se ha medido cómo varía la respuesta del mismo en función del número de entidades que tiene el entorno, de forma que se obtenga una estimación del tamaño del modelo que se puede manejar.

Para realizar las pruebas se han utilizado dos ordenadores PIV con 128MB de memoria conectados entre si a través de una red Ethernet aislada. En uno de los ordenadores se instaló el servidor de pizarra, y en el otro se instaló un cliente. Este cliente es capaz de realizar peticiones del contexto o cambios en el mismo. Las peticiones se envían a intervalos de tiempo fijos.

Se ha fijado el intervalo de tiempo a 1s. y se han realizado siete pruebas distintas cambiando el número de entidades de la pizarra. Para ello se han generado ficheros aleatorios con distintos números de entidades que varían desde 1 hasta 1.000.0000 de entidades. Cada prueba consiste en la repetición de 110 peticiones sobre la pizarra midiendo en cada petición el tiempo desde que se envía la petición hasta que se recibe. Para cada prueba realizada se ha obtenido la media del tiempo de respuesta. En este análisis de los datos no se han contabilizado las diez primeras peticiones. Tampoco se han contabilizado aquellas muestras que cayeran cuatro veces por encima o por debajo de la desviación típica. Este sesgo permite eliminar aquellas medidas inusualmente elevadas.

En la figura 7.5 se puede observar la variación del tiempo de respuesta en función del número de entidades de la pizarra. En el eje X se muestra el número de entidades en escala logarítmica, mientras que en el eje Y se muestra el tiempo de respuesta en milisegundos. Como se puede apreciar la respuesta del servidor se mantiene constante hasta con 10.000. Con 100.000 entidades se mantiene por debajo de los 200ms. A partir de ese instante la respuesta se dispara llegando hasta 2s. en el caso de 1.000.000 de entidades.

7.4.3. Usuarios y roles

Un concepto que ayuda a determinar la relación entre el sistema y los usuarios consiste en definir los distintos roles con los que una persona puede interaccionar con el sistema. Estos roles no tienen una existencia independiente, sino que suelen aparecer interrelacionados como integrantes de una organización. En el caso de los *entornos inteligentes*, el propio entorno marca de forma natural los límites de una posible organización. A la hora de interaccionar con las distintas aplicaciones que se han desarrollado los diferentes roles establecen limitaciones en cuanto a las acciones que se pueden realizar. Éstas se traducen en vetar el acceso a determinados recursos según el rol del usuario. También se puede dar el caso de recursos públicos en los que varios roles tienen derechos de acceso pero con distintas prioridades, de forma que se favorezca a aquellos roles que tenga privilegios sobre el entorno.

Para mejorar la claridad de la explicación nos apoyaremos en la vista organizacional de la metodología AMENITIES [108]. Esta metodología permite describir sistemas cooperativos basándose en modelos de comportamiento y tareas, consiguiéndose una visión muy intuitiva del manejo de roles. La vista organizacional

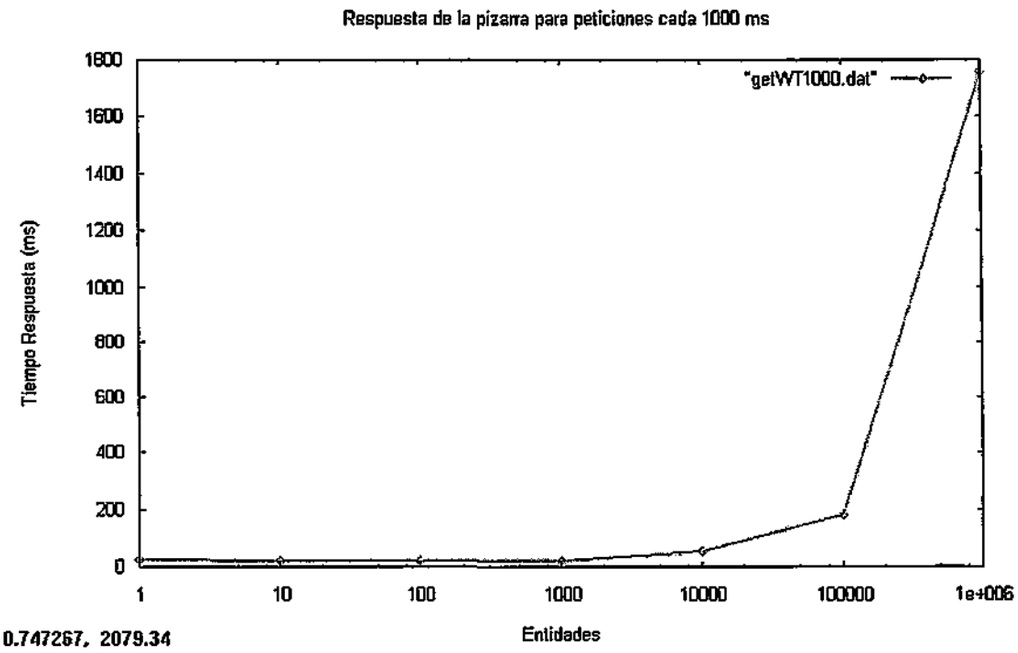


Figura 7.5: Respuesta de la pizarra para peticiones cada 1000 ms

permite describir las relaciones entre los usuarios del sistema en función de sus roles. Además de la vista organizacional, hay otras tres posibles vistas conceptuales del sistema: cognitiva, interacción y de la información. Esta última permite describir el modelo estático de datos; mientras que la primera y segunda especifican la dinámica del sistema. AMENITIES se ha desarrollado como una modificación de UML, por lo que la notación empleada resulta familiar a cualquier lector que tenga conocimientos en este lenguaje.

Una persona dentro del entorno puede comportarse según uno de los siguientes roles:

- **Usuario.** Así se denomina a la persona que está interaccionando con el entorno y cuya identidad se encuentran registrada.
- **Invitado.** Una persona actúa como invitado del entorno, bien cuando no está registrada o bien cuando no ha sido autenticado.
- **Propietario.** Este rol sólo lo puede desempeñar un usuario, y hace referencia a aquellas personas que tienen más privilegios de uso del entorno.

Una persona puede adquirir o perder un rol según ciertas capacidades y leyes. Las capacidades se definen como habilidades o responsabilidades que se asocian a cada persona y que determinan si puede o no asumir un cierto rol. Las leyes imponen restricciones extrínsecas a las personas y que modifican su comportamiento.

Así, una persona para asumir el rol de usuario tiene que cumplir dos condiciones: por un lado, tiene que estar registrado cómo tal; por otro tiene que haberse autenticado correctamente. La primera condición se corresponde con una capacidad de la persona denominada *usuario*. La segunda se identifica con una ley que se denomina *autenticado*. En el caso de que esta ley no se cumpla, a la persona se le asigna el rol de *invitado*.

La adquisición de un rol determina la interacción de ese persona con el entorno. Una persona que tenga el rol de invitado dispondrá de mayores limitaciones a la hora de poder emplear los recursos del entorno que una persona que se identifique como usuario. Es más, si esta persona se corresponde con un propietario dispondrá de mayores privilegios. En el apartado 4.6.1 se exponía que en el modelo del contexto primario se almacena el rol que ocupa cada persona, de manera que se le asociaba o bien usuario o bien invitado. Así, cada persona que se encuentra en el entorno tiene asociado uno de los dos roles. El rol de propietario viene determinado si el usuario además tiene la relación *esPropietario* con el entorno (véase el apartado 5.9). La adquisición de un determinado rol modifica el comportamiento del entorno en tanto en cuanto éste influye en la prioridad que se le asigna a los comandos emitidos por cada persona (véase el apartado 5.10.2).

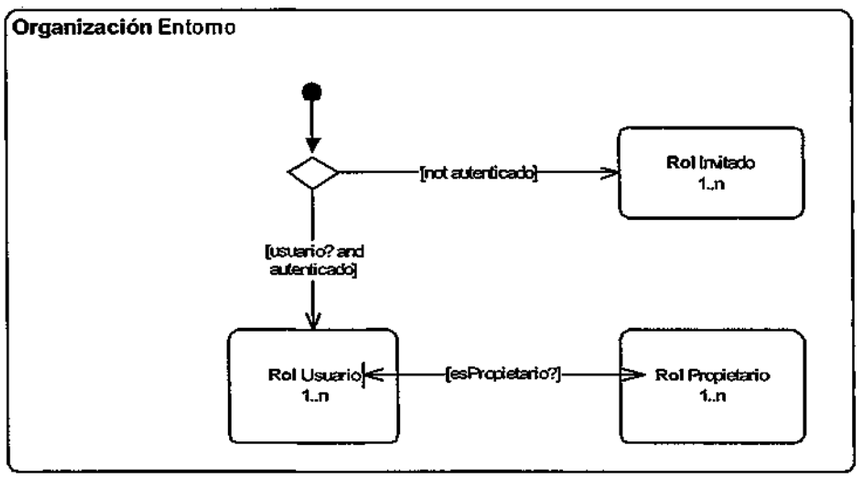


Figura 7.6: Vista organizacional del entorno en que se despliegan los demostradores

La vista organizacional se muestra en la figura 7.6. Las capacidades se distinguen de las leyes en que se les añade el carácter *?* al final de nombre. El cumplimiento de las capacidades y leyes produce que una persona adquiera uno u otro rol. Inicialmente se tiene que decidir si a la persona se le asigna el rol de *usuario* o de *invitado*. Esta decisión gráficamente se modeliza mediante un seudoestado inicial (punto negro) y un elemento de bifurcación (rombo). La transacción entre dos roles puede ser aditiva, o de cambio. En el primer caso, la persona asume el nuevo rol conservando los que ya tuviera. En el segundo caso, la persona sustituye el rol nuevo por los que poseyera. El rol de *propietario* se adquiere mediante una transición aditiva desde rol *usuario*. Este hecho se representa mediante un arco.

de doble flecha indicando el estado origen inicial con pequeña barra vertical.

7.5. Agentes contextuales

La arquitectura de pizarra que se propone en esta tesis facilita la cooperación entre las diversas aplicaciones y dispositivos. Esta ventaja se puede observar en el desarrollo de aplicaciones donde se promueve un diseño modular e incremental. En el apartado 7.6 se van describir las aplicaciones que se han desarrollado como parte de los demostradores de las ideas expuesta en esta tesis. Estas aplicaciones se componen de diversos módulos o agentes independientes que se comunican a través de la pizarra. Previo a la descripción de las aplicaciones, se van a analizar los diferentes agentes disponibles, ya que la construcción de una aplicación se basa en la combinación diversos agentes.

En el diseño de cada agente particular hay que tener presente el mecanismo de comunicación del contexto que impone la pizarra. Cada agente únicamente tiene que conocer la información que se almacena en la pizarra. Ésto fomenta que cada agente se diseñe de manera independiente al resto. La ventaja de este enfoque reside en que se consigue modularizar la funcionalidad ofrecida por el entorno, de manera que la sustitución de un módulo por otro no repercuta en el resto del sistema. Esta facilidad de sustitución favorece además una implementación incremental: los agentes se pueden ir desarrollando en sucesivas versiones que refinen su comportamiento.

El desacoplo entre los agentes desarrollados se obtiene en tres niveles: temporal, espacial y funcional. El primero permite que los agentes se ejecuten de manera asíncrona empleando la pizarra como almacenamiento intermedio de información. Por ejemplo, el Sensor de teclado/Ratón almacena en la pizarra la información sobre la actividad del ordenador. Esta información es consultada posteriormente por un intérprete de localización para determinar si el usuario está frente al ordenador. No existe ninguna comunicación adicional entre ambos agentes para coordinarse. El segundo significa que otros agentes no tienen por qué conocer la existencia de los agentes. Siguiendo con el ejemplo, el sensor no necesita saber si la información la recibe un determinado intérprete, ni si es uno o más de uno. Por último, el desacoplo funcional consiste en que el sensor no precisa conocer para qué se va a emplear el cambio que produce en la pizarra. Cada agente se ocupa de gestionar su parte del contexto sin tener en cuenta en qué influyen sus modificaciones. Estos tres niveles de separación simplifican el desarrollo de cada agente en particular.

La independencia entre los agentes viene también soportada por el mecanismo de resolución de conflictos que se describe en el capítulo 5. Este mecanismo evita que los agentes tengan que negociar con otros agentes el acceso a los dispositivos, ya que esté resuelve en la propia pizarra. De esta manera se sigue manteniendo la autonomía en la implementación de cada agente.

Los agentes desarrollados se pueden catalogar en cinco clases diferentes: productores, interpretes, consumidores, sensores y actuadores. En la tabla 7.1 se han clasificado los agentes desarrollados en función de este criterio.

Sensores	Actuadores	Productores	Consumidores	Intérpretes
Sensor Puerta	Cerradura	Productor Imágen	Agente Apertura-Puerta	Detector Alarma Puerta
Lector RFID	Altavoz	Productor Audio	Agente Seguridad	Detector Intrusos
Sensor Teclado-Ratón	Pantalla		Consumidor Imágenes	Detector Ocupantes
Sensor Encendido/Apagado			Consumidor Audio	Detector Localización
			Identificador de reuniones	Ag. Gestor Imágenes
				Ag. Gestor Audio

Tabla 7.1: Clasificación de los agentes contextuales implementados como parte del *entorno inteligente*

7.5.1. Sensores y Actuadores

Los dispositivos que se emplean como sensores y actuadores han sido descritos dentro del apartado 7.3.3. Dependiendo de cada demostrador se empleara un subconjunto de sensores y actuadores distintos. Por ejemplo, el *Vigilante de puerta bien cerrada* requiere únicamente del sensor de puerta, mientras que en el caso de las dos interfaces de usuario descritas se emplean todos los dispositivos del entorno.

Además de los dispositivos hardware comentados, es decir, el sensor Puerta y el Lector de Tarjetas RFID, se han diseñado dos nuevos sensores que permiten detectar si un ordenador está siendo usado o no. Ambos sensores han sido desarrollados por separado aunque sus funcionalidades se solapan, y a la vez se complementan. Los dos sensores obtienen el valor de la propiedad *ocupado* de un ordenador, de manera que se pueda conocer si un ordenador está siendo utilizado o no. Cada sensor se puede instalar por separado, y ambos aportan una estimación de la ocupación del ordenador. La combinación de ambos sensores se realiza a través de la pizarra. Juntos aportan una mayor fiabilidad del contexto obtenido.

Sensor de teclado y ratón

Este sensor establece si alguien se encuentra interaccionando con el ratón o el teclado conectados a un ordenador. Consiste en un salvapantallas que avisa de que el ordenador está libre cuando se activa y cuando se desactiva avisa de que está ocupado. El sensor modifica en ambos casos la propiedad *ocupado* del ordenador en él cual está instalado.

Sensor de encendido y apagado

Este sensor detecta si un ordenador está encendido o apagado. Consiste en un guión que se ejecuta durante la operación de encendido y de apagado del ordenador. En el primer caso modifica la propiedades *operativo* con el valor de

verdadero, e inicializa la variable *ocupado* a verdadero. En el segundo caso cambia ambas propiedades al valor falso.

7.5.2. Productores

Este grupo lo forman a aquellos agentes software que producen información que vuelcan en la pizarra. La información de los productores puede ser un cambio en el contexto, o una acción a realizar sobre el mundo físico. Estos cambios pueden ser recibidos tanto por los interpretes de forma que genere información más elaborado, como directamente por los consumidores o los actuadores.

También se incluyen en este apartado aquellos agentes capaces de generar información multimedia. Esta información no pasa por la pizarra sino que se distribuye a través de la red multimedia. En cualquier caso, estos productores han sido diseñado para que también se ejecuten independientemente de los consumidores.

Fuente de imágenes contextuales

Este productor se encarga de proveer una imagen para que pueda ser descargada a través de una URL. Esta imagen cambia a lo largo del tiempo, de manera que cada vez se puede descargar una distinta. La selección de la imagen se realiza en función de la identidad de los ocupantes del entorno. El agente almacena una lista circular con los ocupantes de la habitación que obtiene de la pizarra. Para cada ocupante, se mantiene otra lista circular que almacena las URLs que apuntan a sus imágenes favoritas. La URL de la imagen que se muestra en cada momento se elige mediante un procedimiento de selección en dos fases. Primeramente, se selecciona una persona de la lista de ocupantes, y a continuación se selecciona la siguiente URL de la lista de URLs. Ambas listas emplean un algoritmo de *round-robin* para decidir el siguiente candidato. La URL seleccionada se mantiene durante un tiempo. Pasado este tiempo se elige una nueva imagen siguiendo el procedimiento antes explicado.

La lista de ocupantes se mantiene actualizada con la información de la pizarra. Cuando un nuevo ocupante entra el entorno, el agente recibe la notificación. Esta persona se añade a la lista de ocupantes y se crea una nueva lista con las URLs de sus imágenes favoritas. Cuando un ocupante abandona la habitación, se elimina la lista con sus URLs favoritos y se borra de la lista de ocupantes. En el caso de la que la habitación esté vacía, se seleccionan imágenes por defecto que no están asociadas a ningún usuario.

Cuando se inicializa la fuente de imagen almacena una entidad de tipo *ProductorImagen* en la pizarra que contiene la siguiente información:

- **Propiedad url**. URL de la última imagen seleccionada.
- **Propiedad refresco.**Tiempo que permanece una imagen seleccionada.

La fuente de imagen actualiza en la pizarra la URL de la última imagen seleccionada cada vez que se vence el tiempo de permanencia. Los consumidores de imagen están suscritos a este cambio de modo que se les notifica la nueva URL

cada vez que haya un cambio. Con la nueva URL los consumidores muestran la nueva imagen.

Productor de Audio

Los productores de audio se asocian a dispositivos multimedia capaces de difundir audio por el entorno. Se representan en la pizarra como un dispositivo que tiene una entidad asociada de tipo *ProductorAudio*. En el caso del laboratorio *lab_B403* se dispone de un servidor de audio capáz tanto de distribuir una radio digital como de reproducir discos compactos. Este servidor dispone de una dirección IP y un puerto en el que se encuentra emitiendo los paquetes de difusión. Como la retrasmisión se realiza mediante difusión, los productores no necesitan conocer los consumidores conectados. Dispone de propiedades que permiten seleccionar el tipo de fuente que se quiere escuchar (radio o cd), cambiar de emisora en el caso de la radio, cambiar de canción para el cd o cambiar el volumen.

7.5.3. Consumidores

Son los receptores finales de la información contextual. Acceden a la información contextual a través de los cambios en la pizarra y adaptan su comportamiento en función de éstos. También se incluyen aquellos agentes con capacidad para consumir tráfico multimedia, como es el caso de los consumidores de audio e imágenes.

Agente de apertura de puerta

Este agente se encarga de decidir cuándo se abre la cerradura de la puerta. Para ello, cuando se inicializa obtiene las personas que pueden acceder al entorno. Con esta lista obtiene el conjunto de tarjetas válidas que abren la puerta. Una vez inicializado el agente consume los identificadores que lee el lector de tarjetas y si éste es un identificador válido actúa sobre la cerradura de la puerta.

Agente de Seguridad

Este agente consume las alertas de seguridad que se producen en el entorno. Estas alertas, actualmente, pueden ser relativas a la puerta o a la aparición de un intruso. Este agente se encuentra subscrito a los comandos de añadir una nueva relación a la seguridad del entorno. Cuando se recibe un evento de nueva relación el agente entra en acción. Su funcionamiento depende del grado de seguridad de la habitación (véase la tabla 4.7.1) y del tipo de alarma.

Las alarmas que se produzcan sobre la puerta, además tienen en cuenta la presencia de ocupantes en el entorno. Si el grado de seguridad es bajo, el agente se limitará a anotar la alarma pero no tomará ninguna acción al respecto. En el caso de que el grado de seguridad sea normal, el agente accede a la pizarra para recuperar la información relativa a la alarma producida. Para ello emplea el identificador de la alarma que ha recibido en el evento.

Dependiendo del tipo de la alarma la repuesta del agente varía. El primer caso se produce cuando la alarma indica que la puerta se ha quedado abierta. En

este caso, la respuesta del agente será distinta según la habitación se encuentra vacía o no. Si la habitación se está ocupada se avisa mediante un mensaje vocal de la irregularidad. Para ello se hace uso de los sintetizadores que operan sobre los altavoces. En el caso de que no haya nadie en la habitación, se envía un correo electrónico a los propietarios de la habitación. En este caso, cuando la puerta finalmente se cierra, se avisa de nuevo al responsable de que la alarma se ha resuelto. Cuando el grado de seguridad es elevado —aparte de las medidas anteriores— se toman medidas de precaución para evitar que alguien se olvide la puerta abierta, las cuales consisten en enviar mensajes vocales recordatorios de que hay que cerrar correctamente la puerta. Éstos se envían independientemente de si se ha producido o no la alerta.

El segundo caso corresponde a alertas producidas por intrusiones en el entorno. Estas alertas también tienen en cuenta el grado de seguridad. Si éste se encuentra en nivel bajo, de nuevo, sólo se anota la intrusión. Por el contrario si el nivel es normal o elevado se avisa al intruso de que tiene que identificarse introduciendo una contraseña a través de consola. Si no se consigue una identificación válida en un determinado tiempo, el agente toma una serie de medidas que consisten en encender las luces de la habitación (en el caso de que estuvieran apagadas); notificar por correo electrónico a los responsables del entorno; y emitir mensajes disuasorios por los altavoces del entorno.

Consumidores de imágenes

Estos consumidores se encargan de mostrar imágenes en una pantalla. Cada agente se encuentra asociado a una pantalla plana. Cada vez que se inicia un agente se añade una entidad de tipo *ConsumidorImagen* a la pizarra que representa a la nueva pantalla. Este agente se mantiene inactivo hasta que se añade una relación *estaConectadoA* en la pizarra entre una fuente de imagen y él. Entonces, el agente consumidor consulta la pizarra para recuperar la última URL seleccionada por la fuente de imagen asociada. El agente consumidor obtiene mediante el protocolo HTTP la imagen seleccionada y la muestra en la pantalla. Este proceso se repite cada vez que la fuente de imagen selecciona una nueva URL, y se mantiene hasta que se elimina la relación entre las dos entidades.

Consumidor de Audio

Los consumidores de audio se asocian a altavoces. Siempre que se establece una relación *estaConectadoA* entre la fuente y el consumidor de audio, el consumidor configura al altavoz para escuchar en la dirección IP y el puerto en el emite el productor correspondiente. De esta manera comienza a sonar el audio emitido por el servidor. Si la relación se borra, el altavoz correspondiente deja de escuchar en esa dirección y puerto.

Identificador de reuniones

Este consumidor se encarga de avisar a los posibles interesados en una reunión cuando está se pone en marcha. Para ello cada usuario tiene una lista de personas

con las que habitualmente se reúne. Esta lista se modeliza con una relación *tiene-Reunión* entre los dos implicandos. Cuando tres o más usuarios se encuentran en la misma habitación se comprueba si cada uno tiene una relación de *tieneReunión* con los otros. Si es así, para cada usuario se halla el conjunto de usuarios que suelen tener reunión con ambos incluido él mismo, y se extrae la intersección de todos los conjuntos. Si en el subconjunto resultante se encuentran los usuarios presentes en la habitación, se envía un correo electrónico a los restantes usuarios anunciándoles de la inminente reunión. El *IdentificadorReunion* se encuentra subscrito a los cambios en la relación *contiene* del laboratorio *lab_b403*. Cada vez que se produce una modificación en esta relación se comprueba si las personas que está presentes en ese momento son propicias para que se esté produciendo una reunión.

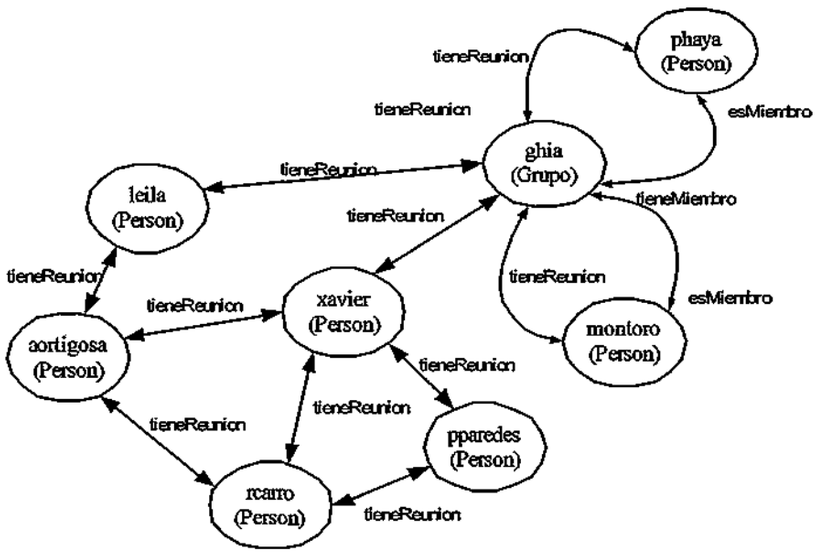


Figura 7.7: Ejemplo de información contextual que se utiliza para detectar reuniones. Las flechas bidireccionales indican una relación de *tieneReunión* en los dos sentidos.

Por ejemplo, supongamos que estamos en la situación de la figura 7.7, y supongamos también que en un momento dado se encuentran en la misma habitación *aortigosa* y *pparedes*. A continuación entra *rcarro*: como hay tres o más personas se ejecuta el agente. Se forman los tres conjuntos anteriormente descritos:

$$C(aortigosa) = \{aortigosa, pparedes, rcarro, xavier, leila\} \quad (7.1)$$

$$C(pparedes) = \{pparedes, aortigosa, rcarro, xavier\} \quad (7.2)$$

$$C(rcarro) = \{rcarro, aortigosa, pparedes, xavier\} \quad (7.3)$$

y se halla la intersección entre ambos

$$I = C(aortigosa) \cap C(pparedes) \cap C(rcarro) \quad (7.4)$$
$$I = \{aortigosa, pparedes, rcarro, xavier\} \quad (7.5)$$

Al encontrarse en la intersección los tres integrantes se deduce que está ocurriendo una reunión, y se envía un correo-electrónico al resto de los usuarios del conjunto, en este caso a *xavier*.

$$I - \{aortigosa\} - \{pparedes\} - \{rcarro\} = \{xavier\} \quad (7.6)$$

Tal como se puede comprobar, el número de relaciones para grupos de reunión grandes es elevado. Por ello, se permite establecer la relación *tieneReunion* con un grupo de usuarios. De esta forma, es como si el usuario tuviera una relación *tieneReunion* con cada una de las personas que conforman el grupo. En el caso de la figura 7.7 a la entidad grupo *ghia* pertenecen *xavier*, *phaya*, *montoro* y *leila*. De esta forma, si en un momento dado se encuentran en la habitación *xavier*, *german* y *leila* se avisará a *phaya* tal como se muestra en 7.7

$$C(xavier) = \{xavier, aortigosa, pparedes, rcarro, leila, phaya, montoro\} \quad (7.7)$$
$$C(montoro) = \{montoro, xavier, phaya, leila\} \quad (7.8)$$
$$C(leila) = \{leila, xavier, phaya, montoro\} \quad (7.9)$$
$$I = C(aortigosa) \cap C(pparedes) \cap C(rcarro) \quad (7.10)$$
$$I = \{xavier, montoro, leila, phaya\} \quad (7.11)$$
$$I - \{xavier\} - \{montoro\} - \{leila\} = \{phaya\} \quad (7.12)$$

7.5.4. Intérpretes

Los intérpretes permiten generar a partir de la información de los sensores nueva información contextual. Se emplean para crear información de más alto nivel de abstracción. Así, por ejemplo, el detector de alerta de puerta transforma la información de que una puerta está abierta en una posible situación peligrosa para el entorno.

Los intérpretes permiten simplificar el diseño de los sensores, que se pueden mantener lo más sencillos posibles. También facilitan la reutilización de los sensores, de manera que un mismo sensor puede servir para generar información contextual de diferente tipo. Por ejemplo, el sensor de puerta se emplea tanto para deducir un mal funcionamiento de la misma como para descubrir que hay un intruso en la habitación.

Al igual que los sensores, se pueden combinar varios intérpretes para obtener la misma información contextual. Éste es el caso del *Detector de ocupantes del entorno* y del *Detector de localización*. Ambos obtienen la localización del usuario aunque con distintas técnicas y con distinta resolución. Mientras que el primero puede determinar si el usuario está dentro del laboratorio, el segundo puede

ajustar hasta el nivel de área dentro de un entorno. Cada uno funciona por separado y se instalan de forma independiente. El *Detector de ocupantes* se basa en la tecnología RFID, mientras que el *Detector de localización* emplea los sensores de detección de ocupación de un ordenador. Ambos se encuentran instalados en el laboratorio *lab_b403*, aunque, los segundos, al ser más baratos y fáciles de instalar, también están disponibles en otros entornos, como son los despachos de profesores y doctorandos del departamento. De esta forma, en los despachos también se obtiene una estimación de la localización, aunque menos fiable que en el laboratorio. Ahora bien, desde el punto de vista funcional de los consumidores que emplean la información de localización, no existe diferencia entre operar en un despacho o en el laboratorio, ya que en ambos obtienen la localización a través de la pizarra sin necesidad de preocuparse de cuántos intérpretes se han instalado.

La generación de nueva información se puede realizar de forma incremental empleando varios intérpretes. En la aplicación de mensajes contextuales (véase el apartado 7.6.4) primeramente se emplea un intérprete que convierte un identificador de una tarjeta RFID en localización. Otro agente transforma esa localización en información sobre la actividad del usuario. En función de esa actividad, un tercer intérprete deduce si al usuario se le puede interrumpir, y dependiendo de esta última información, una *aplicación sensible al contexto* finalmente decide si le envía o no un mensaje. Este encadenamiento facilita que cada agente puede desarrollarse y mejorarse por separado.

Detector de Alerta de Puerta

Este intérprete se denomina *detectorAlertaPuerta*. Cuando el agente se encuentra operativo se subscribe a los cambios en la propiedad que caracterizan el estado de la puerta. Cada vez que el estado cambia a abierto, se lanza un temporizador. Si el temporizador termina antes de que el estado de la puerta cambie a cerrado, el *detectorAlertaPuerta* añadirá a la pizarra una nueva *AlertaFuncionamiento* (véase el apartado 4.7.1). La *AlertaFuncionamiento* constará de la propiedad *descripción* donde se indicará un texto que haga referencia a qué puerta se ha quedado abierta. Además incluye una o más relaciones *localizadaEn* que indican a qué entornos pertenece la puerta; y una relación *recursoAfectado* que apunta al sensor de la puerta. Una vez añadida la entidad a la pizarra, el agente mandará el comando de añadir una nueva relación *alerta* entre la entidad seguridad del entorno y la nueva alerta añadida.

En la figura 7.8 se aprecia la configuración de las entidades y relaciones antes y después de que se haya disparado la alerta.

Detector de Intrusos

El agente *detectorIntrusos* se encuentra subscrito a los cambios en la variable *vacío* del entorno. Cuando detecta que la puerta se ha abierto, evalúa si es posible que haya intrusos. Actualmente este agente obtiene esta estimación en función de una franja horaria en el cual se ha determinado que el entorno tiene que estar desocupado. Si el evento recibido cae dentro de esas horas entonces el *detectorIn-*

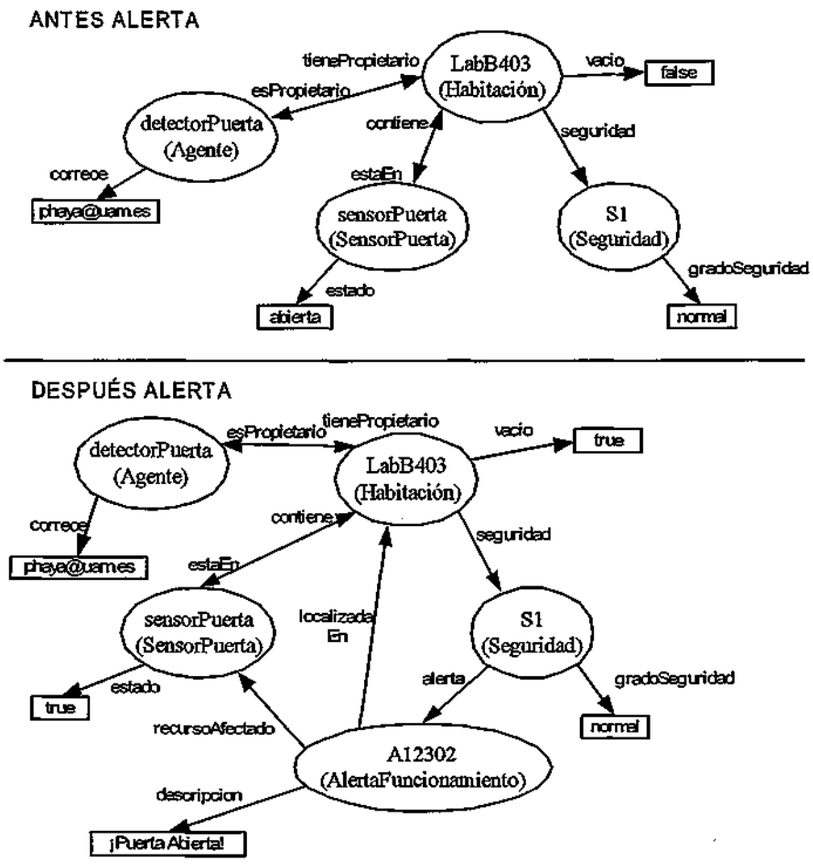


Figura 7.8: Configuración de la pizarra antes y después de haberse producido la alarma

trusos añade una alerta de tipo *Intrusion* mediante un procedimiento similar al que se describió para el *detectorPuerta*.

Detector de ocupantes del entorno

Este intérprete se encarga de actualizar la información de localización del usuario y presencia del entorno. Para ello transforma la información proveniente del lector de tarjetas en contexto de localización. Cada vez que el lector de tarjetas lee una nueva tarjeta el intérprete obtiene de la pizarra si el usuario está dentro de la habitación consultando la relación *contiene* del entorno. Si el usuario ya está en el interior, interpreta que está abandonando la habitación, en caso contrario interpreta que está entrando. En ambos casos modifica la información de la localización y presencia modificando las relaciones *contiene* y *estaEn* y las propiedades *vacio* y *num_ocupantes*.

La figura 7.9 muestra el intercambio de mensajes en dos situaciones distintas en la que se pasa la misma tarjeta. La primera vez el usuario se encuentra dentro de la habitación, con lo que se interpreta que se quiere salir, y en la segunda fuera, con lo que se deduce que quiere entrar.

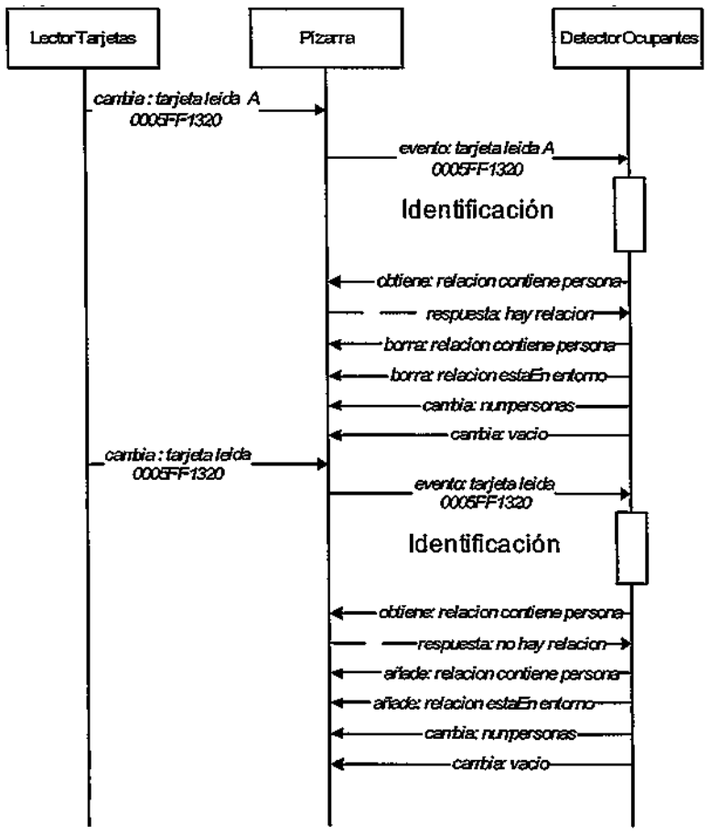


Figura 7.9: Funcionamiento del detector de ocupantes del entorno

Detector de Localización

Este agente aporta un segundo mecanismo para descubrir la localización del usuario. Este intérprete fue desarrollado después que el anterior, y constituye un mecanismo sencillo e independiente para afinar la localización. Se basa en la información de ocupación de un ordenador a partir de los sensores de teclado y ratón, y de encendido y apagado. Cuando se inicializa, el agente realiza una asociación entre ordenadores personales y usuarios del laboratorio consultando las relaciones *esPropietario*. A continuación el agente se subscribe a los cambios en la propiedad *ocupado* de los ordenadores.

Cada vez que la propiedad cambia a verdadero, el agente interpreta que el usuario está delante del ordenador con lo que emplea la información de localización del ordenador personal para actualizar la localización del usuario. Para ello consulta la relación *estaEn* del ordenador y obtiene el lugar donde se encuentra el ordenador personal. Con esa información actualiza las relaciones que hubiera de tipo *estaEn* y *contiene* del usuario.

En el caso del laboratorio *lab_b403* los ordenadores se encuentran localizados en alguna de las dos áreas definidas: salón y oficina. Para este laboratorio, el intérprete utiliza una heurística adicional para determinar el área donde se encuentra un usuario. Si el ordenador se encuentra desocupado y el usuario no ha salido del entorno, entonces el área donde se encuentra pasa a ser la contraria a donde se encuentra el ordenador.

Agente gestor de flujos de imágenes

Este agente se encarga de decidir las asociaciones entre las fuentes y los consumidores de imágenes. Determina qué fuente se conecta a qué consumidor creando una relación en la pizarra entre las dos entidades que representan a ambos módulos. De esta forma se crea una red de conexiones entre fuentes y consumidores que se actualiza dinámicamente.

Se ha desarrollado una interfaz gráfica que actúa como módulo gestor. Ésta permite configurar manualmente la configuración de cada una de las pantallas. Al inicio la herramienta crea dos listas desplegables con las fuentes y consumidores que se encuentran representadas en la pizarra. Estas listas se actualizan instantáneamente cuando aparecen o desaparecen nuevos módulos. Seleccionando un módulo de cada una de las listas se crea un relación entre ambas. Una ventana almacena las relaciones establecidas en la pizarra. Mediante esta lista el usuario puede elegir una relación para ser eliminada. El tiempo de permanencia de las imágenes en las pantallas también se puede modificar a través de la interfaz.

Agente gestor de flujos de audio

Finalmente, se incluye un agente gestor de flujos de audio que actúa como consumidor de su localización modificando los flujos de audio dependiendo de la información que reciba. Este agente tiene asociado una interfaz que permite al usuario interaccionar con la aplicación. Cada vez que el usuario se conecta, se crea una sesión que se mantendrá activa hasta que el usuario decida explícitamente

terminarla. A cada sesión se le asocia un gestor de sesión, y un servidor de audio. La misión del gestor de sesión es conseguir que el usuario siga escuchando el audio independientemente de la localización. Para ello, el gestor se encarga de mantener una relación *estaConectadoA* entre el servidor de audio y el altavoz más cercano. Para determinar la localización del usuario, el gestor de sesión se subscribe a los cambios en la relación *estarEn* que se establece entre el usuario y el entorno en que se encuentra. Cuando aparece una relación de este tipo el gestor detecta que el usuario ha entrado en un nuevo entorno. En ese momento, comprueba si existe algún altavoz disponible. En caso afirmativo, establece una relación entre el servidor de audio y el altavoz, y enciende el altavoz. Cuando el usuario abandona la habitación desaparece la relación *estarEn*. El gestor recibe este evento y procede a borrar la relación *estaConectadoA*.

7.6. Aplicaciones sensibles al contexto

Gracias a la arquitectura de pizarra es posible combinar los agentes previamente descritos de manera sencilla para realizar aplicaciones completas. Cada agente se puede reutilizar en varias aplicaciones como es el caso de los intérpretes *Detector de Ocupantes* y *Detector de Localización*.

La modificación de los agentes no tiene por qué influir en la funcionalidad de la aplicación. Por ejemplo, se puede mejorar la tecnología que se emplea para detectar la localización de un usuario, lo cual aumentará la fiabilidad del álbum de fotos contextuales, pero sin tener que retocar el resto de los agentes que componen la aplicación.

Los agentes que componen cada aplicación pueden evolucionar separadamente sin que afecte a la integración con el resto del sistema. Se podría dividir el agente de Seguridad en varios agentes, cada uno que se encargara de atender una alarma distinta. Por otro lado, se podría por ejemplo mejorar las heurísticas que incorporan los intérpretes para generar un alerta, y finalmente, se podrían integrar más sensores que mejoraran la detección de posibles intrusiones. La aplicación mejora a medida que se enriquecen cada uno de los agentes sin tener que realizar apenas esfuerzo en la integración de las nuevas versiones.

También se podría dar el caso de fundir varios agentes en uno. Por ejemplo, el gestor de audio y el gestor de imágenes se podrían sustituir por un único agente que presentará una interfaz común que permitiría manejar ambos flujos. Tanto el álbum de fotos como la aplicación de audio contextual no verían afectada su funcionalidad. Tampoco los productores y consumidores de ambos tipos de tráfico se verían afectados por la sustitución.

El desarrollo de cada aplicación consiste en dos fases: en la primera se identifica la información contextual que se va a emplear. Esta información incluirá parte del contexto primario descrito en el apartado 4.6, y parte de contexto secundario que dependerá del dominio de aplicación. Este último podrá incluir elementos de los expuestos en las extensiones de los apartados 4.7.1 y 4.7.2. En la segunda fase se identifican los distintos agentes que conformarán la aplicación. Para ello, se determina para cada modulo qué parte de la información contextual va a gestionar y

qué tipo de operaciones va a realizar. En la tabla 7.2 se muestra la descomposición de cada aplicación en los distintos agentes descritos en el apartado anterior.

Aplicación	Sensores	Actuadores	Productores	Consumidores	Intérpretes
Vigilantes puerta bien cerrada	Puerta			Agente Seguridad	Detector AlertaPuerta
Alarma contra intrusos	Puerta			Agente Seguridad	Detector Intrusos
Control acceso al entorno	Lector Tarjetas RFID	Cerradura		Agente AperturaPuerta	
Álbum de fotos contextuales		Pantallas	Productor Imagen	Consumidor Imagen	Gestor Imágenes
Aviso de reunión				Identificador Reunión	
Audio contextuales		Altavoces	Productor Audio	Consumidor Audio	Gestor Flujos Audio, Detector Ocupantes y Detector Localización

Tabla 7.2: Descomposición en agentes de las diferentes aplicaciones desarrolladas.

Las aplicaciones desarrolladas caen dentro de alguna de las cinco siguientes categorías:

- **Aplicaciones de control de acceso.** Son aplicaciones que dan servicios relacionados con el estado de la puerta principal (véase 7.6.1).
- **Aplicaciones sensibles a la identidad del usuario.** Se centran en servicios que depende de la identidad de las personas que se encuentran en la habitación.(véase 7.6.2).
- **Aplicaciones sensibles a la localización del usuario.** Estas aplicaciones aportan un servicio que varía según la localización del usuario.
- **Aplicaciones sensibles a la actividad del usuario.**
- **Interfaces de usuario.** Son dos interfaces de usuario, una basada en la web y la otra un gestor de diálogos en lenguaje natural.(véase 7.7).

7.6.1. Aplicaciones de control de acceso

Esta categoría consiste en tres aplicaciones cuya funcionalidad engloba aspectos relativos al acceso al entorno.

Vigilante de puerta bien cerrada

La primera de la aplicaciones previene de que la puerta permanezca accidentalmente abierta demasiado tiempo. Esta aplicación se crea mediante la combinación

del sensor de puerta, del intérprete *detectorAlertaPuerta*, que se encarga de realizar un seguimiento del estado de la puerta, y del consumidor *agenteSeguridad*, que se encarga de tomar las medidas oportunas en el caso de que ésta permanezca demasiado tiempo abierta.

En la figura 7.10 se representa como interactúan a través de la pizarra los distintos agentes involucrados. En el instante inicial el sensor de puerta cambia en la pizarra el estado de la puerta a abierta. Este evento le llega al agente *detectorPuerta*, el cual lanza un temporizador. Como no recibe un evento de puerta cerrada antes de que se agote el tiempo de espera, añade una nueva alerta a la pizarra. A continuación añade una relación entre la entidad que representa la seguridad del entorno (*S1*) y la alerta creada (*A12302*). El *agenteSeguridad* recibe el evento de que se ha añadido una nueva relación. A continuación obtiene la información de la alerta y consulta la propiedad *vacio* de la habitación (*LabB403*). Como esta última es verdadera, obtiene la información relativa a los propietarios de la habitación, en concreto, su correo electrónico. Envía un correo con los datos de la alerta previamente obtenidos, y espera hasta que recibe una notificación de que la puerta ha sido cerrada. En ese momento vuelve a enviar un correo avisando de la resolución de la alerta.

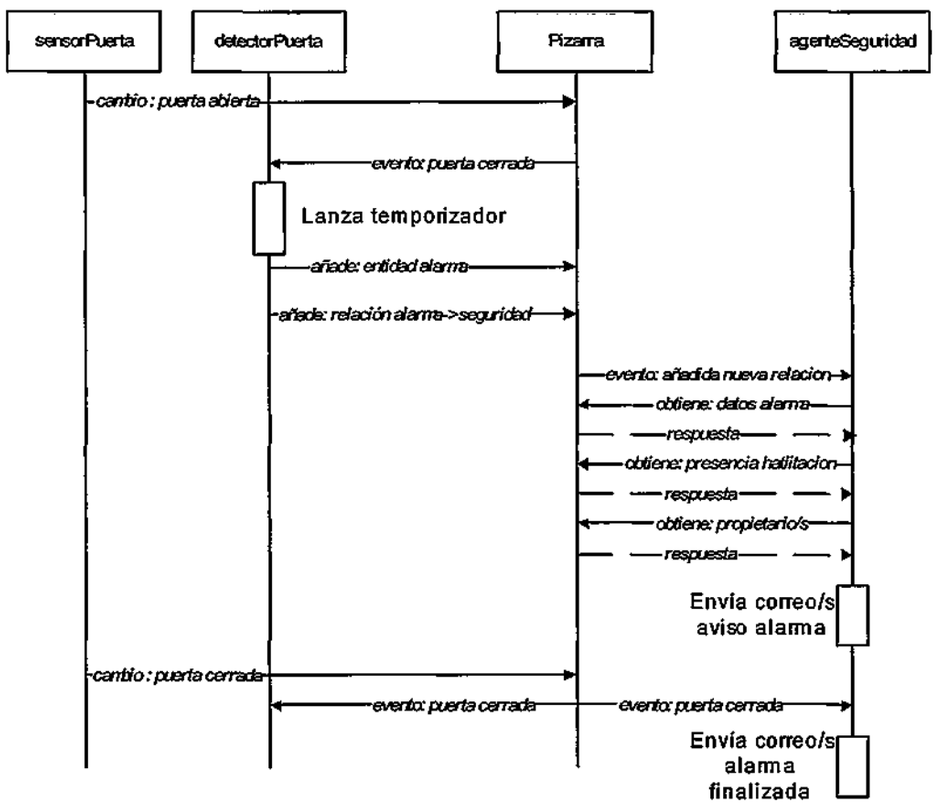


Figura 7.10: Paso de mensajes en la aplicación Vigilante de puerta cerrada

Alarma contra intrusos

La segunda de la aplicaciones previene del acceso de intrusos. Esta aplicación se estructura de manera similar a la anterior. En este caso se cambia el intérprete *detectorAlertaPuerta* por el *detectorIntrusos*. El agente de seguridad *agenteSeguridad* se reutiliza ya que también responde a las alertas de intrusos.

Controlador de acceso al entorno

La última de las aplicaciones decide el acceso a la habitación de una persona. El sistema se compone del sensor *LectorTarjetas* que aporta la última tarjeta leída y del *AgenteAperturaPuerta*, el cual evalúa si se concede o no el paso.

En la figura 7.11 se puede apreciar el paso de mensajes necesario para llevar a cabo la funcionalidad de la aplicación. El *LectorTarjetas* lee una nueva tarjeta y actualiza esa información en la pizarra. El *agenteValidador* comprueba que es una tarjeta válida, y abre la puerta.

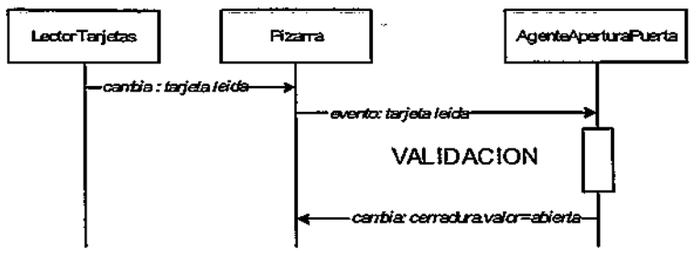


Figura 7.11: Aplicación que permite controla el acceso al entorno

7.6.2. Aplicaciones sensibles a la identidad del usuario

Estas aplicaciones proveen de servicios que dependen del usuario o usuarios que se encuentren en la habitación. La identificación de la persona viene dada por un lector de tarjetas por RFID. Cada usuario tiene asociada una tarjeta que se detecta mediante radiofrecuencia. El entorno posee una antena cercana a la entrada capaz de captar la información almacenada en la tarjeta, que consiste en un identificador unívocamente asociado a cada usuario. Cuando el usuario se acerca a la puerta para acceder al entorno, el sistema lee la identificación del usuario y cambia en la pizarra la propiedad *tarjetaLeida* del lector de tarjetas.

Álbum de fotos contextuales

La primera de las aplicaciones consiste en un álbum de fotos contextuales que sirve como un ejemplo de control de flujos multimedia (véase el apartado 4.7.2). Cada persona tiene asociada una colección de fotos personales. En el laboratorio se dispone de varias pantallas planas que permiten mostrar las fotos asociadas a cada persona. En cada momento se muestra una foto en una pantalla plana. La

foto que se muestra cambia con el tiempo, y depende de las personas que en cada momento se encuentren en el laboratorio.

Para la construcción de la aplicación se han empleado tres de los módulos descritos anteriormente: el *ProductorImagenes*, el *ConsumidorImagenes* y el *GestorFlujosImagenes*.

A continuación se muestra el intercambio de mensajes que se produce en una ejecución de la aplicación (véase la figura 7.12). Al arrancar, tanto el *ProductorImagen* como el *ConsumidorImagen*, añaden una nueva entidad a la pizarra. La entidad *ProductorImagen* constituye un recurso del entorno por sí mismo, mientras que el *ConsumidorImagen* estará asociado a una pantalla de la habitación. Cuando se detecta que una nueva persona se encuentra en la habitación, se añade una relación *contiene* entre la habitación y la persona. La localización del usuario puede ser establecida por alguno de los agentes previamente explicados, ya sea el *detectorOcupantes* o el *detectorLocalización*.

Cuando el *ProductorImagen* recibe que la persona se encuentra dentro de la habitación consulta a la pizarra para obtener las imágenes de la persona que ha entrado en la habitación. Una vez creada la lista de imágenes a mostrar, escoge una y cambia en la pizarra la propiedad *url* con la nueva URL. Como no hay ningún *ConsumidorImagen* conectado a ese *ProductorImagen* no se produce ningún efecto. Dependiendo de la propiedad *refresco*, el *ProductorImagen* modificará en la pizarra cada cierto la URL seleccionada.

En un momento dado, el gestor de imágenes añade un relación *estaConectadoA* entre el productor y el consumidor. Este evento lo recibe el *ConsumidorImagen*, entonces accede a la pizarra para obtener la URL de la imagen que en ese momento se esté mostrando. Lanza la interfaz gráfica y muestra la imagen que previamente se ha descargado de la URL. Una vez que estén conectados productor y consumidor, cada vez que el *ProductorImagen* cambia la URL, el evento lo recibe el *ConsumidorImagen*, que procede a cambiar la imagen. Este proceso se repite hasta que el gestor elimine al relación *estaConectadoA*.

Si entran o salen personas de la habitación, el *ProductorImagen* detecta estos cambios por los eventos de añadir y borrar de la relación *contiene*. Ésto producirá que la lista de imágenes a mostrar aumente o disminuya.

Aviso de reuniones

La segunda aplicación emplea la misma información de identificación que la anterior. Se compone del consumidor *IdentificadorReunion* que se encuentra subscrito a los cambios en la relación *contiene* del entorno *lab_b403*. Así, además del consumidor, se utilizan los intérpretes que proveen la localización. Este aplicación no requiere de mayor explicación dado que toda la funcionalidad la aporta el agente *IdentificadorReunión* que ya ha sido detallado previamente. Este demostrador es un ejemplo de aplicación aislada cuya funcionalidad está encapsulada en un único agente.

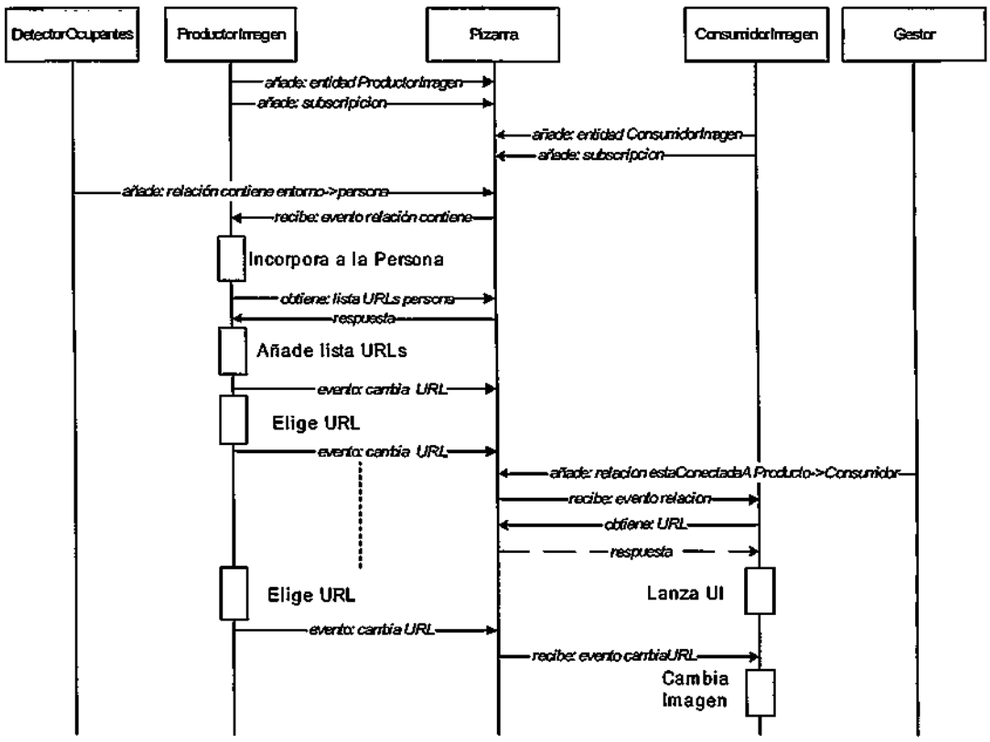


Figura 7.12: Intercambio de mensajes entre los módulos involucrados en la aplicación *Álbum de fotos contextuales*

7.6.3. Aplicaciones sensibles a la localización del usuario

Dentro de este apartado se ha desarrollado una aplicación de distribución de audio dependiente de la localización. El contexto que se emplea viene representado por la relación bidireccional *contiene/estarEn* entre el usuario y el entorno donde se encuentra.

Esta aplicación se compone de un *ProductorAudio*, asociado al servidor de audio, varios *ConsumidoresAudio* asociados cada uno a un altavoz y localizados en distintos lugares, un *Gestor de Audio* y uno o varios intérpretes que se encargan de proveer la información de localización correspondiente.

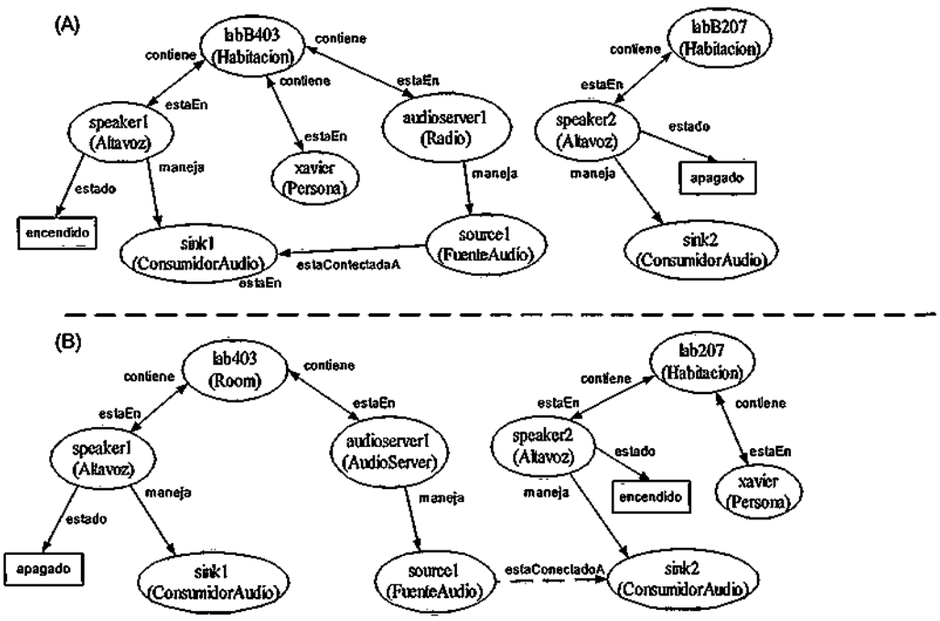


Figura 7.13: Configuración de las relaciones en dos instantes de tiempo. (a) cuando la persona *xavier* se encuentra en la habitación *labB403*, (b) tras descubrirse que *xavier* ha cambiado de habitación al detectarse actividad en el ordenador de la habitación *labB207*

En la figura 7.14 se muestra el intercambio de mensajes que se produce a través de la pizarra en una ejecución de la aplicación. Primeramente *xavier* entra en el *labB403*, el gestor detecta esta situación y configura los altavoces para que se escuche la emisora de radio seleccionada por *xavier*. La configuración de la pizarra se correspondería con la configuración (a) de la figura 7.13. A continuación *xavier* sale del laboratorio *labB403* —se desconecta el altavoz de ese laboratorio— y entra en el laboratorio *labB207*. De nuevo el gestor detecta el cambio y configura el altavoz del laboratorio *labB207* para escuchar la música de *xavier*. La información en la pizarra queda como la configuración (b) de la figura 7.13

Tal como se muestra en la figura 7.13, en la pizarra se almacena un grafo con

las conexiones que en un momento dado existen entre los distintos productores y consumidores.

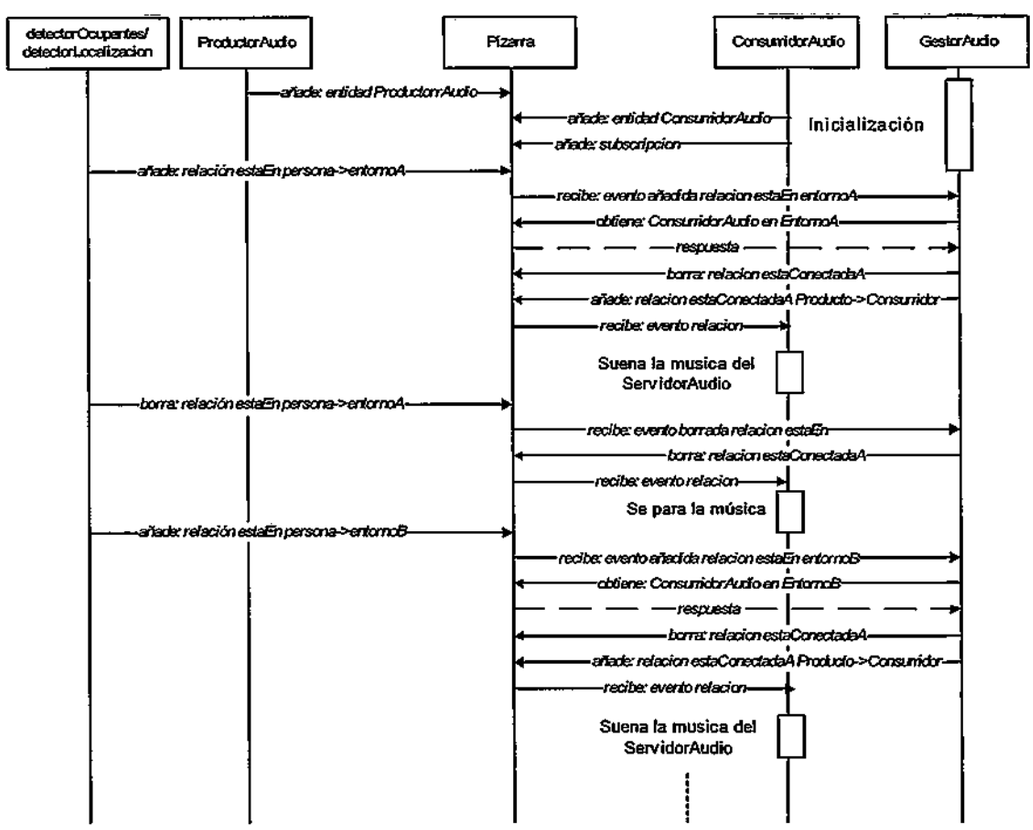


Figura 7.14: Ejemplo de intercambio de mensajes entre los distintos módulos que conforman la aplicación Audio Contextual

Hay que tener presente que el comportamiento de la aplicación puede ser alterado por el estado de la cola de prioridades de la relación *estaConectadoA*. Cada vez que se envía un comando se calcula la prioridad del agente en función del tipo de agente que sea, de la relación del agente/usuario con el entorno y del tipo de orden (véase el apartado 5.10.2). Por ejemplo, si la aplicación *Audio Contextual* emite una orden para cambiar la configuración del altavoz, ésta estará sujeta a las órdenes que previamente se hubieran dado. Si hubiera alguien en la habitación que previamente estuviera escuchando música con mayor prioridad, las órdenes del *Audio Contextual* no tendrían efecto hasta que caducaran las órdenes anteriores. Dado que las órdenes pueden permanecer en la cola sin ejecutarse, es obligación del agente eliminarlas en el caso de que hayan dejado de ser necesarias.

7.6.4. Aplicaciones sensibles a la actividad

El demostrador que se plantea en este apartado se encuentra aún en fase de desarrollo, aunque se prevé que se finalice en breve. Esta aplicación va a servir como caso de estudio de la metodología que se sigue en el desarrollo de una aplicación. Así, se parte del análisis de la información contextual necesaria para desarrollar la aplicación basándose en el modelo de información contextual descrito en el capítulo 4.

La aplicación consiste en un mensajero sensible tanto a la localización como a la actividad del usuario. La primera parte ya se encuentra desarrollada, mientras que la segunda está casi finalizada. La aplicación permite enviar un mensaje corto, ya sea a través de correo electrónico o mediante una interfaz oral. Los mensajes siempre se reciben mediante correo electrónico, y dependiendo de los recursos disponibles en la habitación en la cual se encuentre el usuario —y en función de la actividad que está realizando— también se puede emitir por voz. Esta segunda opción tiene la ventaja de la inmediatez con que se recibe el mensaje, ahora bien, tiene la desventaja de que en determinadas circunstancias puede ser molesto para el usuario.

El caso que se va estudiar en este apartado es aquel en el que se conoce la localización del usuario y la habitación dispone de una interfaz oral. En este punto la aplicación tiene que decidir si emite el mensaje vocal o no. Para ello deberá conocer la actividad en la que está involucrado el usuario. Un primer análisis extrae que las tareas que se realizan cotidianamente en el laboratorio son: investigar, reunirse, leer, descansar, ver una película o tomar café. Ahora bien, en el apartado 4.6.3 se cuando se describía que la actividad del usuario se compone de dos aspectos ortogonales: la finalidad de la actividad y el medio con que se realiza. El primero se engloba dentro de lo que se denominó *ActividadFinalidad* y el segundo dentro de la *ActividadMedio*. La decisión que tome la aplicación tendrá que venir dada en función de ambos aspectos. Las tareas antes identificadas se descomponen de la siguiente manera:

La actividad de *investigar* tiene como finalidad el *Trabajo* y como medio *UsandoOrdenador*. Esta generalización es válida en tanto en cuanto la herramienta principal que se utiliza en el laboratorio para trabajar es el ordenador.

La actividad que inicialmente se identificó como *reunirse* puede tener dos finalidades distintas: una persona se puede reunir bien sea para trabajar o por motivo de ocio. Así, en realidad, son dos actividades distintas, ya que las preferencias del entorno cambian sensiblemente en un caso o en otro. El medio que se emplea para llevar a cabo una reunión en el laboratorio es *Conversando*.

En la actividad de lectura, de nuevo, pueden identificarse dos fines: *Trabajo* y *Ocio*. El medio se corresponde con *Leyendo*.

La actividad *descansar* tiene como fin el *Descansar* y no tiene medio.

La actividad *ver película* tiene como fin el *Ocio* y como medio *Visionando*.

La actividad que se había identificado como *tomar café* no se va a tener en cuenta en el modelo ya que se equipara a una reunión. Ésto es así debido a que la tarea de *tomar café* o bien es una excusa para reunirse a charlar amigablemente[2],

[2]De hecho, se da la paradoja que ir a *tomar café* no tiene por qué implicar consumir café

o bien es una actividad que se desarrolla en segundo plano durante una reunión de trabajo.

Otro aspecto que se definía de cada actividad es el carácter social de la misma. Esta propiedad indica si la persona desarrolla la actividad individualmente o por el contrario existen otras personas involucradas. En este sentido *investigar* se considera siempre una tarea individual, ya que de realizarse combinadamente con otras personas estaríamos en el caso de una reunión. El mismo caso se da con la actividad *Descansar*. *Reunirse* por su propia definición siempre es una actividad social. Finalmente tanto *leer* como *ver una película* pueden ser tanto individual como social.

En la tabla 7.3 se resume la descomposición de las actividades.

	Actividad Finalidad	Actividad Medio	Social
Investigar	Trabajo	UsandoOrdenador	Falso
Reunirse	Trabajo / Ocio	Conversando	Verdadero
Descansar	Descansar		Falso
Leer	Trabajo / Ocio	Leyendo	V/F
Ver película	Ocio	Visionando	V/F

Tabla 7.3: Descomposición de la tareas que se realizan en el laboratorio según el fin y el medio que se emplea

Una parte importante de la aplicación consiste en detectar los cambios de actividad. Este proceso puede implicar agentes de distinta complejidad. En esta primera versión se van a emplear las siguientes heurísticas:

- Si el ordenador del usuario está ocupado entonces está *Trabajando* y *UsandoOrdenador*.
- Si la cafetera se ha encendido, son entre las 15.00 y las 16.00, y hay más de tres personas en el salón entonces esas personas se encuentran *Ociosas* y *Conversando*.
- Si hay más de tres personas en el salón entonces esas personas se encuentran *Trabajando* y *Conversando*.
- Si la televisión está encendida entonces las personas que se encuentran en el salón están *Ociosas* y *Visionando*.
- Si el usuario enciende una luz de lectura entonces el usuario se encuentra *Trabajando* y *Leyendo*.
- En cualquier otro caso la actividad el usuario se encuentra *Descansando*.

En todos los casos se tiene que aplicar la restricción de que el usuario se encuentre en el laboratorio. Ésto implica que salir del laboratorio finaliza cualquier actividad que estuviera realizando.

Ya sean uno o varios agentes los que deduzcan la actividad, se dispondrá de un agente por cada usuario que se encargue de actualizar la propiedad *no_molestar*

en función de la actividad que está desarrollando. Esta propiedad se definió en el apartado 4.6.3 y establece —en el caso de que sea verdadera— que el usuario quiere recibir las menos interrupciones posibles. Finalmente, la aplicación de mensajería decidirá si envía el mensaje vocalmente sólo en el caso de que al usuario no le importe ser molestado.

En el diseño de la aplicación se ha mantenido en lo posible la independencia entre los distintos agentes involucrados. Por un lado, los sensores y productores de contexto que intervienen descubren la actividad del usuario y almacenan sus resultados en la entidad *Actividad* del usuario. A medida que evolucione la aplicación se pueden desarrollar agentes especializados en la detección de actividades concretas que mejoren el reconocimiento. Es más, dado que la *Actividad* tiene dos partes (fin y medio), se pueden desarrollar agentes especializados en la detección de cada una de ellas. Por otro lado, los agentes asociados a cada usuario interpretan de manera particular la información sobre la actividad modificando una variable que condensa la decisión del usuario en cuanto a ser molestado. Toda la información generada por los sensores, productores e interpretes puede ser reutilizada por otras aplicaciones distintas a la planteada.

La aplicación de mensajería también puede ir mejorándose de forma independiente. Por ejemplo, en futuro se puede ir ajustando su comportamiento incluyendo consideraciones de privacidad. La variable *no_molestar* indica que las interrupciones —ya sea de aplicaciones proactivas o de interfaces de usuario— sean las menos posibles, pero no incluye cuestiones como la privacidad del usuario. Así, un usuario puede estar investigando en el laboratorio con otros personas. Aunque el usuario no le importe ser interrumpido cuando está investigando, es posible que sí que le preocupe recibir mensajes vocales cuando hay otras personas presentes.

7.7. Interfaces de usuario

Los demostradores de este apartado consisten en dos interfaces de usuario que han sido integradas como parte del *entorno inteligente*. Estas interfaces constituyen una aplicación compleja que abarca la producción, consumición e interpretación de la información contextual. El desarrollo de estas interfaces ha permitido comprobar como tanto la pizarra como la representación del contexto ha sido beneficiosas para su implementación.

La primera de las interfaces se denomina *Jeoffrey* [14] y ha sido desarrollada por Rubén Cabello, como parte de su trabajo de prácticas de empresa (el equivalente en la UAM al Proyecto Fin de Carrera). *Jeoffrey* consiste en una interfaz gráfica accesible a través de la *red*. La segunda interfaz, *Odisea*, forma parte de la tesis doctoral realizada por Germán Montoro [182], y es un gestor de diálogos que provee una interfaz basada en lenguaje natural.

Jeoffrey y *Odisea* comparten dos aspectos. Primero, ambas interfaces proporcionan una funcionalidad común. Aunque el modo de interacción es sensiblemente distinto, ambas permiten gestionar los dispositivos que se encuentran en un entorno. Segundo, las dos interfaces se configuran dinámicamente al iniciarse, de modo que la presentación de la interfaz se adapta automáticamente a la com-

posición de entorno. Un aspecto notable es cómo se ha conseguido que las dos aplicaciones se intergraron dentro del entorno de forma independiente. Ésto ha sido gracias a la separación que impone la arquitectura de pizarra elegida. Ésta almacena un único modelo que describe el entorno lo cual simplifica sustanciablemente el mantenimiento ambas interfaces.

Tanto *Jeoffrey* y *Odisea* ya han sido parcialmente descritas en los apartados 6.3.1 y 6.3.2 respectivamente, donde se expuso cómo, mediante el lenguaje BBXML, se definen las entidades que representan los dispositivos, incorporando información propietaria de cada interfaz que luego sería empleada para su generación. En este apartado se va hacer hincapié en cómo —tanto *Jeoffrey* como *Odisea*— son capaces de obtener la configuración del entorno empleando los mecanismos que provee la capa de contexto.

7.7.1. Interacción de *Jeoffrey* con la pizarra

En el caso de *Jeoffrey*, cada hogar se subdivide en una serie de habitaciones. Por cada habitación se pueden controlar cero o más dispositivos. Tal como se definió en el contexto primario (véase el apartado 4.6), por cada dispositivo del entorno existe una relación del tipo *contiene* entre la entidad que representa a la habitación y la que representa al recurso. Existe también la relación inversa *estarEn* entre el recurso y la habitación.

Cuando *Jeoffrey* se ejecuta por primera vez realiza una consulta a la pizarra para extraer todos las habitaciones que conforman. Para ello emplea el comando GET (véase el apartado 5.6.3) con la ruta:

Formato de la petición HTTP:

```
http://odisea.ii.uam.es:8080/interact/bb/get/room/*
```

Esta petición devuelve las entidades de tipo habitación definidas en la pizarra.

Con las habitaciones que encuentra muestra una interfaz que permite seleccionar una de ellas. Para cada habitación obtiene la imagen de fondo y el tamaño de la misma. Estos parámetros se encuentran agrupados en un *paramSet* asociado a la entidad que representa la habitación (véase el apartado 6.3.1).

Formato de la petición HTTP:

```
http://odisea.ii.uam.es:8080
/interact/bb/room/lab_b403/params/jeoffrey/* HTTP/1.0
```

Así, para la habitación *lab_b403* se obtiene los parámetros necesarios para dibujar el mapa del entorno (véase la figura 7.15)

Para poder dibujar los dispositivos de los que consta el entorno, se recuperan en una sola consulta todas las entidades que representan a dispositivos del entorno junto con sus parámetros de tipo *Jeoffrey*. Se emplea el comando GET con la siguiente ruta:

```
<GetResponse>
 <entity name="lab_b403">
  <paramType name="jeoffrey">
   <param name="width">764</param>
   <param name="height">513</param>
   <param name="background">fondof.jpg</param>
  </paramType>
 </entity>
</GetResponse>
```

Figura 7.15: Descripción en XML de la respuesta de la pizarra a la consulta /get/room/lab_b403/params/jeoffrey/*

Formato de la petición HTTP:

```
http://odisea.ii.uam.es:8080
/interact/bb/get/roomdevice/lab_b403/*/params/jeoffrey/*
```

La primera parte de la ruta */roomdevice/lab_b403/** especifica que se quieren obtener todos los dispositivos que se encuentren dentro de la habitación *hab1*. Ésto se consigue gracias a una de las rutas predefinidas del espacio de nombres (véase el apartado 4.8). La segunda parte *params/jeoffrey/** incluye en la petición todos los parámetros pertenecientes al conjunto de parámetros *Jeoffrey* en la petición y asociados a la entidad. Una versión simplificada de la respuesta obtenida se puede encontrar en la figura 7.16.

La segunda consulta obtiene de cada dispositivo todas las propiedades genéricas y todos los parámetros de tipo *Jeoffrey* asociados a cada propiedad. Siguiendo el ejemplo, para obtener todas las propiedades de los dispositivos de la habitación primera, la consulta se traduce en un envió del comando GET empleando la ruta */roomdevice/hab1/*/props/*/jeoffrey/**

Formato de la petición HTTP:

```
http://odisea.ii.uam.es:8080
/interact/bb/get/roomdevice/lab_b403/*/props/*/jeoffrey/*
```

El resultado arrojado por la pizarra de los dos primeros dispositivos se muestra en la figura 7.17.

Una vez realizadas las dos consultas, la información de cada dispositivo incluye la representación gráfica para poder pintarlo en la interfaz, y los controles que se emplean para cada propiedad (véase el apartado 6.3.1). Ésto permite generar la imagen completa de la habitación con la ubicación de los dispositivos, y el panel de control particular para cada dispositivo.

Finalmente, *Jeoffrey* se subscribe a los cambios en todas las propiedades del entorno. Para ello debe enviar una petición de subscripción por cada dispositivo

```
<GetResponse>
 <GetResponse>
 <entity name="lab_b403">
  <entity name="lamp_1">
   <paramType name="jeoffrey">
    <param name="image">reflectante.gif</param>
    <param name="x">460</param>
    <param name="y">247</param>
   </paramType>
  </entity>
  <entity name="lamp_2">
   <paramType name="jeoffrey">
    <param name="image">reflectante.gif</param>
    <param name="x">150</param>
    <param name="y">300</param>
   </paramType>
  </entity>
 </entity>
</GetResponse>
```

Figura 7.16: Descripción en XML de la respuesta de la pizarra a la consulta get/roomdevice/lab_b403/*/params/jeoffrey/*

```
<GetResponse>
 <entity name="lab_b403">
  <entity name="lamp_1">
   <property name="status">
    <paramType name="jeoffrey">
     <param name="type">switch</param>
     <param name="text_off">Encender luz</param>
     <param name="cmd_off">1</param>
     <param name="text_on">Apagar luz</param>
     <param name="cmd_on">0</param>
     <param name="color_on">0x00FF00</param>
    </paramType>
   </property>
  </entity>
  <entity name="lamp_2">
   <property name="status">
    <paramType name="jeoffrey"">
     <param name="type">switch</param>
     <param name="text_off">Encender luz</param>
     <param name="cmd_off">1</param>
     <param name="text_on">Apagar luz</param>
     <param name="cmd_on">0</param>
     <param name="color_on">0x00FF00</param>
    </paramType>
   </property>
  </entity>
</entity>
```

Figura 7.17: Descripción en XML de la respuesta de la pizarra a la consulta get/roomdevice/lab_b403/*/props/*/jeoffrey/*

que exista en el entorno en la cual se especifica que desea recibir eventos de todas las propiedades de esa entidad.

7.7.2. Interacción de *Odisea* con la pizarra

Odisea —al igual que *Jeoffrey*— emplea la pizarra durante su inicialización para obtener una representación estática del de los dispositivos del entorno. Las entidades que recupera de la pizarra tienen asociadas parámetros particulares de *Odisea* (véase el apartado 6.3.2) que contienen la información lingüística necesaria para gestionar los diálogos.

Cuando se ejecuta por primera vez, *Odisea* realiza una única consulta empleando la siguiente ruta:

Formato de la petición HTTP:

```
http://odisea.ii.uam.es:8080
/interact/bb/get/roomdevice/lab_b403/*/props/*/dialogue/*
```

Esta consulta le permite obtener todos los dispositivos —junto con los parámetros lingüísticos asociados— que contiene el laboratorio B-403. La respuesta de la pizarra se muestra en la figura 7.18 en la que se expone uno de los dispositivos obtenidos. El *paramSet* de tipo *dialogue* asociado a cada entidad contiene la información necesaria para definir la interfaz oral para cada propiedad de cada dispositivo.

Una vez procesados todos los dispositivos se crea el árbol lingüístico de interacción con el entorno. Este árbol sirve para realizar los procesos de interpretación y generación de los diálogos.

Tras la inicialización, *Odisea* se comunica con la pizarra a medida que precisa información para llevar a cabo un diálogo. Los diálogos cambian de forma dinámica dependiendo del estado de las entidades presentes en el entorno. Así, en cada interacción, la interfaz necesita comunicarse con la pizarra para actualizar la información sobre los dispositivos del entorno.

```
<GetResponse>
 <entity name="lab_b403">
  <entity name="luz_ambiente">
   <property name="status">
     <paramSet name="dialogue">
       <param name="action1">encender</param>
       <param name="skeleton1">VP encender poner OP luz lámpara
                               MP ambiente halógena
       </param>
       <param name="action2">apagar</param>
       <param name="skeleton2">VP apagar quitar OP luz lámpara
                               MP ambiente halógena
       </param>
     </paramSet>
   </property>
   <property name="value">
      <paramSet name="dialogue">
        <param name="action1">subir</param>
        <param name="skeleton1">VP subir aumentar OP luz lámpara
                                MP ambiente halógena
        </param>
        <param name="skeleton2">VP subir aumentar OP intensidad
                                LP luz lámpara MP ambiente halógena
        </param>
        <param name="action2">bajar</param>
        <param name="skeleton3">VP bajar disminuir reducir
                                OP luz lámpara MP ambiente halógena
        </param>
        <param name="skeleton4">VP bajar disminuir OP intensidad
                               LP luz lámpara MP ambiente halógena
        </param>
      </paramSet>
    </property>
  </entity>
</entity>
</GetResponse>
```

Figura 7.18: Descripción en XML de la respuesta de la pizarra a la consulta get/roomdevice/lab_b403/*/props/*/dialogue/*

Capítulo 8

Conclusiones

8.1. Conclusiones

En este trabajo se ha presentado una propuesta para conseguir un comportamiento armonioso entre los componentes de un *entorno inteligente*. Se ha señalado el contexto como ingrediente básico a incluir en el diseño e implementación de las aplicaciones ubicuas, y en concreto de los *entornos inteligentes*. Para comprender esta afirmación hay que tener en cuenta que una de las premisas de la *Computación Ubicua* radica en conseguir una interacción transparente entre la persona y el ordenador. Esta transparencia se mide como el esfuerzo que le supone a la persona interaccionar con el ordenador en relación con el esfuerzo que le supone comunicarse con otra persona. Cuando dos personas participan en una conversación parte de la información que intercambian se comunica explícitamente, mientras que otra parte lo forma el conocimiento implícito que ambos interlocutores tienen de la situación en la que se encuentran. A la información primera se le denomina mensaje, y a la segunda se le denomina contexto. La composición del mensaje y el contexto constituye el conjunto de información transmitida. Un mismo mensaje puede tener distintos significados dependiendo del contexto en que se emita. Por el contrario, en el plano de la comunicación persona-ordenador el tratamiento del contexto es pobre. La comunicación radica casi exclusivamente en el intercambio de información explícita, lo que obliga al usuario a un sobreesfuerzo. El resultado es que el usuario tiene que adaptarse a la computadora, y no a la inversa. Un primer paso para reducir la barrera entre ambos mundos es el tratamiento de la información contextual como parte de la interacción. En el campo de los *entornos inteligentes*, el contexto juega un papel fundamental ya que la interacción se traslada a todo el espacio físico.

Nuestra propuesta se centra en el tratamiento de la información contextual dentro de este tipo de entornos. Este tratamiento se vertebra en dos partes. Por un lado, el estudio y caracterización del contexto (véase el capítulo 4). El objetivo de esta primera parte es determinar la naturaleza de la información contextual, de forma que se pueda encontrar una representación adecuada sobre la que construir nuevas aplicaciones. Por otro lado, el diseño e implementación de una arquitectura para distribuir el contexto entre los componentes del entorno (véase el capítulo 5).

El estudio del contexto parte de la noción que se tiene comúnmente de este concepto (véase el apartado 4.2). Comparándolo con cómo se ha utilizado en las ciencias de la computación (véase el apartado 4.2.1), se extrae que ambos usos se encuentran bastantes distanciados. El problema radica en la carencia de una definición consensuada por toda la comunidad. Dentro del campo de la *Computación Ubicua* se han intentado diversas definiciones de cuyo estudio se puede extraer un conjunto de características de la información contextual que son comunes (véase el apartado 4.3). A la vista de estas cualidades, un primer resultado es la dificultad que supone caracterizar el contexto, ya que, tal como apuntan varios trabajos, éste depende de quién sea el observador. No existe un conjunto de propiedades intrínsecas que permita discriminar qué información se puede catalogar como contexto. Esto arrojaría un modelo en el que sería necesario incluir toda la información disponible por el usuario en un momento dado. Abarcar tal complejidad es imposible, lo cual conlleva una inevitable reducción de la información a incluir. A la hora de discriminar, se ha tomado la decisión de dividir el modelo en dos partes. Una que se ha denominado contexto primario (véase el apartado 4.6) y que incluye las propiedades que más comúnmente han caracterizado al contexto (véase el apartado 4.3); y otra que se ha denominado contexto secundario (véase el apartado 4.7), y que permite extender el contexto primario según las necesidades del dominio. La ventaja de este enfoque radica en poner en primer plano la información más frecuentemente usada como contexto, dejando la posibilidad de expandir el modelo para incluir nuevas características si fueran necesarias.

La elección del mecanismo de representación surge tras estudiar diversas soluciones (véase el apartado 4.5). Se ha elegido una representación en forma de red semántica porque: (a) Es un mecanismo flexible y fácilmente extensible; (b) Se centra en la representación de la organización del mundo, separando la parte algorítmica del modelo de datos. Las elementos que maneja el modelo son conceptos, entidades y relaciones. Los conceptos representan categorías de la realidad. Las entidades son instancias de estas categorías que hacen referencia a componentes del entorno. Finalmente, las relaciones asocian entidades reflejando la estructura de la información contextual en un momento dado. La representación del contexto queda interpretada como un grafo donde los vértices son las distintas entidades que lo conforman, y las aristas son las relaciones que asocian estas entidades.

En relación al contexto primario (véase el apartado 4.6) se han propuesto tres conceptos básicos: Entorno, Persona y Recurso. Sobre estos tres conceptos se han definido una serie de relaciones y propiedades que aportan información sobre la localización, la actividad y la identidad del usuario. Estas tres características son comunes en la mayoría de las *aplicaciones sensibles al contexto* (véase los apartados 2.4.1 y 4.3). Mientras que la localización y la identidad se pueden obtener a menudo directamente de los sensores, la detección de la actividad suele requerir de un proceso de inferencia. En este proceso entra en juego información intermedia que hay que modelar. Así, el modelo incluye una representación de la actividad en dos niveles de abstracción. Por otra parte, existe cierta confusión que tiende a identificar la localización con el contexto, y a este último con el entorno, por lo que se ha realizado un esfuerzo en diferenciar estos tres conceptos (véase el apartado 4.4).

Como ejemplo de contexto secundario se han propuesto dos modelos que trabajan en dominios distintos. El primero de ellos se enfoca al hogar inteligente (véase apartado 4.7.1). Esta extensión trata de modelar los nuevos componentes que aparecerán en el hogar, y las nuevas variables contextuales a tener en cuenta. En el segundo se realiza una aproximación para gestionar flujos de información continuos como información contextual (véase el apartado 4.7.2). Los nuevos conceptos permiten añadir instancias que reflejan la existencia de productores de y consumidores información continua. Las diferentes posibilidades de interconexión entre los componentes también quedan modeladas.

La aportación del modelo propuesto es una representación del contexto independiente de la fuente que lo genera y del nivel de abstracción. Esta representación permite combinar tanto conceptos abstractos como información proveniente de los sensores. Este modelo unificado reduce la complejidad en el tratamiento del contexto por parte de las aplicaciones.

El segundo pilar de esta tesis se halla en la propuesta de la arquitectura de distribución del contexto. Un aspecto clave que se ha tomado en cuenta es que la información contextual para que sea efectiva debe ser compartida por todos los componentes del entorno. Desde nuestro punto de vista, la manera más efectiva se consigue mediante una arquitectura tipo pizarra en la que se combina un modelo global del mundo con un mecanismo asíncrono de comunicación. Esta pizarra contiene toda la información relativa al contexto y componentes del entorno. De esta forma la realización del modelo previo puede ser compartida por todas las aplicaciones. Las modificaciones que se producen en la pizarra se traducen a los dispositivos y aplicaciones mediante un mecanismo de eventos asíncronos.

La metáfora de pizarra lleva a una arquitectura sencilla (véase el apartado 5.3) y fácilmente configurable (véase el apartado 5.3). El proceso de distribución del contexto se ve mejorado en dos sentidos. Por un lado, esta metáfora se basa en que los productores y los consumidores de contexto se encuentran desacoplados tanto temporal, espacial, como funcionalmente. El primero se refiere a que los dos participantes no necesitan estar sincronizados: el productor deposita la información en la pizarra —de donde— posteriormente, el consumidor la recoge. Al quedar almacenada temporalmente en la pizarra, el consumidor no necesita estar ejecutándose cuando el productor deja la información. El desacoplo espacial establece que el productor no necesita conocer quién es el receptor. Cuando un consumidor obtiene información de la pizarra, lo hace de forma anónima para el productor. Por último, el desacoplo funcional evita que el productor tenga que conocer para que usar la información el consumidor. Los tres tipos de desacoplo suponen una ventaja en entornos altamente dinámicos, ya que facilitan añadir o eliminar componentes sin tener que reconfigurar el resto del sistema.

Por otro lado, la segunda mejora que aporta una pizarra es un procesamiento orientado a datos. El paradigma propuesto gira en torno a los datos, cómo representarlos y cómo distribuirlos, y mientras que un paradigma orientado a servicios se centra en el procesamiento. En el primero la capa intermedia provee de primitivas para distribuir eficientemente los datos, en el segundo provee de operaciones que permiten buscar servicios y componer nuevos servicios a partir de lo ya existentes. La caracterización del contexto realizada en el capítulo 4 se basa en proponer

marco común de representación del contexto. Éste se caracteriza por un esquema único, y una realización de este esquema que refleja la composición dinámica del entorno. Se desprende que un paradigma orientado datos se ajusta mejor. Existen otra de serie de ventajas que afectan al diseño y a la implementación. Este tipo de arquitecturas favorecen que el modelo de datos y la funcionalidad evolucionen por separado (véase el apartado 3.4). Además, una vez desplegada una capa intermedia orientada a datos, ésta puede servir como base para una capa intermedia sobre la que se construyan servicios estándar que procesen los datos.

La composición de la arquitectura queda como sigue. Aparte de los ya mencionados productores, y consumidores, se añade un nuevo tipo de componente denominado intérprete. Los tres tipos de componentes se encuentran distribuidos por el entorno, y conocen cómo acceder a la pizarra. Los productores los constituyen dispositivos y módulos software que adquieren la información directamente del mundo físico. Esta información la depositan en la pizarra, donde los intérpretes la recogen. Éstos se encargan de deducir nueva información contextual que dejan también en la pizarra. De forma que los consumidores pueden acceder tanto a la información de los productores como a la de los intérpretes según les convenga. La arquitectura contempla dos posibilidades para conseguir el contexto, bien obteniéndola directamente de la pizarra, bien recibiéndola a través del mecanismo de eventos. Ambas posibilidades están soportadas por un conjunto de operaciones básicas, que son: obtener el valor de una propiedad, una relación o de toda una entidad; modificar el valor de una propiedad o una relación; añadir o eliminar entidades de la pizarra; y, finalmente, añadir o borrar relaciones. A la hora de obtener entidades de la pizarra el espacio de nombres definido (véase el apartado 4.8) permite recuperar en una sola operación varias entidades relacionadas entre sí.

La aportación de esta segunda parte se resume en una capa intermedia denominada capa de contexto. Esta capa de contexto incluye un modelo de datos común soportado por una arquitectura basada en la metáfora de pizarra. Ésta combina el procesamiento orientado a datos y la comunicación asíncrona. Para ello, dispone de un repositorio central, accesible por un conjunto reducido de operaciones, y un mecanismo de eventos asíncronos.

Durante el diseño y la implementación de la capa intermedia han surgido necesidades comunes a cualquier *entorno inteligente*. Así, la capa intermedia ha ido quedando completada con una serie de extensiones que se encargan de dar solución a estos requerimientos.

Un servicio que se incluye frecuentemente en capas intermedias para la *Computación Ubicua* es un histórico de los eventos producidos (véase el apartado 3.4). Guardar los cambios producidos en el contexto es útil para las *aplicaciones sensibles al contexto* (véase el apartado 2.4.1). Nuestra capa de contexto proporciona un mecanismo que almacena los cambios en cualquiera de los componentes del modelo, ya sea una entidad, una propiedad o una relación (véase el apartado 5.8). La arquitectura centralizada simplifica el proceso de registro de cambios, ya que todas las operaciones tienen que pasar por la pizarra.

Otro tema recurrente es el control de acceso a los recursos del entorno. En un entorno donde conviven múltiples aplicaciones y usuarios es necesario disponer

de un mecanismo que prevenga de usos indebidos. Se ha optado por una gestión basada en listas de control de acceso (véase el apartado 5.9). Cada recurso tiene un propietario que es el encargado de decidir los agentes que tienen derecho a utilizar el recurso. Los permisos se establecen mediante una relación entre el recurso y el agente. Esta relación puede indicar que un agente tiene derecho a emplear cierto recurso, o al contrario, que no lo tiene.

Una vez discriminado quién tiene derecho a utilizar un recurso, hay que resolver los conflictos que surgen cuando dos o más agentes autorizados intentan modificar a la vez la información gestionada por la pizarra. Para ello se propone un novedoso mecanismo basado en un esquema centralizado de asignación de prioridades (véase el apartado 5.10.2). Cuando un agente envía una operación, ésta se almacena en una cola asociada. El mecanismo propuesto decide cuál de los comandos se lleva a cabo en primer lugar según:

- Una política de asignación de prioridades que depende de quién es el emisor de la orden; del tipo de orden; de las relaciones entre el emisor y el recurso, y entre el emisor y el entorno; y del estado en que se encuentra el entorno.
- Una política particular que se define para cada recurso que no se ajuste a la política general.

Mientras que la prioridad se asigna automáticamente, los agentes deciden la caducidad de los comandos, para evitar que estos queden indefinidamente esperando en la cola.

Un tema que por su relevancia ha merecido un capítulo separado son las herramientas de apoyo al desarrollo de *entornos inteligentes* (véase el capítulo 6). Se ha propuesto un lenguaje de definición de la pizarra que facilita las labores de implementación. Este lenguaje, denominado BBXML, permite especificar en XML tanto el esquema como la realización de la capa de la contexto. Cada concepto, entidad, propiedad y relación se representa empleando BBXML. Se proporcionan una serie de herramientas que, a partir de la definición de clases y de las instancias iniciales del entorno, son capaces de generar un documento XML que define la configuración del entorno. A partir de este documento, otra herramienta se encarga de crear la pizarra (véase el apartado 6.2.4). Además, la definición de la pizarra a partir de un documento XML facilita el trabajo de documentación, gracias a las herramientas disponibles en el mercado que permiten generar texto con estilo.

Desde el punto de vista de implementación, en el proceso de creación del documento final XML participan tres roles. El *diseñador de recursos* que aporta una descripción XML del concepto que define el recurso. El *administrador del entorno*, el cual es el responsable de decidir qué instancias aparecen en el entorno inicialmente, y cómo se relacionan. Por último, el *diseñador de aplicaciones* que se encarga de desarrollar los servicios e interfaces que tendrán el entorno. Para ello, puede requerir incluir para cada clase de recurso información adicional propietaria de su aplicación. Con la información aportada por los dos primeros diseñadores se obtiene un modelo que es independiente de la capa de aplicación. Cada *diseñador*

de aplicaciones incorpora información necesaria a su aplicación que permite configurar cada instancia que se ejecute de esa aplicación. El lenguaje posibilita añadir esta información extra sin que interfiera con el resto del modelo (véase el apartado 6.2.3).

Este enfoque se ha mostrado útil en la creación automática de interfaces de usuario. Así, se ha podido comprobar con la implementación de *Jeoffrey* (véase el apartado 6.3.1) y *Odisea* (véase el apartado 6.3.2). Cada interfaz se crea a partir del modelo común que descansa en la pizarra, incluyendo información extra necesaria para personalizar la presentación propia de cada modo de interacción. La interfaz, al lanzarse, se adapta a la configuración del entorno, y posteriormente reacciona a las modificaciones en éste al estar subscrita a los cambios en la pizarra. Se ha mostrado cómo con un único modelo se pueden generar múltiples interfaces. La *Computación Ubicua* se basa en que el usuario dispone de multitud de dispositivos a través de los cuales puede interaccionar. Proveer de mecanismos de creación de interfaces dinámica e independientemente del dispositivo, se convierte en un objetivo prioritario. Esta tarea se ve facilitada en gran medida por la posibilidad de compartir entre todas las interfaces un único modelo de datos. Como mecanismo de distribución proponemos el empleo de la metáfora pizarra.

El estudio realizado sobre el tratamiento de la información contextual se ha centrado en el análisis, diseño e implementación de la capa contexto. Esta capa intermedia engloba las ideas planteadas en esta tesis. Una premisa que se impuso desde el principio fue que la arquitectura resultante se probara en un entorno real. De esta forma, en paralelo al desarrollo de la arquitectura se construyó un entorno real que se asemeja a un salón de un hogar (véase el apartado 7.3). Dos redes de comunicación que interconectan diversos dispositivos, conforman la capa física. Tal como se explicó (véase el apartado 3.2.2), se hace necesaria la presencia de las dos redes debido a las diferentes características del tráfico que transportan. Como red de control de dispositivos se ha escogido al bus EIB (véase los apartados 3.2.2 y 7.3.1), mientras que para red multimedia se confía en la red Ethernet (véase los apartados 3.2.2 y 7.3.2). La parte de pruebas queda completada con el desarrollo de un conjunto amplio de aplicaciones que cubren los aspectos más destacados de la tesis. Así, se han implementado aplicaciones sensibles tanto a la identidad (véase los apartados 7.6.2 y 7.6.1) como a la localización del usuario (véase el apartado 7.6.3), así como dos interfaces de usuario (véase el apartado 7.7) que emplean dos modos de interacción diferentes, pero que se basan en el mismo modelo de datos.

8.2. Trabajo futuro

Cualquier tesis, y ésta no quiere ser una excepción, no tiene un punto final, sino que es un puente hacia futuras investigaciones. Dentro del trabajo de investigación desarrollado se han identificado diversos temas donde sería interesante continuar explorando.

En primer lugar la caracterización del contexto se puede ampliar si se piensa en un modelo probabilístico en vez de un modelo determinista. Desde el punto de vista del desarrollador de aplicaciones la información que se obtiene es siempre

cierta, lo cual supone una comodidad para éste. Por otro lado, los datos tienen una precisión y una fiabilidad limitada. El tratamiento de esta incertidumbre, si fuera necesaria, en estos momentos se deja para ser realizado de manera individual por la capa de aplicación. La pizarra actúa como mecanismo de intercambio de información, de modo que todos los sensores relacionados con una misma variable depositan la información en ella. Uno —o varios— módulos *ad-hoc*, que actúan como interpretes, recogen cada uno de los datos depositados y generan un contexto preciso, que vuelven a dejar en la pizarra. Sería interesante estudiar mecanismos de fusión de sensores que se pudieran incluir como parte de la capa de contexto. Esto llevaría a un división del modelo en dos: uno probabilístico, donde se incluiría la incertidumbre generada por los sensores, y otro determinista, que se generaría a partir del anterior, y que coincidiría con el propuesto en esta tesis.

Actualmente la implementación de la pizarra reside en un servidor centralizado. Esta versión se ha mostrado apta para entornos de tamaño medio. Para mejorar el rendimiento y la escalabilidad se puede pensar en una implementación distribuida, pero sin perder la visión lógica centralizada. Desde el punto de vista de la capa de aplicación no se observarían cambios, ya que se mantendría la misma interfaz de programación. La mejoras se llevarían a cabo en la capa intermedia. Ésta quedaría compuesta por una red de pizarras distribuidas donde el contexto se encontraría distribuido. La instancia de una entidad en una pizarra ya no tendría que contener la representación completa, sino que ésta podría sustituirse por un enlace que apuntara a la pizarra donde estuviera almacenada la entidad.

El lenguaje de representación (véase el capítulo 6) propuesto se ha mostrado útil, aunque se han detectado algunos aspectos mejorables. Los recientes avances en tecnologías XML aplicadas a la red semántica sugieren adoptar alguno de los estándares predominantes. Tal como se describió en el apartado 3.3.1, diversos grupos de investigación han integrado en sus capas intermedias lenguajes de representación como RDF y OWL. Estos lenguajes exhiben interesantes características que permiten mejorar la representación de las entidades y las relaciones.

En relación al mecanismo de resolución de conflictos (véase el apartado 5.10), se está estudiando si conviene modificar el cálculo de prioridad para que contemple otros aspectos. Por un lado, se podría añadir el tiempo que un comando ha permanecido en la cola, aumentando la prioridad cuanto más tiempo pase. Por otro lado, se podría incorporar el número de veces que un agente envía un comando, disminuyendo la prioridad de cada sucesiva petición. De esta forma, se evitaría que agentes con una prioridad a priori elevada pudieran acaparar un recurso enviando un gran número de peticiones consecutivas.

Otras líneas de investigación que no han sido abordadas explícitamente en esta tesis son la privacidad y la seguridad. Estos dos temas son críticos para que las personas acepten a los *entornos inteligentes* como lugares de trabajo y convivencia dentro su vida cotidiana. Ambas líneas poseen una larga trayectoria de investigación en las ciencias de la computación, pero la idiosincrasia de los escenarios que surgen en la *Computación Ubicua* conlleva nuevos desafíos tecnológicos. La *Computación Ubicua* se extiende a todo el espacio que ocupa el usuario. Hay que asumir la aparición de nuevos tipos de amenazas y violaciones de la privacidad que no estaban contempladas en los esquemas actuales de seguridad. Por

ejemplo, las cámaras de vídeo pueden ser utilizadas como fuentes de información contextual (véase el apartado 3.2.1). Si se instala una cámara de vídeo en una habitación, es preciso que los ocupantes puedan determinar el grado de resolución de las imágenes, así como ser avisados cuando están siendo grabados. Además, se ha de asegurar que el vídeo que genere la cámara no llegue a aplicaciones malintencionadas.

8.3. Publicaciones a las que ha dado lugar este trabajo

Esta tesis ha sido realizado dentro del marco del proyecto Interact financiado por el Ministerio de Ciencia y Tecnología TIC2000-0464, el cual ha dado como resultado el *entorno inteligente* descrito en el presente documento. Actualmente, los desarrollados realizados están siendo implementados como parte del proyecto *U-CAT* (también financiado por el Ministerio de Ciencia y Tecnología TIN2004-03140). El objetivo del proyecto U-CAT es el desarrollo de un entorno integrado que facilite la realización de actividades educativas desde cualquier lugar mediante la utilización de distintos dispositivos físicos (ordenadores personales, portátiles, teléfonos móviles y PDAs), y en diferentes contextos y situaciones. Este proyecto va a permitir probar la arquitectura propuesta en la tesis en entornos educativos. El modelo del contexto y la arquitectura de pizarra se van a emplear en el proyecto *ITech-Calli* perteneciente a la *4ª Convocatoria de concesión de ayudas para Proyectos de Investigación UAM-Grupo Santander para la Cooperación con América Latina*. En este proyecto se van a emplear tecnologías de capa física distintas a las que se han empleado en Interact, lo cual sólo supondrá desarrollar los controladores necesarios para comunicarse con la pizarra pudiendo reutilizar el resto de la arquitectura.

Las aportaciones y los resultados de esta tesis se condensan en el siguiente conjunto de publicaciones internacionales y nacionales.

8.3.1. Publicaciones internacionales con índice de impacto

- Germán Montoro, Pablo A. Haya and Xavier Alamán. 2004. *Context adaptive interaction with an automatically created spoken interface for intelligent environments.* The 2004 IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM 04). Bangkok, Tailandia. Noviembre 23-26, 2004. Lecture Notes in Computer Science (LNCS). ISSN 0302-9743. Pag. 120-127.

- Pablo A. Haya, Germán Montoro and Xavier Alamán. 2004. *A prototype of a context-based architecture for intelligent home environments.* International Conference on Cooperative Information Systems (CoopIS 2004), Larnaca, Chipre. Octubre 25-29, 2004. Lecture Notes in Computer Science (LNCS). ISSN 0302-9743. Pag. 477-491.

- Germán Montoro, Xavier Alamán and Pablo A. Haya. 2004. *A plug and play spoken dialogue interface for smart environments.* Fifth International Conference on Intelligent Text Processing and Computational Linguistics (CICLing'04). Seoul, Korea. Febrero 15-21, 2004. Lecture Notes in Computer Science (LNCS), volume number 2945. ISSN 0302-9743. Pag. 360-370.

8.3.2. Capítulos de Libro

- Pablo A. Haya and Germán Montoro. 2006. *A spoken interface based on the contextual modelling of smart homes.* HCI related papers of Interacción 2004, Raquel Navarro-Prieto and Jesús Lorés-Vidal, Eds. Springer Verlag. ISBN: 1-4020-4204-3. 2006. Pag. 147-154.

- Germán Montoro, Xavier Alamán and Pablo A. Haya. 2004. *Spoken interaction in intelligent environments: a working system.* Advances in Pervasive Computing. Alois Ferscha, Horst Hoertner and Gabriele Kotsis, Eds. Austrian Computer Society (OCG). ISBN 3-85403-176-9. Pag. 217-222.

8.3.3. Publicaciones en línea

- Pablo A. Haya, Germán Montoro, Abraham Esquivel, Manuel García-Herranz and Xavier Alamán. 2006. *A Mechanism for Solving Conflicts in Ambient Intelligent Environments.* Journal of Universal Computer Science. ISSN 0948-695x. Aceptado para publicación.

- Pablo Haya, Xavier Alamán and Germán Montoro. 2001. *A Comparative Study of Communication Infrastructures for the Implementation of Ubiquitous Computing.* UPGRADE, The European Journal for the Informatics Professional, 2, 5. ISSN 1684-5285. Pag. 26-40.

8.3.4. Conferencias Internacionales

- Pablo A. Haya, Abraham Esquivel, Germán Montoro, Manuel García-Herranz, Xavier Alamán, Ramón Hervás, José Bravo. 2006. *A Prototype of Context Awareness Architecture for Ambience Intelligence at Home.* International Symposium on Intelligent Environments. Cambridge, Reino Unido, Abril, 2006. Aceptado para publicación.

- Pablo A. Haya, Germán Montoro, Xavier Alamán, Rubén Cabello and Javier Martínez. 2004. *Extending an XML environment definition language for spoken dialogue and web-based interfaces.* Developing User Interfaces with XML: Advances on User Interface Description Languages Workshop at AVI04. Gallipoli, Italia. Mayo 25, 2004. Pag. 127-134.

- Germán Montoro, Xavier Alamán and Pablo A. Haya. 2003. *Facts and challenges in a dialogue system for Smart Environments.* 3rd Workshop on Knowledge and Reasoning in Practical Dialog Systems. 18th International

Joint Conference on Artificial Intelligence (IJCAI'03). Acapulco, Mexico. Agosto 9-15, 2003. Pag. 29-34.

- Xavier Alamán, Rubén Cabello, Francisco Gómez-Arriba, Pablo Haya, Antonio Martínez, Javier Martínez, Germán Montoro. 2003. *Using context information to generate dynamic user interfaces.* 10th International Conference on Human-Computer Interaction, HCI International 2003. Creta, Grecia. Junio 22-27, 2003. Pag. 345-349.

8.3.5. Conferencias Nacionales

- Pablo A. Haya, Germán Montoro, and Xavier Alamán. 2005. *Un mecanismo de resolución de conflictos en entornos de Inteligencia Ambiental.* UCAmI 2005, ISBN: 84-9732-442-0. Granada. Septiembre. Pag 11-18.

- Abraham Esquivel, Pablo A. Haya, Germán Montoro, and Xavier Alamán. 2005. *Una propuesta para un modelo de privacidad en entornos activos.* ISBN: 84-9732-436-6 UCAmI 2005, Granada. 13 a 16 Septiembre. Pag. 381-388.

- Germán Montoro, Pablo A. Haya and Xavier Alamán. 2005. *Propuesta de definición de una interfaz dinámica para la interacción con el entorno.* VI Congreso Interacción Persona Ordenador (INTERACCION 2005), Granada. Septiembre, 2005. ISBN: 84-9732-436-6. Pag. 191-198.

- Germán Montoro, Pablo A. Haya and Xavier Alamán. 2004. *Interacción adaptada al contexto en una interfaz de diálogos orales para entornos inteligentes.* III Meeting on speech technology: 35-40. Valencia. November 17-19. Pag. 35-40.

- Germán Montoro, Xavier Alamán and Pablo A. Haya. 2004. *Interacción con entornos inteligentes mediante diálogos basados en contexto.* Conference on Human-Computer Interaction (Interaction 2004). Lleida. Mayo 3-7, 2004.

- Pablo Haya, Xavier Alamán y Germán Montoro. 2003. *El proyecto Interact: el rol de la información contextual.* Conference on Human-Computer Interaction (Interaction 2003). Vigo. Junio 11-13, 2003. Pag. 443-444.

- Xavier Alamán, Pablo Haya and Germán Montoro. 2001. *El proyecto Inter-Act: Una arquitectura de pizarra para la implementación de Entornos Activos.* Conference on Human-Computer Interaction (Interaction 2001). Salamanca. Mayo 16-18, 2001.

- Xavier Alamán, Pablo Haya and Germán Montoro. 2000. *ODISEA: Hacia un entorno inteligente basado en un interfaz en lenguaje natural.* I Meeting on speech technology. Sevilla. Noviembre 6-10, 2000.

- Xavier Alamán, Eloy Anguiano, Fernando Corbacho, Francisco Gómez, Pablo A. Haya, Javier Martínez and Germán Montoro. 2000. *ODISEA: Active Environments for Intelligent Offices and Homes.* First Technical Workshop of the Computer Engineering Department. Universidad Autónoma de Madrid. March 31st. Pag. 10-13.

Apéndice A

Clases correpondientes al contexto primario y secundario

```
<?xml version='1.0'?>
<classes>
    <class name="Raiz"/>

    <class name="Persona" extends="Raiz">
        <property name="nombre_completo" type="string" />
        <property name="apellidos" type="string" />
        <property name="ocupado" type="boolean" />
        <property name="noMolestar" type="boolean" />
        <relation name="estaEn" />
        <relation name="desempeña" />
        <relation name="involucradoEn" />
        <relation name="usa" />
        <relation name="transporta" />
        <relation name="esPropietario" />
    </class>

    <class name="Lugar" extends="Raiz">
        <property name="vacio" type="boolean" />
        <property name="num_ocupantes" type="int" />
        <relation name="contiene" />
        <relation name="estaEn" />
    </class>

    <class name="Entorno" extends="Lugar" >
        <property name="permitidoDefecto" type="boolean" />
        <relation name="tienePropietario" />
        <relation name="tieneAgentePermitido />
        <relation name="tieneAgenteRestringido />
        <relation name="tieneGrupoPermitido />
        <relation name="tieneGrupoRestringido />
```

```
<class/>

<class name="Area" extends="Lugar" />

<class name="Recurso" extends="Raiz">
    <property name="permitidoDefecto" type="boolean" />
    <relation name="tienePropietario" />
    <relation name="estaEn" />
    <relation name="usadoPor" />
    <relation name="transportadoPor" />
    <relation name="tieneAgentePermitido />
    <relation name="tieneAgenteRestringido />
    <relation name="tieneGrupoPermitido />
    <relation name="tieneGrupoRestringido />
</class>

<class name="Union" extends="Raiz">
    <property name="permiteSiemprePaso" type="boolean" />
    <relation name="estaEn" />
</class>

<!-- Actividad -->

<class name="Actividad" extends="Raiz">
    <property name="social" type="boolean" />
    <property name="paraque" type="string" />
    <property name="conque" type="string" />
</class>

<class name="ActividadFinalidad" extends="Raiz" />

<class name="Obligacion" extends="ActividadFinalidad" />
<class name="Trabajo" extends="Obligacion" />
<class name="TareaDomestica" extends="Obligacion" />

<class name="TiempoLibre" extends="ActividadFinalidad" />
<class name="Ocio" extends="TiempoLibre" />
<class name="Descansar" extends="TiempoLibre" />

<class name="CuidadoPersonal" extends="ActividadFinalidad" />
<class name="Aseo" extends="CuidadoPersonal" />
<class name="Medicarse" extends="CuidadoPersonal" />
<class name="Alimentarse" extends="CuidadoPersonal" />

<class name="ActividadMedio" extends="Raiz" />
```

```
<class name="Escribiendo" extends="ActividadMedio" />
<class name="Escuchando" extends="ActividadMedio" />
<class name="Leeyendo" extends="ActividadMedio" />
<class name="Visionando" extends="ActividadMedio" />
<class name="Conversando" extends="ActividadMedio" />
<class name="ConversandoTelefono" extends="Conversando" />
<class name="Conversando" extends="ActividadMedio" />
<class name="UsandoOrdenador" extends="ActividadMedio" />

<!-- Insertar Actividades -->
<class name="Rol" extends="Raiz">
    <relation name="actor" />
</class>

<class name="Usuario" extends="rol" />

<class name="Invitado" extends="rol" />

<!-- Permisos -->
<class name="Grupo" extends="Raiz">
    <relation name="tieneMiembro" />
    <relation name="esPropietario" />
    <relation name="esGrupoPermitido" />
    <relation name="esGrupoRestringido" />
</class>

<class name="Agente" extends="Raiz">
    <property name="estado" type="string" />
    <property name="valor"  type="string" />
    <property name="operativo" type="boolean" />
    <relation type="miembroDe" />
    <relation name="esPropietario" />
    <relation name="esAgentePermitido" />
    <relation name="esAgenteRestringido" />
</class>
<class name="AgenteSeguridad" extends="Agente" />
<class name="AgenteIU" extends="Agente" />


<!-- Hogar -->

<class name="Habitacion" extends="Entorno">
    <relation name="confort" />
    <relation name="seguridad" />
</class>
```

```
<class name="Hogar" extends="Entorno">
    <relation name="seguridad" />
</class>

<class name="Edificio" extends="Entorno">
    <relation name="seguridad" />
</class>

<class name="Confort" extends="Raiz">
    <relation name="nivelRuidoActual" />
    <relation name="luminosidadActual" />
    <relation name="temperaturaActual" />
</class>

<class name="NivelRuido">
    <property name="valor" type="string" />
    <property name="valorDeseado" type="string" />
</class>

<class name="Temperatura">
    <property name="valor" type="real" />
    <property name="valorDeseado" type="real" />
    <property name="unidad" type="string" />
</class>

<class name="Luminosidad">
    <property name="valor" type="real" />
    <property name="valorDeseado" type="real" />
    <property name="unidad" type="string" />
</class>

<!-- Hogar / Seguridad -->

<class name="Seguridad" extends="Raiz">
    <property name="gradoSeguridad" type="string" />
    <property name="estado" type="string" />
    <relation name="alarma" />
</class>

<class name="Alerta">
    <property name="descripcion" type="string" />
    <relation name="localizadaEn" />
</class>

<class name="AlertaEntorno" extends="Alerta" />
<class name="Intrusion" extends="AlertaEntorno" />
```

```
<class name="Incendio" extends="AlertaEntorno" />
<class name="Inundacion" extends="AlertaEntorno" />
<class name="EscapeGas" extends="AlertaEntorno" />

<class name="AlertaFuncionamiento" extends="Alerta">
    <relation name="recursoAfectado" />
</class>

<class name="AlertaRecurso" extends="Alerta">
    <relation name="recursoAfectado" />
</class>
<class name="Robo" extends="AlertaRecurso" />
<class name="Desperfecto" extends="AlertaRecurso" />

<class name="AlertaPersona" extends="Alerta">
    <relation name="personaAfectada" />
</class>

<!-- Dispositivos -->
<class name="Dispositivo" extends="Recurso">
    <property name="estado" type="string" />
    <property name="valor"  type="string" />
    <property name="ocupado" type="boolean" />
    <property name="operativo" type="boolean" />
    <property name="potencia" type="int" />
</class>

<class name="DispositivoSalida" extends="Dispositivo" />

<class name="Pantalla" extends="DispositivoSalida" />
<class name="Altavoz" extends="DispositivoSalida" />

<class name="Actuador" extends="DispositivoSalida" />
<class name="Electrodomestico" extends="Actuador" />
<class name="Lavadora" extends="Electrodomestico" />
<class name="Lavaplatos" extends="Electrodomestico" />
<class name="Lavadora" extends="Electrodomestico" />
<class name="Cafetera" extends="Electrodomestico" />
<class name="Microondas" extends="Electrodomestico" />
<class name="Horno" extends="Electrodomestico" />
<class name="Frigorifico" extends="Electrodomestico" />
<class name="Secadora" extends="Electrodomestico" />
<class name="Aspiradora" extends="Electrodomestico" />
<class name="Congelador" extends="Electrodomestico" />
<class name="Plancha" extends="Electrodomestico" />
```

```
<class name="Cerradura" extends="Actuador" />
<class name="AireAcondicionado" extends="Actuador" />
<class name="Persiana" extends="Actuador" />
<class name="Calefaccion" extends="Actuador" />
<class name="RiegoAutomatico" extends="Actuador" />
<class name="Electrovalvula" extends="Actuador" />
<class name="Luz" extends="Actuador" />
<class name="LuzRegulable" extends="Actuador" />
<class name="FuenteAlimentacion" extends="Actuador" />

<class name="DispositivoEntrada" extends="Dispositivo" />

<class name="Microfono" extends="DispositivoEntrada" />
<class name="ServidorAudio" extends="DispositivoEntrada" />
<class name="VideoCamara" extends="DispositivoEntrada" />
<class name="ServidorVideo" extends="DispositivoEntrada" />
<class name="Teclado" extends="DispositivoEntrada" />
<class name="Raton" extends="DispositivoEntrada" />
<class name="LectorTarjeta" extends="DispositivoEntrada" />

<class name="Sensor" extends="DispositivoEntrada" />
<class name="FugaGas" extends="Sensor" />
<class name="Luminosidad" extends="Sensor" />
<class name="Humo" extends="Sensor" />
<class name="Movimiento" extends="Sensor" />
<class name="Inundacion" extends="Sensor" />
<class name="SensorPuerta" extends="Sensor" />
<class name="Termoestato" extends="Sensor" />
<class name="SensorVentana" extends="Sensor" />
<class name="Interruptor" extends="Sensor" />
<class name="Temperatura" extends="Sensor" />
<class name="Incendio" extends="Sensor" />
<class name="CaidaTension" extends="Sensor" />
<class name="LlamadaTelefono" extends="Sensor" />
<class name="DetectorSeñalVideo" extends="Sensor" />
<class name="DetectorSeñalAudio" extends="Sensor" />
<class name="Presion" extends="Sensor" />
<class name="Regulador" extends="Sensor" />
<class name="TemperaturaAgua" extends="Sensor" />
<class name="Anemometro" extends="Sensor" />

<!-- Flujos de multimedia -->
<class name="Productor" extends="Raiz">
    <property name="conexionPermitidaDefecto" type="boolean" />
    <property name="maxNumConexiones" type="int" />
    <relation name="estaConectadoA" />
```

```
        <relation name="conexionPermitida" />
        <relation name="conexionDenegada" />
    </class>

    <class name="ProductorAudio" extends="Productor" />
    <class name="ProductorVideo" extends="Productor" />
    <class name="ProductorAudioVisual" extends="Productor" />
    <class name="ProductorImagen" extends="Productor" />

    <class name="Consumidor" extends="Raiz" />

    <class name="ConsumidorAudio" extends="Consumidor" />
    <class name="ConsumidorVideo" extends="Consumidor" />
    <class name="ConsumidorAudioVisual" extends="Consumidor" />
    <class name="ConsumidorImagen" extends="Consumidor" />

</classes>
```

Apéndice B

Instancias de componentes del laboratorio B-403

```
<instances>
 <!-- Entidad Laboratorio B403 -->
 <entity name="lab_B403" type="Room">
  <property name="habitants">
   <paramSet name="default">
    <param name="location">0</param>
    <param name="access">3</param>
    <param name="initialValue">0</param>
   </paramSet>
  </property>
  <property name="empty">
   <paramSet name="default">
    <param name="location">0</param>
    <param name="access">2</param>
    <param name="initialValue">0</param>
   </paramSet>
  </property>
  <property name="permitidoDefecto">
   <paramSet name="default">
    <param name="location">0</param>
    <param name="access">2</param>
    <param name="initialValue">0</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">Laboratorio de Entornos Inteligentes y Computación Ubicua</pa
  </paramSet>
  <paramSet name="jeoffrey">
    <param name="width">764</param>
    <param name="height">513</param>
```

```
    <param name="background">fondof.jpg</param>
  </paramSet>
  <paramSet name="jeoffreyPDA">
    <param name="width">181</param>
    <param name="height">270</param>
    <param name="background">fondofPDA.jpg</param>
  </paramSet>
  <!-- Dispositivos dentro de la habitación B403 -->
  <relation name="contains" destination="Lamp_1" type="Light" />
  <relation name="contains" destination="Lamp_2" type="Light" />
  <relation name="contains" destination="Switch0_1" type="Actuator" />
  <relation name="contains" destination="Switch1_1" type="Actuator" />
  <relation name="contains" destination="SwitchX_1" type="Actuator" />
  <relation name="contains" destination="SwitchX_2" type="Actuator" />
  <relation name="contains" destination="Led_izq_1" type="Led" />
  <relation name="contains" destination="Led_der_1" type="Led" />
  <relation name="contains" destination="LampV1" type="DimmableLight" />
  <relation name="contains" destination="LampV2" type="DimmableLight" />
  <relation name="contains" destination="Puerta1" type="Lock" />
  <relation name="contains" destination="Detector_Puerta" type="DoorSensor" />
  <relation name="contains" destination="Altavoz" type="Speaker" />
  <relation name="contains" destination="Altavoz2" type="Speaker" />
  <relation name="contains" destination="Genmusic" type="AudioServer" />
  <relation name="contains" destination="Video" type="VideoServer" />
  <relation name="contains" destination="Tarjeta" type="CardReader" />
  <relation name="contains" destination="eib3" type="Plug" />
  <relation name="contains" destination="eib6" type="Plug" />
  <relation name="contains" destination="eib5" type="Plug" />
  <relation name="contains" destination="synt" type="Synthetizer" />
  <relation name="contains" destination="syntmanu" type="Synthetizer" />
  <relation name="contains" destination="abraxas" type="PC" />
  <relation name="contains" destination="monolito" type="PC" />
  <relation name="contains" destination="robinson" type="PC" />
  <relation name="contains" destination="kubrick" type="PC" />
  <relation name="security" destination="security1" type="Security" />
  <relation name="tieneAgentePermitido" destination="manu" type="Person" />
  <relation name="tieneAgentePermitido" destination="phaya" type="Person" />
  <relation name="tieneAgentePermitido" destination="german" type="Person" />
  <relation name="tieneAgentePermitido" destination="paco" type="Person" />
  <relation name="tieneAgentePermitido" destination="abraham" type="Person" />
  <relation name="tienePropietario" destination="xavier" type="Person" />
</entity>

<entity name="security1" type="Security" />
```

```
<!-- Entidad Luz del salón -->
 <entity name="Lamp_1" type="Light">
  <property name="Status">
   <paramSet name="default">
    <param name="location">1</param>
    <param name="access">2</param>
    <param name="initialValue">0</param>
   </paramSet>
   <paramSet name="jeoffrey">
     <param name="type">switch</param>
     <param name="text_off">Encender&#x20;luz</param>
     <param name="cmd_off">1</param>
     <param name="text_on">Apagar&#x20;luz</param>
     <param name="cmd_on">0</param>
    <param name="color_on">0x00FF00</param>
   </paramSet>
   <paramSet name="jeoffreyPDA" >
     <param name="type">switch</param>
     <param name="text_off">Encender&#x20;luz</param>
     <param name="cmd_off">1</param>
     <param name="text_on">Apagar&#x20;luz</param>
     <param name="cmd_on">0</param>
    <param name="color_on">0x00FF00</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">Luces&#x20;del&#x20;salón</param>
  </paramSet>
  <paramSet name="snmp">
   <param name="DeviceAddress">0801</param>
   <param name="DeviceIndex">1</param>
  </paramSet>
  <paramSet name="jeoffrey">
   <param name="image">reflectante.gif</param>
   <param name="x">460</param>
   <param name="y">247</param>
  </paramSet>
 </entity>

  <!-- Entidad Luz del oficina -->
 <entity name="Lamp_2" type="Light">
  <property name="Status">
   <paramSet name="default">
    <param name="access">2</param>
    <param name="location"  >1</param>
    <param name="initialValue"   >0</param>
```

```
  </paramSet>
  <paramSet name="jeoffrey">
    <param name="type">switch</param>
    <param name="text_off">Encender&#x20;luz</param>
    <param name="cmd_off">1</param>
    <param name="text_on">Apagar&#x20;luz</param>
    <param name="cmd_on">0</param>
   <param name="color_on">0x00FF00</param>
  </paramSet>
  <paramSet name="jeoffreyPDA">
    <param name="type">switch</param>
    <param name="text_off">Encender&#x20;luz</param>
    <param name="cmd_off">1</param>
    <param name="text_on">Apagar&#x20;luz</param>
    <param name="cmd_on">0</param>
   <param name="color_on">0x00FF00</param>
  </paramSet>
 </property>
 <paramSet name="default">
  <param name="desc">Luces&#x20;de&#x20;la&#x20;oficina</param>
  <param name="type">Lamp</param>
 </paramSet>
 <paramSet name="snmp">
  <param name="DeviceAddress">0802</param>
  <param name="DeviceIndex">2</param>
 </paramSet>
 <paramSet name="jeoffrey">
  <param name="image">reflectante.gif</param>
  <param name="x">150</param>
  <param name="y">300</param>
 </paramSet>
</entity>

<!-- Entidad Lampara V1 -->
<entity name="LampV1" type="DimmableLight">
 <property name="Status">
  <paramSet name="default">
   <param name="access">2</param>
   <param name="location"  >1</param>
   <param name="initialValue"   >0</param>
  </paramSet>
  <paramSet name="jeoffrey">
    <param name="type">switch</param>
    <param name="text_off">Encender&#x20;luz</param>
    <param name="cmd_off">1</param>
    <param name="text_on">Apagar&#x20;luz</param>
```

```
    <param name="cmd_on">0</param>
   <param name="color_on">0x00FF00</param>
  </paramSet>
  <paramSet name="jeoffreyPDA">
    <param name="type">switch</param>
    <param name="text_off">Encender&#x20;luz</param>
    <param name="cmd_off">1</param>
    <param name="text_on">Apagar&#x20;luz</param>
    <param name="cmd_on">0</param>
   <param name="color_on">0x00FF00</param>
  </paramSet>
 </property>
 <property name="Value">
  <paramSet name="default">
   <param name="type">1A</param>
   <param name="access">2</param>
   <param name="location"  >1</param>
   <param name="initialValue"   >31</param>
  </paramSet>
  <paramSet name="jeoffrey">
    <param name="type">slider</param>
    <param name="lo">0</param>
    <param name="hi">64</param>
    <param name="unit">4</param>
  </paramSet>
 </property>
 <paramSet name="default">
  <param name="desc">LamparaV1</param>
  <param name="type">Variable_Lamp</param>
 </paramSet>
 <paramSet name="snmp">
  <param name="DeviceAddress">0830</param>
  <param name="DeviceIndex">11</param>
 </paramSet>
 <paramSet name="jeoffrey">
  <param name="image">lampPie.gif</param>
  <param name="x">655</param>
  <param name="y">130</param>
 </paramSet>
</entity>

<!-- Entidad Lampara V2 -->
 <entity name="LampV2" type="DimmableLight">
  <property name="Status">
   <paramSet name="default">
    <param name="access">2</param>
```

```
    <param name="location"  >1</param>
    <param name="initialValue"   >0</param>
   </paramSet>
   <paramSet name="jeoffrey">
     <param name="type">switch</param>
     <param name="text_off">Encender&#x20;luz</param>
     <param name="cmd_off">1</param>
     <param name="text_on">Apagar&#x20;luz</param>
     <param name="cmd_on">0</param>
    <param name="color_on">0x00FF00</param>
   </paramSet>
  </property>
  <property name="Value">
   <paramSet name="default">
    <param name="type">1A</param>
    <param name="access">2</param>
    <param name="location">1</param>
    <param name="initialValue">31</param>
   </paramSet>
   <paramSet name="jeoffrey">
     <param name="type">slider</param>
     <param name="lo">0</param>
     <param name="hi">64</param>
     <param name="unit">4</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">LamparaV2</param>
   <param name="type">Variable_Lamp</param>
  </paramSet>
  <paramSet name="snmp">
   <param name="DeviceAddress">0831</param>
   <param name="DeviceIndex">12</param>
  </paramSet>
  <paramSet name="jeoffrey">
   <param name="image">lampPie.gif</param>
   <param name="x">655</param>
   <param name="y">310</param>
  </paramSet>
  <paramSet name="jeoffreyPDA">
   <param name="image">lampPiePDA.gif</param>
   <param name="x">50</param>
   <param name="y">225</param>
  </paramSet>
 </entity>
```

```
<!-- Entidad Puerta 1 -->
<entity name="Puerta1" type="Lock">
 <property name="Status">
  <paramSet name="default">
   <param name="access">1</param>
   <param name="location"  >1</param>
   <param name="initialValue"   >0</param>
  </paramSet>
  <paramSet name="jeoffrey">
   <param name="type">button</param>
   <param name="text">Abrir&#x20;puerta</param>
   <param name="cmd">1</param>
   <param name="timeout">10000</param>
   <param name="color">0x00FF00</param>
  </paramSet>
 </property>
 <paramSet name="snmp">
  <param name="DeviceAddress">0804</param>
  <param name="DeviceIndex">13</param>
 </paramSet>
 <paramSet name="jeoffrey">
  <param name="image">puerta.gif</param>
  <param name="x">520</param>
  <param name="y">30</param>
 </paramSet>
</entity>

<!-- Entidad Detector de la puerta -->
<entity name="Detector_Puerta" type="DoorSensor">
 <property name="Value">
  <paramSet name="default">
   <param name="access">0</param>
   <param name="location"  >1</param>
   <param name="initialValue"   >0</param>
  </paramSet>
  <paramSet name="jeoffrey">
   <param name="type">alarm</param>
   <param name="entry">1&#x20;0xFF0000</param>
   <param name="entry">0&#x20;0x000000</param>
  </paramSet>
 </property>
 <paramSet name="default">
  <param name="desc">Detector&#x20;de&#x20;la&#x20;puerta</param>
  <param name="type">Detector_Puerta</param>
 </paramSet>
 <paramSet name="snmp">
```

```
  <param name="DeviceAddress">0818</param>
  <param name="DeviceIndex">15</param>
 </paramSet>
 <paramSet name="jeoffrey">
  <param name="image">pcerradas.jpg</param>
  <param name="x">589</param>
  <param name="y">54</param>
 </paramSet>
</entity>

<!-- Entidad Lector de tarjeta -->
<entity name="Tarjeta" type="CardReader">
 <property name="Tarjeta_Leida">
  <paramSet name="default">
   <param name="access">0</param>
   <param name="location">0</param>
   <param name="initialValue"></param>
  </paramSet>
  <paramSet name="jeoffrey">
   <param name="type">entry</param>
   <param name="timeout">10000</param>
  </paramSet>
 </property>
 <paramSet name="jeoffrey">
  <param name="image">tarjeta.gif</param>
  <param name="x">490</param>
  <param name="y">68</param>
 </paramSet>
</entity>


<!-- Entidad Altavoz 1-->
<entity name="Altavoz" type="Speaker">
 <property name="Habilitar">
  <paramSet name="default">
   <param name="access">2</param>
   <param name="location">1</param>
   <param name="initialValue">0</param>
  </paramSet>
  <paramSet name="jeoffrey">
   <param name="dependences">1</param>
   <param name="type">switch</param>
   <param name="text_on" >Apagar</param>
   <param name="text_off" >Encender</param>
   <param name="cmd_on" >0</param>
   <param name="cmd_off" >1</param>
```

```
  <param name="color_on">0x00FF00</param>
 </paramSet>
</property>
<property name="Ready" >
 <paramSet name="default">
  <param name="access">2</param>
  <param name="location" >1</param>
  <param name="initialValue" >0</param>
 </paramSet>
 <paramSet name="jeoffrey" >
  <param name="type" >alarm</param>
  <param name="entry" >1&#x20;0x00FF00</param>
  <param name="entry" >0&#x20;0xFF0000</param>
 </paramSet>
</property>
<property name="Status" >
 <paramSet name="default">
  <param name="access">0</param>
  <param name="location"  >1</param>
  <param name="initialValue"   >0</param>
 </paramSet>
</property>
<property name="Puerto" >
 <paramSet name="default">
  <param name="access">3</param>
  <param name="location" >1</param>
  <param name="initialValue" >9006</param>
 </paramSet>
 <paramSet name="jeoffrey" >
  <param name="enable" >1.0</param>
  <param name="type" >entry</param>
 </paramSet>
</property>
<property name="IP" >
 <paramSet name="default">
  <param name="access">2</param>
  <param name="location"  >1</param>
  <param name="initialValue"   >225.0.0.1</param>
 </paramSet>
 <paramSet name="jeoffrey" >
  <param name="enable" >1.0</param>
  <param name="type" >entry</param>
 </paramSet>
</property>
<property name="Velocidad" >
 <paramSet name="default">
```

```
  <param name="type">1A</param>
  <param name="access">3</param>
  <param name="location"  >1</param>
  <param name="initialValue"   >64s</param>
 </paramSet>
 <paramSet name="jeoffrey" >
  <param name="enable" >1.0</param>
  <param name="type" >choice</param>
  <param name="entry" >64kbit_stereo&#x20;64s</param>
  <param name="entry" >64kbit_mono&#x20;64m</param>
  <param name="entry" >128kbit_stereo&#x20;128s</param>
  <param name="entry" >128kbit_mono&#x20;128m</param>
</paramSet>
</property>
<property name="Left_Volume" >
 <paramSet name="default">
  <param name="access">2</param>
  <param name="location"  >1</param>
  <param name="initialValue"   >50</param>
 </paramSet>
 <paramSet name="jeoffrey" >
  <param name="type" >slider</param>
  <param name="lo" >0</param>
  <param name="hi" >100</param>
  <param name="unit" >5</param>
 </paramSet>
</property>
<property name="Right_Volume" >
 <paramSet name="default">
  <param name="access">2</param>
  <param name="location"  >1</param>
  <param name="initialValue"   >50</param>
 </paramSet>
 <paramSet name="jeoffrey" >
  <param name="type" >slider</param>
  <param name="lo" >0</param>
  <param name="hi" >100</param>
  <param name="unit" >5</param>
 </paramSet>
</property>
<property name="Graves" >
 <paramSet name="default">
  <param name="access">2</param>
  <param name="location"  >1</param>
  <param name="initialValue"   >50</param>
 </paramSet>
```

```
 <paramSet name="jeoffrey" >
  <param name="type" >slider</param>
  <param name="lo" >0</param>
  <param name="hi" >100</param>
  <param name="unit" >5</param>
 </paramSet>
</property>
<property name="Agudos" >
 <paramSet name="default">
  <param name="access">2</param>
  <param name="location"  >1</param>
  <param name="initialValue"   >50</param>
 </paramSet>
 <paramSet name="jeoffrey" >
  <param name="type" >slider</param>
  <param name="lo" >0</param>
  <param name="hi" >100</param>
  <param name="unit" >5</param>
 </paramSet>
</property>
<property name="Master_Volume" >
 <paramSet name="default">
  <param name="access">2</param>
  <param name="location"  >0</param>
  <param name="initialValue"   >50</param>
 </paramSet>
 <paramSet name="jeoffrey" >
  <param name="type" >link</param>
  <param name="id" >7</param>
  <param name="id" >8</param>
 </paramSet>
</property>
<paramSet name="default">
 <param name="desc">Altavoz</param>
</paramSet>
<paramSet name="snmp">
 <param name="DeviceAddress">2192</param>
 <param name="DeviceIndex">1</param>
</paramSet>
<paramSet name="jeoffrey" >
 <param name="image">altavoz.gif</param>
 <param name="x">400</param>
 <param name="y">185</param>
</paramSet>
</entity>
```

```
<!-- Genmusic -->

<!-- Entidad Genmusic -->
<entity name="Genmusic" type="AudioServer">
 <property name="Fuente">
  <paramSet name="default">
   <param name="access">2</param>
   <param name="location"  >1</param>
   <param name="initialValue"   >NADA</param>
  </paramSet>
  <paramSet name="jeoffrey" >
   <param name="type" >choice</param>
   <param name="entry" >Off&#x20;NADA</param>
   <param name="entry" >Lector_cd&#x20;LECTOR_CD</param>
   <param name="entry" >Radio&#x20;RADIO</param>
   <param name="dependences" >1</param>
  </paramSet>
 </property>
 <property name="Puerto" >
  <paramSet name="default">
   <param name="access">2</param>
   <param name="location"  >1</param>
   <param name="initialValue"   >9006</param>
  </paramSet>
  <paramSet name="jeoffrey" >
   <param name="type" >entry</param>
   <param name="enable" >1.NADA</param>
  </paramSet>
 </property>
 <property name="IP" >
  <paramSet name="default">
   <param name="access">3</param>
   <param name="location"  >1</param>
   <param name="initialValue"   >225.0.0.1</param>
  </paramSet>
  <paramSet name="jeoffrey" >
   <param name="type" >entry</param>
   <param name="enable" >1.NADA</param>
  </paramSet>
 </property>
 <property name="Volumen_CD" >
 <paramSet name="default">
   <param name="access">2</param>
   <param name="location"  >1</param>
   <param name="initialValue"   >90</param>
  </paramSet>
```

```
 <paramSet name="jeoffrey" >
  <param name="type" >slider</param>
  <param name="lo" >0</param>
  <param name="hi" >100</param>
  <param name="unit" >5</param>
  <param name="enable" >1.LECTOR_CD</param>
 </paramSet>
</property>
<property name="Status_CD" >
 <paramSet name="default">
  <param name="access">0</param>
  <param name="location"  >1</param>
  <param name="initialValue"   >0</param>
 </paramSet>
 <paramSet name="jeoffrey" >
  <param name="type" >alarm</param>
  <param name="entry" >-1&#x20;0xFFFFFF</param>
  <param name="entry" >0&#x20;0xFF0000</param>
  <param name="entry" >1&#x20;0x00FF00</param>
  <param name="entry" >2&#x20;0x000000</param>
  <param name="entry" >3&#x20;0x0000FF</param>
  <param name="enable" >1.LECTOR_CD</param>
 </paramSet>
</property>
<property name="Escuchar_track_unico" >
 <paramSet name="default">
  <param name="access">1</param>
  <param name="location"  >1</param>
  <param name="initialValue"   >0</param>
 </paramSet>
 <paramSet name="jeoffrey" >
  <param name="type" >entry</param>
  <param name="enable" >1.LECTOR_CD</param>
 </paramSet>
</property>
<property name="Escuchar_track_aleatorio" >
 <paramSet name="default">
  <param name="type">1</param>
  <param name="access">3</param>
  <param name="location"  >1</param>
  <param name="initialValue"   >0</param>
 </paramSet>
 <paramSet name="jeoffrey" >
  <param name="type" >entry</param>
  <param name="enable" >1.LECTOR_CD</param>
 </paramSet>
```

```
</property>
<property name="Escuchar_a_partir_de_track" >
 <paramSet name="default">
  <param name="access">1</param>
  <param name="location"  >1</param>
  <param name="initialValue"   >0</param>
 </paramSet>
 <paramSet name="jeoffrey" >
  <param name="type" >entry</param>
  <param name="enable" >1.LECTOR_CD</param>
 </paramSet>
</property>
<property name="Escucha_continua" >
 <paramSet name="default">
  <param name="access">1</param>
  <param name="location"  >1</param>
  <param name="initialValue"   >0</param>
 </paramSet>
 <paramSet name="jeoffrey" >
  <param name="type" >entry</param>
  <param name="enable" >1.LECTOR_CD</param>
 </paramSet>
</property>
<property name="Pausa" >
 <paramSet name="default">
  <param name="access">1</param>
  <param name="location"  >1</param>
  <param name="initialValue"   >0</param>
 </paramSet>
 <paramSet name="jeoffrey" >
  <param name="type" >button</param>
  <param name="text" >Pausa</param>
  <param name="cmd" >1</param>
  <param name="enable" >1.LECTOR_CD</param>
 </paramSet>
</property>
<property name="Continuar" >
 <paramSet name="default">
  <param name="access">1</param>
  <param name="location"  >1</param>
  <param name="initialValue"   >0</param>
 </paramSet>
 <paramSet name="jeoffrey" >
  <param name="type" >button</param>
  <param name="text" >Continuar</param>
  <param name="cmd" >1</param>
```

```
    <param name="enable" >1.LECTOR_CD</param>
   </paramSet>
  </property>
 <property name="Volumen_Radio" >
  <paramSet name="default">
    <param name="access">2</param>
    <param name="location"  >1</param>
    <param name="initialValue"   >60</param>
   </paramSet>
   <paramSet name="jeoffrey" >
    <param name="type" >slider</param>
    <param name="lo" >0</param>
    <param name="hi" >100</param>
    <param name="unit" >5</param>
    <param name="enable" >1.RADIO</param>
   </paramSet>
  </property>
  <property name="Canal" ">
  <paramSet name="default">
    <param name="access">2</param>
    <param name="location"  >1</param>
    <param name="initialValue"   >11</param>
   </paramSet>
   <paramSet name="jeoffrey" >
    <param name="type" >choice</param>
    <param name="entry" >M80&#x20;0</param>
    <param name="entry" >Alcobendas&#x20;1</param>
    <param name="entry" >Radio5&#x20;2</param>
    <param name="entry" >Los_40&#x20;3</param>
    <param name="entry" >Radio3&#x20;4</param>
    <param name="entry" >Top40&#x20;5</param>
    <param name="entry" >Onda0&#x20;6</param>
    <param name="entry" >Cadena100&#x20;7</param>
    <param name="entry" >Cope&#x20;8</param>
    <param name="entry" >Onda_Madrid&#x20;9</param>
    <param name="entry" >Kiss_FM&#x20;10</param>
    <param name="entry" >Maxima&#x20;11</param>
    <param name="entry" >Radio1&#x20;12</param>
    <param name="entry" >SER&#x20;13</param>
    <param name="enable" >1.RADIO</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">Genmusic</param>
  </paramSet>
  <paramSet name="snmp">
```

```
   <param name="DeviceAddress">2192</param>
   <param name="DeviceIndex">3</param>
  </paramSet>
  <paramSet name="jeoffrey" >
   <param name="image">cadena.jpg</param>
   <param name="x">390</param>
   <param name="y">315</param>
  </paramSet>
 </entity>

<!-- Entidad Interruptor arriba izquierdo -->
 <entity name="switch0_1" type="Plug">
  <property name="value">
   <paramSet name="default">
    <param name="location"  >1</param>
    <param name="access">0</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">Interruptor&#x20;arriba&#x20;izquierda</param>
  </paramSet>
  <paramSet name="snmp">
   <param name="DeviceAddress">080A</param>
   <param name="DeviceIndex">3</param>
  </paramSet>
 </entity>

<!-- Entidad Interruptor abajo izquierdo -->
 <entity name="switch1_1" type="Plug">
  <property name="value">
   <paramSet name="default">
    <param name="location"  >1</param>
    <param name="access">0</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">Interruptor&#x20;abajo&#x20;izquierda</param>
  </paramSet>
  <paramSet name="snmp">
   <param name="DeviceAddress">080B</param>
   <param name="DeviceIndex">4</param>
  </paramSet>
 </entity>

<!-- Entidad Interruptor arriba derecho -->
 <entity name="switchx_1" type="Plug">
```

```
  <property name="value">
   <paramSet name="default">
    <param name="location"  >1</param>
    <param name="access">0</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">Interruptor&#x20;arriba&#x20;derecha</param>
  </paramSet>
  <paramSet name="snmp">
   <param name="DeviceAddress">080D</param>
   <param name="DeviceIndex">5</param>
  </paramSet>
 </entity>

<!-- Entidad Interruptor abajo derecho -->
 <entity name="switchx_2" type="Plug">
  <property name="value">
   <paramSet name="default">
    <param name="location"  >1</param>
    <param name="access">0</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">Interruptor&#x20;abajo&#x20;derecha</param>
  </paramSet>
  <paramSet name="snmp">
   <param name="DeviceAddress">080D</param>
   <param name="DeviceIndex">6</param>
  </paramSet>
 </entity>

<!-- Entidad Led izquierdo -->
 <entity name="Led_izq_1" type="Led">
  <property name="value">
   <paramSet name="default">
    <param name="location"  >1</param>
    <param name="access">2</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">Led&#x20;izquierda</param>
  </paramSet>
  <paramSet name="snmp">
   <param name="DeviceAddress">080E</param>
   <param name="DeviceIndex">7</param>
```

```
  </paramSet>
 </entity>

<!-- Entidad Led derecho -->
 <entity name="Led_der_1" type="Led">
  <property name="value">
   <paramSet name="default">
    <param name="location"  >1</param>
    <param name="access">2</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">Led&#x20;derecha</param>
  </paramSet>
  <paramSet name="snmp">
   <param name="DeviceAddress">080F</param>
   <param name="DeviceIndex">8</param>
  </paramSet>
 </entity>

 <!-- Entidad EIB 3 -->
 <entity name="eib3" type="Plug">
  <property name="Status">
   <paramSet name="default">
    <param name="access">2</param>
    <param name="location"  >1</param>
    <param name="initialValue"   >0</param>
   </paramSet>
   <paramSet name="jeoffrey" >
     <param name="type">switch</param>
     <param name="text_off">Encender&#x20;cafetera</param>
     <param name="cmd_off">1</param>
     <param name="text_on">Apagar&#x20;cafetera</param>
     <param name="cmd_on">0</param>
    <param name="color_on">0x00FF00</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">EIB3</param>
  </paramSet>
  <paramSet name="snmp">
   <param name="DeviceAddress">0805</param>
   <param name="DeviceIndex">14</param>
  </paramSet>
  <paramSet name="jeoffrey" >
   <param name="image">cafetera.gif</param>
```

```
   <param name="x">650</param>
   <param name="y">78</param>
  </paramSet>
 </entity>

<!-- Entidad EIB 5 -->
 <entity name="eib5" type="Plug">
  <property name="Status">
   <paramSet name="default">
    <param name="access">2</param>
    <param name="location"  >1</param>
    <param name="initialValue"   >0</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">EIB5</param>
  </paramSet>
  <paramSet name="snmp">
   <param name="DeviceAddress">0807</param>
   <param name="DeviceIndex">16</param>
  </paramSet>
 </entity>

<!-- Entidad EIB 6 -->
 <entity name="eib6" type="Plug">
  <property name="Status">
   <paramSet name="default">
    <param name="access">2</param>
    <param name="location"  >1</param>
    <param name="initialValue"   >0</param>
   </paramSet>
   <paramSet name="jeoffrey" >
     <param name="type">switch</param>
     <param name="text_off">Encender&#x20;TV</param>
     <param name="cmd_off">1</param>
     <param name="text_on">Apagar&#x20;TV</param>
     <param name="cmd_on">0</param>
    <param name="color_on">0x00FF00</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">EIB6</param>
  </paramSet>
  <paramSet name="snmp">
   <param name="DeviceAddress">0808</param>
   <param name="DeviceIndex">17</param>
```

```
 </paramSet>
 <paramSet name="jeoffrey" >
  <param name="image">tele.gif</param>
  <param name="x">390</param>
  <param name="y">247</param>
 </paramSet>
</entity>

 <!-- Personas -->
 <!-- Entidad Xavier Alaman -->
<entity name="xavier" type="Person">
 <property name="Nombre" >
  <paramSet name="default">
   <param name="access">0</param>
   <param name="location"  >0</param>
   <param name="initialValue"   >Xavier</param>
  </paramSet>
 </property>
 <property name="Apellidos" >
  <paramSet name="default">
   <param name="access">0</param>
   <param name="location"  >0</param>
   <param name="initialValue"   >Alaman</param>
  </paramSet>
 </property>
 <property name="NumTarjeta" >
  <paramSet name="default">
   <param name="access">0</param>
   <param name="location"  >0</param>
   <param name="initialValue"   >0005FF0320</param>
  </paramSet>
 </property>
 <property name="Correoe" >
  <paramSet name="default">
   <param name="access">0</param>
   <param name="location"  >0</param>
   <param name="initialValue"   >xavier.alaman@uam.es</param>
  </paramSet>
 </property>
 <paramSet name="default">
  <param name="desc">Personas&#x20;autorizadas</param>
 </paramSet>
 <relation name="likes" destination="meninas" type="Picture"></relation>
 <relation name="likes" destination="guernica" type="Picture"></relation>
 <relation name="likes" destination="Ascending_descending" type="Picture"></relati
 <relation name="hasMeeting" destination="ruth" type="Person"></relation>
```

```
  <relation name="hasMeeting" destination="german" type="Person"></relation>
  <relation name="hasMeeting" destination="pablo" type="Person"></relation>
 </entity>

<!-- Entidad German Montoro -->
 <entity name="german" type="Person">
  <property name="Nombre" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >German</param>
   </paramSet>
  </property>
  <property name="Apellidos" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >Montoro</param>
   </paramSet>
  </property>
  <property name="NumTarjeta" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >0005FF0188</param>
   </paramSet>
  </property>
  <property name="Esta" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >0</param>
   </paramSet>
  </property>
  <property name="Correoe" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >German.montoro@uam.es</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">Personas&#x20;autorizadas</param>
  </paramSet>
  <relation name="likes" destination="Maja_Vestida" type="Picture"></relation>
  <relation name="likes" destination="Puente_Argenteuil" type="Picture"></relation>
```

```
  <relation name="likes" destination="Concepcion" type="Picture"></relation>
 </entity>

<!-- Entidad Pablo Haya -->
 <entity name="phaya" type="Person">
  <property name="Nombre" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >Pablo</param>
   </paramSet>
  </property>
  <property name="Apellidos" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >Haya</param>
   </paramSet>
  </property>
  <property name="NumTarjeta" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >0005FEFFF1</param>
   </paramSet>
  </property>
  <property name="Correoe" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >Pablo.haya@uam.es</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">Personas&#x20;autorizadas</param>
  </paramSet>
  <relation name="likes" destination="phaya01" type="Picture"></relation>
  <relation name="likes" destination="phaya02" type="Picture"></relation>
  <relation name="likes" destination="phaya01" type="Picture"></relation>
 </entity>

<!-- Entidad Francisco Gomez -->
 <entity name="paco" type="Person">
  <property name="Nombre">
   <paramSet name="default">
    <param name="access">0</param>
```

```
   <param name="location"  >0</param>
   <param name="initialValue"   >Francisco</param>
  </paramSet>
 </property>
 <property name="Apellidos">
  <paramSet name="default">
   <param name="access">0</param>
   <param name="location"  >0</param>
   <param name="initialValue"   >Gomez</param>
  </paramSet>
 </property>
 <property name="NumTarjeta">
  <paramSet name="default">
   <param name="access">0</param>
   <param name="location"  >0</param>
   <param name="initialValue"   >0005FF1DA8</param>
  </paramSet>
 </property>
 <property name="Correoe" >
  <paramSet name="default">
   <param name="access">0</param>
   <param name="location"  >0</param>
   <param name="initialValue"   >Francisco.gomez@uam.es</param>
  </paramSet>
 </property>
 <paramSet name="default">
  <param name="desc">Personas&#x20;autorizadas</param>
 </paramSet>
 <relation name="likes" destination="nacimiento_Venus" type="Picture"></relation>
 <relation name="likes" destination="afueras_París" type="Picture"></relation>
 <relation name="likes" destination="jardín" type="Picture"></relation>
</entity>

<!-- Cuadros -->
<!-- Entidad El nacimiento de Venus -->
<entity name="nacimiento_Venus" type="Person">
 <property name="Nombre_Cuadro" >
  <paramSet name="default">
   <param name="access">0</param>
   <param name="location"  >0</param>
   <param name="initialValue"   >El&#x20;nacimiento&#x20;de&#x20;Venus</param>
  </paramSet>
  </property>
  <property name="Autor" >
   <paramSet name="default">
   <param name="access">0</param>
```

```
    <param name="location"  >0</param>
    <param name="initialValue"   >Botticelli</param>
   </paramSet>
  </property>
  <property name="Año_Pintado" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >1484-1485</param>
   </paramSet>
  </property>
  <property name="Nacionalidad" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >Italiana</param>
   </paramSet>
  </property>
  <property name="Fuente" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >boticelli_1.jpg</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">Cuadros</param>
  </paramSet>
 </entity>

<!-- Entidad La Persistencia de la Memoria -->
 <entity name="Persistencia_Memoria" type="Person">
  <property name="Nombre_Cuadro" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >La&#x20;Persistencia&#x20;de&#x20;la&#x20;Memoria
   </paramSet>
   </property>
   <property name="Autor" >
    <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >Dali</param>
   </paramSet>
  </property>
```

```
  <property name="Año_Pintado" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >1931</param>
   </paramSet>
  </property>
  <property name="Nacionalidad" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >Española</param>
   </paramSet>
  </property>
  <property name="Fuente" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >dali_1.jpg</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">Cuadros</param>
   <param name="type">Persistencia_Memoria</param>
  </paramSet>
 </entity>

<!-- Entidad Mona Lisa -->
 <entity name="Mona_Lisa" type="Person">
  <property name="Nombre_Cuadro" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >Mona&#x20;Lisa</param>
   </paramSet>
   </property>
   <property name="Autor" >
    <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >Leonardo&#x20;da&#x20;Vinci</param>
   </paramSet>
  </property>
  <property name="Año_Pintado" >
   <paramSet name="default">
    <param name="access">0</param>
```

```
    <param name="location"  >0</param>
    <param name="initialValue"   >1502</param>
   </paramSet>
  </property>
  <property name="Nacionalidad" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >Italiana</param>
   </paramSet>
  </property>
  <property name="Fuente" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >davinci_1.jpg</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">Cuadros</param>
   <param name="type">Mona_Lisa</param>
  </paramSet>
 </entity>

<!-- Entidad El 2 de Mayo -->
 <entity name="El_2_Mayo" type="Person">
  <property name="Nombre_Cuadro" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >El&#x20;2&#x20;de&#x20;Mayo</param>
   </paramSet>
   </property>
   <property name="Autor" >
    <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >Francisco&#x20;de&#x20;Goya</param>
   </paramSet>
  </property>
  <property name="Año_Pintado" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >1814</param>
   </paramSet>
```

```
  </property>
  <property name="Nacionalidad" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >Española</param>
   </paramSet>
  </property>
  <property name="Fuente" >
   <paramSet name="default">
    <param name="access">0</param>
    <param name="location"  >0</param>
    <param name="initialValue"   >goya_1.jpg</param>
   </paramSet>
  </property>
  <paramSet name="default">
   <param name="desc">Cuadros</param>
   <param name="type">El_2_Mayo</param>
  </paramSet>
 </entity>
</instances>
```

Bibliografía

[1] AARTS, E., HARWIG, R., AND SCHUURMANS, M. *The invisible future: the seamless integration of technology into everyday life.* McGraw-Hill, Inc., New York, NY, USA, 2002, ch. Ambient intelligence, pp. 235–250.

[2] ABASCAL, J., CAGIGAS, D., GARAY, N., AND GARDEAZABAL, L. Mobile interface for a smart wheelchair. In *Fourth International Symposium on Human Computer Interaction with Mobile Devices* (Pisa (Italy), 2002), F. Paternò, Ed., vol. 411 of *LNCS*, Springer-Verlag, pp. 373–377.

[3] ABASCAL, J., SEVILLANO, J., CIVIT, A., JIMÉNEZ, G., AND FALCÓ, J. Integration of heterogeneous networks to support ambient intelligence in assistive environments. In *IFIP WG 9.3 Home Oriented Informatics and Telematics. HOIT 2005* (13-15 April 2005), A. Sloane (ed.): Home-Oriented Informatics ad Telematics. Springer-IFIP, pp. 323–335.

[4] ABOWD, G., AND MANKOFF, J. Domisilica: Providing ubiquitous access to the home. Technical Report GIT-GVU-97-17, GVU Center, Georgia Institute of Technology, May 1997.

[5] ABOWD, G. D. Classroom 2000: an experiment with the instrumentation of a living educational environment. *IBM System Journal 38*, 4 (1999), 508–530.

[6] ABOWD, G. D., ATKESON, C. G., HONG, J. I., LONG, S., KOOPER, R., AND PINKERTON, M. Cyberguide: A mobile context-aware tour guide. *Wireless Networks 3*, 5 (October 1997), 421–433.

[7] ABU-HAKIMA, S. The use of context in diagnostic systems. In *IJCAI-93 Workshop on Using Knowledge In Its Context* (Chambéry, France, August 1993).

[8] ACCENTURE. Homelab. http://www.accenture.com/.

[9] ACCORD. http://www.sics.se/accord, 2001.

[10] AGORAS, A. http://www.ambient-agoras.org, 2001.

[11] AIRE. http://aire.csail.mit.edu/, 2003.

[12] ÅKESSON, K.-P., BULLOCK, A., GREENHALGH, C., KOLEVA, B., AND RODDEN, T. A toolkit for user re-configuration of ubiquitous domestic environments. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology* (Paris, 2000), ACM Press.

[13] ALAMÁN, X., ANGUIANO, E., CORBACHO, F., GÓMEZ-ARRIBA, F., HAYA, P. A., MARTÍNEZ, J., AND MONTORO, G. ODISEA: Active environments for intelligent offices and home. In *First Technical Workshop of the Computer Engineering Department* (UAM, Spain, March 31st 2000).

[14] ALAMÁN, X., CABELLO, R., GÓMEZ-ARRIBA, F., HAYA, P. A., MARTÍNEZ, A., MARTÍNEZ, J., AND MONTORO, G. Using context information to generate dynamic user interfaces. In *10th International Conference on Human-Computer Interaction, HCI International 2003* (Crete, Greece, June 22-27 2003).

[15] ALAMÁN, X., HAYA, P. A., AND MONTORO, G. El proyecto Interact: Una arquitectura de pizarra para la implementación de entornos activos. In *International Conference on Human-Computer Interaction (Interaction 2001)* (Salamanca, Spain, May 16-18 2001).

[16] ALI, M. F., PÉREZ-QUIÑONES, M. A., ABRAMS, M., AND SHELL, E. *Building Multi-Platform User Interfaces with UIML*, computer-aided design of user interfaces iii (cadui) ed. Kluwer Academic Publishers, Dordrecht, Hardbound, 2002, ch. 22, pp. 225–236.

[17] ALLIANCE, O. M. User agent profile (UAProf). Technical Report WAP-248-UAProf-20011020-a, Open Mobile Alliance, 2001.

[18] ALLIANCE, Z. http://www.zigbee.org/, 2004.

[19] ALMENAREZ, F., MARÍN, A., CAMPO, C., AND GARCÍA-RUBIO, C. Security model for intelligent-agents-based pervasive computing environments. *IEEE Pervasive Computing - Security and Privacy 2*, 1 (Jan-Mar 2003).

[20] ALONSO, N., BALAGUERA, A., JUNYENT, E., LAFUENTE, A., LÓPEZ, J. B., LORÉS, J., MUÑOZ, D., PÉREZ, M., AND TARTERA, E. Virtual reality as an extension of the archaeological record: The reconstruction of the iron age fortress el vilars. In *CAA, Computer Applications and quantitative methods in Archaeology* (Ljubliana, Eslovenia, 2000).

[21] ARAUJO, A., JIMÉNEZ, S., NIETO-TALADRIZ, O., AND DEL POZO, F. Red personal inalámbrica de sensores para telemonitorización domiciliaria. In *Simposio Computación Ubicua Inteligencia Ambiental, UCAmI'05* (Granada, Spain, 2005), Thomson, pp. 128–135.

[22] ARNSTEIN, L., HUNG, C.-Y., FRANZA, R., ZHOU, Q. H., BORRIELLO, G., CONSOLVO, S., AND SU, J. Labscape: A smart environment for the cell biology laboratory. *IEEE Pervasive Computing 1*, 3 (2002), 13–21.

[23] ASSOCIATION, E. H. S. Ehs 1.3a - european home systems specification, 2002.

[24] ASSOCIATION, E. I. *CEBus*, 1995.

[25] BACON, J., BATES, J., AND HALLS, D. Location-oriented multimedia. *IEEE Personal Communications 4*, 5 (october 1997), 48–57.

[26] BALLESTEROS, F. J., SORIANO, E., LEAL, K., AND GUARDIOLA, G. Plan b: An operating system for ubiquitous computing environments. In *Proceedings of Fourth Annual IEEE International Conference on Pervasive Computing and Communications PerCom'06* (Próxima publicación, 2006).

[27] BANAVAR, G., BECK, J., GLUZBERG, E., MUNSON, J., SUSSMAN, J., AND ZUKOWSK, D. Challenges: An Application Model for Pervasive Computing. In *Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)* (2000), pp. 266–274.

[28] BAO, L., AND INTILLE, S. S. Activity recognition from user-annotated acceleration data. In *Pervasive Computing, Second International Conference, PERVASIVE 2004* (Vienna, Austria, April 2004), vol. 3001 of *Lecture Notes in Computer Science*, Springer, pp. 1–17.

[29] BARR, A., AND FEIGENBAUM, E., Eds. *The Handbook of Artificial Intelligence*, vol. 1 of *Chap III: Representation of Knowledge, representation.* William Kaufmann, Inc., 1981.

[30] BARTLEBY. http://www.bartleby.com/61/, 2000.

[31] BATIBUS. http://www.batibus.com.

[32] BEIGL, M., GELLERSEN, H. W., AND SCHMIDT, A. Mediacups: Experience with design and use of computer-augmented everyday objects. *Computer Networks 35*, 4 (March 2001), 401–409.

[33] BEIGL, M., SCHMIDT, A., LAUFF, M., AND GELLERSEN, H.-W. The ubicompbrowser. In *Proceedings of the 4th ERCIM Workshop on User Interfaces for All* (Stockholm, Sweden, October 1998).

[34] BELOTTI, R., DECURTINS, C., GROSSNIKLAUS, M., NORRIE, M. C., AND PALINGINIS, A. Modelling context for information environments. In *Ubiquitous Mobile Information and Collaboration Systems, Second CAiSE Workshop, UMICS 2004* (Riga, Latvia, June 7-8 2004), vol. 3272 of *Lecture Notes in Computer Science*, Springer, pp. 43–56.

[35] BERNERS-LEE, T., MASINTER, L., AND MCCAHILL, M. Uniform resource locators (URL). Request For Comments 1738, Internet Engineering Task Force, December 1994.

[36] BLANCO, J. M. R., AND FERNÁNDEZ, V. M. G. Desarrollo de un proveedor de servicios http para dispositivos móviles mediante etiquetas rfid. In *I Simposio Computación Ubicua Inteligencia Ambiental, UCAmI'05* (Granada, Spain, 2005), Thomson, pp. 209–216.

[37] BLY, S. A., HARRISON, S. R., AND IRWIN, S. Media spaces: bringing people together in a video, audio, and computing environment. *Communications of the ACM 36*, 1 (1993), 28–46.

[38] BOBICK, A., INTILLE, S., DAVIS, J., BAIRD, F., PINHAEZ, C., CAMPBELL, L., IVANOV, Y., SCHÜTTE, A., AND WILSON, A. The kidsroom: A perceptually-based interactive and immersive story environment. *PRESENCE: Teleoperators and Virtual Environments 8*, 4 (august 1999), 367–391.

[39] BOY, G. Computer integrated documentation. NASA Technical Memorandum 103870, NASA Ames Research Center, Moffett Field, CA 94035-1000, USA, September 1991.

[40] BRAVO, J., HERVÁS, R., NAVA, S., AND CHAVIRA, G. Ubiquitous computing at classroom: An approach through identification process. *Special Issue on Computer and Educations. Journal of Universal Computer Science* (september 2005).

[41] BRAVO, J., HERVÁS, R., SÁNCHEZ, I., AND CRESPO, A. Servicios por identificación en el aula ubicua. In *Avances en Informática Educativa. Juan Manuel Sánchez et al. (Eds.)* (2005), Servicio de Publicaciones Universidad de Extremadura.

[42] BRAVO, J., ORTEGA, M., REDONDO, M. A., AND BRAVO., C. Interacción abstracta en una clase ubicua. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, 16 (2002), 49–54.

[43] BROOKS, R. The intelligent room project. In *Proceedings of the 2nd International Cognitive Technology Conference (CT'97)* (Aizu, Japan, 1997).

[44] BROWN, P. Triggering information by context. *Personal and Ubiquitous Computing 2*, 1 (1998), 1–9.

[45] BROWN, P. J. The Stick-e document: A framework for creating context-aware applications. In *Proceedings of the Sixth International Conference on Electronic Publishing, Document Manipulation and Typography, 24–26 September, 1996, Palo Alto, CA, USA* (New York, NY, USA; London, UK; Sydney, Australia, 1996), A. Brown, A. Brüggemann-Klein, and A. Feng, Eds., John Wiley and Sons, pp. 259–272.

[46] BROWN, P. J., BOVEY, J. D., AND CHEN, X. Context-aware applications: From the laboratory to the marketplace. *IEEE Personal Communications 4*, 5 (October 1997), 58–64.

[47] BRUMITT, B., MEYERS, B., KRUMM, J., KERN, A., AND SHAFER, S. A. Easyliving: Technologies for intelligent environments. In *Handheld and Ubiquitous Computing, 2nd Intl. Symposium* (september 2000), pp. 12–27.

[48] BRUMITT, B., AND SHAFER, S. Topological world modeling using semantic spaces. In *Workshop on Location Modeling for Ubiquitous Computing (UBICOMP)* (October 2001).

[49] BRÉZILLON, P. Context in problem solving: A survey. *The Knowledge Engineering Review 1*, 14 (1999).

[50] BUXTON, W., AND MORAN, T. Europarc's integrated interactive intermedia facility (iiif): Early experience. In *Multiuser interfaces and applications, Proceedings of the IFIP WG 8.4 Conference on Multi-user Interfaces and Applications* (Heraklion, Crete, 1990), S. G. . A. Verrijn-Stuart, Ed., Elsevier Science Publishers B.V. (North-Holland), pp. 11–34.

[51] CABRI, G., LEONARDI, L., AND ZAMBONELLI, F. Mobile-agent coordination models for internet applications. *IEEE Computer 33*, 2 (February 2000), 82–89.

[52] CAHOUR, B., AND KARSENTY, L. Context of dialogue: a cognitive point of view. In *IJCAI-93 Workshop on Using Knowledge In Its Context* (Chambéry, France, August 1993).

[53] CALDERÓN, L., CERES, R., PONS, J., SECO, F., AND JIMÉNEZ, A. Sensores de localización ultrasónicos y su aplicación a la movilidad de personas con discapacidad. In *Contribuciones Tecnológicas para la Discapacidad. Congreso Riberdiscap* (Natal, Brasil, May 2003), pp. 95–100.

[54] CAMBRIDGE, A. L. omniorb c++ corba orb. http://omniorb.sourceforge.net/.

[55] CAMPBELL, L. W., AND BOBICK, A. F. Recognition of human body motion using phase space constraints. In *Fifth International Conference on Computer Vision ICCV* (Cambridge, MA, June 1995), pp. 624–630.

[56] CAMPO, C. Agentes móviles en computación ubicua. In *Actas del Tercer Congreso Interacción Persona-Ordenador* (Madrid, Spain, 2002).

[57] CAN YOU SEE ME NOW? http://www.canyouseemenow.co.uk/, 2002.

[58] CASAS, R., CUARTIELLES, D., FALCO, J., AND MALMBORG, L. Positioning technologies in learning. In *Proceedings IEEE International Workshop on Wireless and Mobile Technologies in Education* (Sweden, 2002), IEEE Computer Society, pp. 161–162.

[59] CASTRO, A., CHAQUET, J., MOREJÓN, E., RIESGO, T., AND UCEDA, J. A system-on chip for smart sensors. In *IEEE InternationalSymposium on Industrial Electronics (ISIE)* (L'Aquila, Italy, July 2002).

[60] CASTRO, P., AND MUNTZ, R. Managing context data for smart spaces. *IEEE Personal Communication Interactive 7*, 5 (October 2000), 21–28.

[61] CHEN, G., AND KOTZ, D. A survey of context-aware mobile computing research. Technical Report TR2000-381, Darmouth College, 2000.

[62] CHEN, H., FININ, T., AND JOSHI, A. Semantic web in in the context broker architecture. In *Proceedings of PerCom 2004* (March 2004).

[63] CHEN, H., PERICH, F., FININ, T., AND JOSHI, A. SOUPA: Standard ontology for ubiquitous and pervasive applications. In *Proceedings of First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04)* (2004), pp. 258–267.

[64] CHEVERST, K., DAVIES, N., MITCHELL, K., FRIDAY, A., AND EFSTRATIOU, C. Developing a context-aware electronic tourist guide: some issues and experiences. In *CHI* (2000), pp. 17–24.

[65] CHEVERST, K., MITCHELL, K., AND DAVIES, N. Design of an object model for a context sensitive tourist guide. In *In Proceedings of Workshop on Interactive Applications of Mobile Computing (IMC98)* (Rostock, Germany, November 1998).

[66] CHOU, L.-D., LEE, C.-C., LEE, M.-Y., AND CHANG, C.-Y. A tour guide system for mobile learning in museums. In *Proceedings 2nd IEEE International Workshop on Wireless and Mobile Technologies in Education (WMTE 2004), Mobile Support for Learning Communities* (Taoyuan, Taiwan, March 2004), pp. 195–196.

[67] CHRISTENSSON, B., AND LARSSON, O. Universal plug and play connects smart devices. In *Windows Hardware Engineering Conference* (1999).

[68] CISCO. Internet home. http://www.cisco.com/warp/public/3/uk/ihome/.

[69] CLARK, H. H. *Using language.* Cambridge University Press, 1996.

[70] COEN, M., PHILLIPS, B., WARSHAWSKY, N., WEISMAN, L., PETERS, S., AND FININ, P. Meeting the computational needs of intelligent environments: The metaglue system. In *1st International Workshop on Managing Interactions in Smart Environments (MANSE'99)* (Dublin, Ireland, December 1999), P.Ñixon, G. Lacey, and S. Dobson, Eds., Springer-Verlag, pp. 201–212.

[71] COEN, M. H. Sodabot: A software agent enviroment and construction system. Tech. Rep. 1493, AI Lab Technical, Massachusetts Institute of Technology, Cambrigde, MA, USA, 1994.

[72] COEN, M. H. Building brains for rooms: Designing distributed software agents. In *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence* (Providence, Rhode Island, USA, 1997).

[73] COEN, M. H. Design principles for intelligent environments. In *Fifteenth National Conference on Artificial Intelligence (AAAI'98)* (Madison, WI, 1998), AAAI, AAAI Press, pp. 547–554.

[74] COHN, A., BENNETT, B., GOODAY, J., AND GOTTS, N. RCC: a calculus for region based qualitative spatial reasoning. *GeoInformatica 1* (1997), 275–316.

[75] COOK, D. J., YOUNGBLOOD, M., HEIERMAN, E., GOPALRATNAM, K., RAO, S., LITVIN, A., AND KHAWAJA, F. Mavhome: An agent-based smart home. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications* (2003), pp. 521–524.

[76] COOPERSTOCK, J. R., TANIKOSHI, K., BEIRNE, G., NARINE, T., AND BUXTON, W. Evolution of a reactive environment. In *Proceedings of the ACM SIGCHI95 Conference* (1995).

[77] COUTAZ, J., AND REY, G. Foundations for a theory of contextors. In *Computer-Aided Design of User Interfaces III* (May 2002), C. Kolski and J. Vanderdonckt, Eds., Kluwer Academic Publishers, pp. 13–34.

[78] DARRELL, T., MOGHADDAM, B., AND PENTLAND, A. P. Active face tracking and pose estimation in an interactive room. In *CVPR '96: Proceedings of the 1996 Conference on Computer Vision and Pattern Recognition (CVPR '96)* (Washington, DC, USA, 1996), IEEE Computer Society, pp. 67–72.

[79] DAVIES, N., CHEVERST, K., MITCHELL, K., AND FRIDAY, A. Caches in the air: Disseminating tourist information in the guide system. In *Proceedings of Second IEEE Workshop on Mobile Computing Systems and Applications* (February 1999), IEEE Computer Press.

[80] DE IPIÑA, D. L., VÁZQUEZ, J. I., GARCIA, D., FERNÁNDEZ, J., AND GARCÍA, I. A middleware for the deployment of ami spaces. In *Ambient Intelligence and (Everyday) Life* (Donostia, Spain, 2005).

[81] DEARLE, A., KIRBY, G. N. C., MORRISON, R., MCCARTHY, A., MULLEN, K., YANG, Y., CONNOR, R. C. H., WELEN, P., AND WILSON, A. Architectural support for global smart spaces. In *Mobile Data Management, 4th International Conference, MDM* (Melbourne, Australia, January 2003), vol. 2574 of *Lecture Notes in Computer Science*, Springer, pp. 153–164.

[82] DEVAUL, R., SUNG, M., GIPS, J., AND PENTLAND, A. S. Mithril 2003: Applications and architecture. http://web.media.mit.edu/ rich/209_DeVaul_R.pdf.

[83] DEY, A. K. Understanding and using context. *Personal and Ubiquitous Computing, Special Issue on Situated Interaction and Ubiquitous Computing 5*, 1 (2001), 4–7.

[84] DEY, A. K., AND ABOWD, G. D. Towards a better understanding of context and context-awareness. Technical Report GIT-GVU-99-22, Georgia Institute of Technology, June 1999.

[85] DEY, A. K., AND ABOWD, G. D. Cybreminder: A context-aware system for supporting reminders. In *Handheld and Ubiquitous Computing, Second International Symposium, HUC 2000* (Bristol, UK, september 2000), P. J. Thomas and H.-W. Gellersen, Eds., vol. 1927 of *LNCS*, Springer, pp. 172–186.

[86] DEY, A. K., SALBER, D., AND ABOWD, G. D. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction (HCI) Journal 16*, 2-4 (2001), 97–166.

[87] DEY, A. K., SALBER, D., ABOWD, G. D., AND FUTAKAWA, M. The conference assistant: Combining context-awareness with wearable computing. In *Proceedings of Third International Symposium on Wearable Computers (ISWC 1999)* (San Francisco, California, USA, October 1999), IEEE Computer Society, pp. 21–28.

[88] DIX, A., RODDEN, T., DAVIES, N., TREVOR, J., FRIDAY, A., AND PALFREYMAN, K. Exploiting space and location as a design framework for interactive mobile systems. *ACM Trans. Comput.-Hum. Interact. 7*, 3 (2000), 285–321.

[89] ECHELON. *Introduction to control networks System*, 1.0 ed., 1999.

[90] EDWARDS, W. K. *Core Jini.* Prentice-Hall, Inc., 1999.

[91] ELROD, S., BRUCE, R., GOLD, R., GOLDBERG, D., HALASZ, F. G., JR., W. C. J., LEE, D., MCCALL, K., PEDERSEN, E. R., PIER, K. A., TANG, J. C., AND WELCH, B. Liveboard: A large interactive display supporting group meetings, presentations, and remote collaboration. In *Proceedings of the Conference on Human factors in computing systems (CHI'92)* (Monterey, CA, USA, May 1992), ACM, pp. 599–607.

[92] ELROD, S., HALL, G., CONSTANZA, R., DIXON, M., AND RIVIÈRES, J. D. Responsive office environments. *Communications of the ACM 36*, 7 (1993), 84–85.

[93] ENGELMORE, R., AND MORGAN, T. *Blackboard Systems.* Addison-Wesley, 1988.

[94] EQUATOR. http://www.equator.ac.uk, 2000.

[95] ERMAN, L. D., HAYES-ROTH, F., LESSER, V. R., AND REDDY, R. The hearsay-ii speech-understanding system: Integrating knowledge to resolve uncertainty. *ACM Computing Surveys 12*, 2 (1980), 213–253.

[96] ESLER, M., HIGHTOWER, J., ANDERSON, T., AND BORRIELLO, G. Next century challenges: Data-centric networking for invisible computing: The portolano project at the university of washington. In *Proceedings of 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking* (Seattle, Washington, USA., august 1999), pp. 256–262.

[97] ESSA, I., ABOWD, G., AND ATLESON, C. Ubiquitous smart spaces. White paper submitted to darpa, Georgia Institute of Technology, 1998.

[98] ESSA, I., BASU, S., DARRELL, T., AND PENTLAND, A. Modeling, tracking and interactive animation of faces and heads using input from video. In *Proceedings of Computer Animation '96 Conference* (Geneva, Switzerland, June 1996), IEEE Computer Society Press.

[99] ESSA, I. A. Ubiquitous sensing for smart and aware environments. *IEEE Personal Communications 7*, 5 (2000), 47–49.

[100] FARRINGDON, J., MOORE, A., TILBURY, N., CHRUCH, J., AND BIEMOND, P. Wearable sensor badge & sensor jacket for context awareness. In *Proc. of the International Symposium on Wearable Computing (ISWC'99)* (San Francisco, CA, USA, November 1999), pp. 107–113.

[101] FERSCHA, A. Contextware: Bridging physical and virtual worlds. In *Proceeding of Reliable Software Technologies - 7th Ada-Europe International Conference on Reliable Software Technologies* (2002), J. Blieberger and A. Strohmeier, Eds., vol. 2361 of *Lecture Notes in Computer Science*, Springer, pp. 51–64.

[102] FOLDOC. http://wombat.doc.ic.ac.uk/foldoc/, 1993.

[103] FOX, A., JOHANSON, B., HANRAHAN, P., AND WINOGRAD, T. Integrating information appliances into an interactive workspace. *IEEE Computer Graphics and Applications 20*, 3 (2000), 54–65.

[104] FREED, N., AND BORENSTEIN, N. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. Request For Comments 2046, Internet Engineering Task Force, 1996.

[105] GAJOS, K. Rascal - a resource manager for multi agent systems in smart spaces. In *Proceedings of The Second International Workshop of Central and Eastern Europe on Multi-Agent Systems CEEMAS* (Kraków, Poland, 2001).

[106] GAL, C. L., MARTIN, J., LUX, A., AND CROWLEY, J. L. Smartoffice: Design of an intelligent environment. *IEEE Intelligent Systems 16*, 4 (July-August 2001), 60–66.

[107] GALIBERT, O. P., GAROFOLO, J. S., LAPRUN, C., MICHEL, M., AND STANFORD, V. The nist smart space and meeting room projects: Signals,

acquisition, annotation, and metrics. In *IEEE Conference on Acoustics, Speech, and Signal Processing* (January 2003).

[108] GARRIDO, J. L. *AMENITIES: Una Metodología para el desarrollo de sistemas cooperativos basada en modelos de comportamiento y tareas.* PhD thesis, Universidad de Granada, 2003.

[109] GEISSLER, J. Shuffle, throw or take it! working efficiently with an interactive wall. In *Conference on Human Factors in Computing Systems* (Los Angeles, California, United States, 1998), CHI'98, pp. 265–266.

[110] GELERNTER, D. Generative communication in linda. *ACM Transactions on Programming Languages and Systems 7*, 1 (January 1985), 80–112.

[111] GELLERSEN, H. W., SCHMIDT, A., AND BEIGL, M. Multi-sensor context-awareness in mobile devices and smart artifacts. *Mobile Networks and Applications 7*, 5 (2002), 341–351.

[112] GONG, L. A software architecture for open service gateways. *IEEE Internet Computing 5*, 1 (2001), 64–70.

[113] GONZÁLEZ, J. L., RUBIO, A., AND MOLL, F. Human powered piezoelectric batteries to supply power to wearable electronic devices. *International Journal of the Society of Materials Engineering for Resources 10*, 1 (2002), 33–40.

[114] GOOSSENS, M. The eib system for home and building. Tech. rep., EIBA s.c., 1998.

[115] GRAY, P., AND SALBER, D. Modelling and using sensed context information in the design of interactive applications. In *EHCI* (2001), vol. 2254 of *Lecture Notes in Computer Science*, Springer, pp. 157–172.

[116] GRIBBLE, S. D., WELSH, M., VON BEHREN, J. R., BREWER, E. A., CULLER, D. E., BORISOV, N., CZERWINSKI, S. E., GUMMADI, R., HILL, J. R., JOSEPH, A. D., KATZ, R. H., MAO, Z. M., ROSS, S., AND ZHAO, B. Y. The Ninja architecture for robust Internet-scale systems and services. *Computer Networks 35*, 4 (2001), 473–497.

[117] GRIMM, R., DAVIS, J., LEMAR, E., MACBETH, A., SWANSON, S., ANDERSON, T., BERSHAD, B., BORRIELLO, G., GRIBBLE, S., AND WETHERALL, D. System support for pervasive applications. *ACM Trans. Comput. Syst. 22*, 4 (2004), 421–486.

[118] GRIMM, R., DAVIS, J., LEMAR, E., MACBETH, A., SWANSON, S., GRIBBLE, S., ANDERSON, T., BERSHAD, B., BORRIELLO, G., AND WETHERALL, D. Programming for pervasive computing environments. Technical report UW-CSE-01-06-01, Department of Computer Science and Engineering, University of Washington, June 2001.

[119] GRISWOLD, W. G., BOYER, R., BROWN, S. W., AND TRUONG, T. M. A component architecture for an extensible, highly integrated context-aware computing infrastructure. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering* (2003), IEEE Computer Society, pp. 363–372.

[120] GRISWOLD, W. G., BOYER, R., BROWN, S. W., TRUONG, T. M., BHASKER, E., JAY, G. R., AND SHAPIRO, R. B. Activecampus - sustaining educational communities through mobile technology. Technical Report CS2002-0714, Computer Science and Engineering, UC San Diego, July 2002.

[121] GROSS, T. Ambient interfaces: Desing challenges and recommendations. In *Proceedings of the 10th International Conference on Human-Computer Interaction - HCII 2003* (Crete, Greece, June 2003), D. Harris, V. Duffy, M. Smith, and C. Stephanidis, Eds., Lawrence Erlbaum, Hillsdale, NJ, pp. 68–72.

[122] GROSSMAN, M., LEONHARDI, A., MITSCHANG, B., AND ROTHERMEL, K. A world model for location-aware systems. *Informatique-Revue des Organisations Suisses d'Informatique*, 5, Special issue on Ubiquitous Computing (October 2001).

[123] GROUP, O. M. The common object request broker architecture: Architecture and specification, October 1999.

[124] GU, T., PUNG, H. K., AND ZHANG, D. Q. Towards an osgi-based infrastructure for context-aware applications. *IEEE Pervasive Computing 3*, 4 (Octuber-December 2004), 66–74.

[125] GUHA, R. V. *Contexts: a formalization and some applications.* PhD thesis, Stanford University, 1991.

[126] GÁRATE, A., HERRASTI, N., AND LÓPEZ, A. An ambient intelligence application in a real home automation environment. In *I Simposio Computación Ubicua Inteligencia Ambiental, UCAmI'05* (Granada, Spain, 2005), Thomson, pp. 329–335.

[127] HAĆ, A. *Wireless Sensor Network Designs.* John Wiley & Sons, 2003.

[128] HAGRAS, H., CALLAGHAN, V., COLLEY, M., CLARKE, G., POUNDS-CORNISH, A., AND DUMAN, H. Creating an ambient-intelligence environment using embedded agents. *IEEE Intelligent Systems 19*, 6 (2004), 12–20.

[129] HALPIN, T. *Information modeling and relational databases: from conceptual analysis to logical design.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[130] HANSMANN, U., MERK, L., NICKLOUS, M. S., AND STOBER, T. *Pervasive Computing Handbook.* Springer-Verlag, 2001.

[131] HARRY CHEN, TIM FININ, A. J. An ontology for context-aware pervasive computing environments. *The Knowledge Engineering Review 18*, 3 (September 2003), 197–207.

[132] HARTER, A., HOPPER, A., STEGGLES, P., WARD, A., AND WEBSTER, P. The anatomy of a context-aware application. In *5th annual ACM/IEEE international conference on Mobile Computing and Networking* (1999), pp. 59–68.

[133] HAYA, P. A., ALAMÁN, X., AND MONTORO, G. A comparative study of communication infrastructures for the implementation of ubiquitous computing. *UPGRADE, The European Journal for the Informatics Professional 2*, 5 (2001).

[134] HAYA, P. A., MONTORO, G., ALAMÁN, X., CABELLO, R., AND MARTÍNEZ, J. Extending an xml environment definition language for spoken dialogue and web-based interfaces. In *In Developing User Interfaces with XML: Advances on User Interface Description Languages Workshop at AVI04* (Gallipoli, Italy, May 25 2004).

[135] HAYES-ROTH, B. A blackboard architecture for control. *Artificial Intelligence*, 26 (1985), 251–321.

[136] HEALEY, J., AND PICARD, R. W. Startlecam: A cybernetic wearable camera. In *Second International Symposium on Wearable Computers (ISWC 1998)* (Pittsburgh, Pennsylvania, USA, October 1998), pp. 42–49.

[137] HECKMANN, D. Ubiquitous user modeling for situated interaction. In *UM '01: Proceedings of the 8th International Conference on User Modeling 2001* (2001), Springer-Verlag, pp. 280–282.

[138] HELAL, S., MANN, W., EL-ZABADANI, H., KADDOURA, Y., AND JANSEN, E. The gator tech smart house: A programmable pervasive space. *IEEE Computer 38*, 3 (March 2005), 50–60.

[139] HELAL, S., WINKLER, B., LEE, C., KADDOURA, Y., RAN, L., GIRALDO, C., KUCHIBHOTLA, S., AND MANN, W. Enabling location-aware pervasive computing applications for the elderly. In *Proceedings of First IEEE International Conference on Pervasive Computing and Communications (PerCom'03)* (Fort Worth,Texas, 2003), IEEE CS Press, pp. 531–538.

[140] HENRICKSEN, K., INDULSKA, J., AND MCFADDEN, T. Modelling context information with orm. In *In International Workshop on Object-Role Modeling (ORM)* (2005), vol. 3762 of *LNCS*, Springer-Verlag, pp. 626–635.

[141] HOLLAR, S. E.-A. *COST Dust.* PhD thesis, University of California, Berkeley, 1996.

[142] THE HOME PHONELINE NETWORKING ALLIANCE. *Simple, High-Speed Ethernet Technology for the Home*, 1998.

[143] HomeRF Working Group. *Introduction to Home RF Technical Specification*, 2000.

[144] Hong, J. I., and Landay, J. A. An infrastructure approach to context-aware computing. *Human-Computer Interaction 16*, 2-4 (2001), 287–303.

[145] Hong Z. Tan, I. L., and Pentland, A. The chair as a novel haptic user interface. In *In Proceedings of the Workshop on Perceptual User Interfaces* (Banff, Alberta, Canada, October 1997), pp. 56–57.

[146] HP. http://www.cooltown.com/, 2002.

[147] Humble, J., Crabtree, A., Hemmings, T., Åkesson, K.-P., Koleva, B., Rodden, T., and Hansson, P. 'playing with the bits' user-configuration of ubiquitous domestic environments. In *UbiComp 2003: Ubiquitous Computing, 5th International Conference* (2003), vol. 2864 of *Lecture Notes in Computer Science*, Springer, pp. 256–263.

[148] Information, and of European Commissions Research DG, C. U. The priorities of the sixth framework programme 2002 - 2006, November 2002.

[149] Institute of Electrical and Electronics Engineers, Inc. *IEEE Standard for a High Performance Serial Bus*, 1995.

[150] Intille, S. S. Designing a home of the future. *IEEE Perpasive Computing* (April-June 2002), 80–86.

[151] Ishii, H., and Ullmer, B. Tangible bits: Towards seamless interfaces between people, bits and atoms. In *Proceedings of CHI '97,* (March 1997).

[152] Ishii, H., Wisneski, C., Brave, S., Dahley, A., Gorbet, M., Ullmer, B., and Yarin, P. ambientroom: integrating ambient media with architectural space. In *CHI 1998 Proceedings of the conference on Human Factors in Computing Systems* (Los Angeles, California, United States, 1998), ACM, pp. 173–174.

[153] JavaSpace. http://java.sun.com/products/jini/2.0/doc/specs/html/js-spec.html, 2001.

[154] Johanson, B., and Fox, A. The event heap: A coordination infrastructure for interactive workspaces. In *4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002), 20-21 June 2002, Callicoon, NY, USA* (2002), IEEE Computer Society, pp. 83–93.

[155] Johanson, B., Fox, A., and Winograd, T. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervasive Computing Magazine 1*, 2 (April-June 2002), 67–74.

[156] KAGAL, L., FININ, T., AND JOSHI, A. A policy language for a pervasive computing environment. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks* (June 2004), IEEE Computer Society, pp. 63–76.

[157] KANTARJIEV, C. K., DEMERS, A., FREDERICK, R., KRIVACICA, R. T., AND WEISER, M. Experiences with x in a wireless environment. In *Proceedings of USENIX Mobile & Location-Independent Computing Symposium* (Cambridge, Massachusetts, August 1993).

[158] KICIMAN, E., AND FOX, A. Using dynamic mediation to integrate COTS entities in a ubiquitous computing environment. In *Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing (HUC2K)* (Heidelberg, Germany, 2000), pp. 211–226.

[159] KIDD, C. K., ORR, R., ABOWD, G. D., ATKENSON, C. G., ESSA, I. A., MACINTYRE, B., MYNATT, E., STARNER, T. E., AND NEWSTETTER, W. The aware home: A living laboratory for ubiquitous computing research. In *Proceedings of the Second International Workshop on Cooperative Buildings - CoBuild'99* (1999).

[160] KINDBERG, T., AND BARTON, J. A web-based nomadic computing system. *Computer Networks 35*, 4 (2001), 443–456.

[161] KINDBERG, T., BARTON, J., MORGAN, J., BECKER, G., CASWELL, D., DEBATY, P., GOPAL, G., FRID, M., KRISHNAN, V., MORRIS, H., SCHETTINO, J., AND SERRA, B. People, places, things: Web presence for the real world. Technical Report HPL-2000-16, Hewlett-Packard Laboratories, 2000.

[162] KORKEA-AHO, M. Context-aware applications survey. Tech. rep., Helsinki University of Technology, 2002.

[163] LAMMING, M., AND FLYNN, M. Forget-me-not: Intimate computing in support of human memory. In *Friend21: International Symposium on Next Generation Human Interface* (Meguro Gajoen, Japan, 2003), D. Harris, V. Duffy, M. Smith, and C. Stephanidis, Eds., pp. 125–128.

[164] LEIBERMAN, H., AND SELKER, T. Out of context: Systems that adapt to, and learn from , context. *IBM Systems Journal 39*, 3&4 (2000), 617–632.

[165] LEONHARDT, U. *Supporting location-awareness in open distributed systems.* PhD thesis, Imperial College of Science, University of London, 1998.

[166] LESTER, J., HANNAFORD, B., AND BORRIELLO, G. 'are you with me?' - using accelerometers to determine if two devices are carried by the same person. In *Pervasive Computing, Second International Conference, PERVASIVE 2004* (Vienna, Austria, April 2004), vol. 3001 of *Lecture Notes in Computer Science*, Springer, pp. 33–50.

[167] LOPEZ, A., DOCTOR, F., SANCHEZ, L., HAGRAS, H., AND CALLAGHAN, L. A comparison of some data-based methods for the off-line generation of fuzzy logic controllers for an intelligent building environment. In *the IEE International Workshop on Intelligent Environments* (Colchester, UK, June 28-29 2005).

[168] LOZANO, J. A., MONTESA, J., JUAN, M. C., ALCAÑIZ, M., REY, B., GIL, J., MARTINEZ, J. M., GAGGIOLI, A., AND MORGANTI, F. Vr-mirror: A virtual reality system for mental practice in post-stroke rehabilitation. In *Smart Graphics, 5th International Symposium, SG 2005, Frauenwörth Cloister, Germany, August 22-24, Proceedings* (2005), vol. 3638 of *Lecture Notes in Computer Science*, Springer, pp. 241–251.

[169] LUKOWICZ, P., WARD, J. A., JUNKER, H., STÄGER, M., TRÖSTER, G., ATRASH, A., AND STARNER, T. Recognizing workshop activity using body worn microphones and accelerometers. In *Pervasive Computing, Second International Conference, PERVASIVE 2004* (Vienna, Austria, April 2004), vol. 3001 of *Lecture Notes in Computer Science*, Springer, pp. 18–32.

[170] M-W. http://www.m-w.com, 2003.

[171] MAGLIO, P. P., AND CAMPBELL, C. S. Attentive agents. *Communications of the ACM 46*, 3 (2000), 47–51.

[172] MAO, Z. M. *Fault-tolerant, Scalable, Wide-Area Internet Service Composition.* PhD thesis, University of California at Berkeley, Berkeley, 2000.

[173] MARTIN, D. L., CHEYER, A. J., AND MORAN, D. B. A framework for building distributed software systems. *Applied Artificial Intelligence, 13*, 1&2 (January-March 1999), 91–128.

[174] MARTINEZ, A. E., CABELLO, R., GÓMEZ, F. J., AND MARTÍNEZ, J. Interact-ddm: A solution for the integration of domestic devices on network management platforms. In *IFIP/IEEE International Symposium on Integrated Network Management* (Colorado Springs, Colorado, USA, 2003).

[175] MARTÍN, J. M., JIMÉNEZ, A. R., SECO, F., CALDERÓN, L., PONS, J. L., AND CERES, R. Estimating the 3-d position from timedelay data of us-waves: experimental analysis and a new processing algorithm. *Sensors and Actuators A 101*, 3 (2002), 311–321.

[176] MARX, M., AND SCHMANDT, C. Clues: Dynamic personalized message filtering. In *CSCW '96, Proceedings of the ACM 1996 Conference on Computer Supported Cooperative Work* (Boston, MA, USA, November 1996), ACM, pp. 113–121.

[177] MASKERY, H., AND MEADS, J. Context: in the eyes of users and in computer systems. *SIGCHI Bulletin 24*, 2 (April 1992), 12–21.

[178] McCarthy, J. Notes on formalizing context. In *Proc. of the 13th IJCAI* (Chambéry, France, September 1993), vol. 1, pp. 555–560.

[179] Metcalfe, M. R., and Boggs, D. Ethernet: Distributed packet switching for local computer networks. *Communications of ACM 19*, 5 (1976), 395–404.

[180] Mikic, I., Huang, K., and Trivedi, M. M. Activity monitoring and summarization for an intelligent meeting room. In *Workshop on Human Motion (HUMO'00)* (Austin, Texas, December 2000), IEEE Computer Society, pp. 107–112.

[181] Minar, N., Gray, M., Roup, O., Krikorian, R., and Maes, P. Hive: Distributed agents for networking things. In *Proceedings of ASAMA'99 the First International Symposium on Agent Systems and Applications and Third International Symposium on Mobile Agents* (August 1999).

[182] Montoro, G. *Estudio e integración de un sistema de diálogos dinámico en un entorno inteligente.* PhD thesis, Universidad Autónoma de Madrid, 2005.

[183] Montoro, G., Alamán, X., and Haya, P. A. A plug and play spoken dialogue interface for smart environments. In *Fifth International Conference on Intelligent Text Processing and Computational Linguistics (CICLing'04)* (February 15-21 2004), vol. 2945 of *Lecture Notes in Computer Science (LNCS)*, Springer-Verlag.

[184] Moya, F., and López, J. Senda: An alternative to osgi for large scale domotics. *Networks, World Scientific Publishing* (2002), 165–176.

[185] Mozer, M. M. The neural network house: An environment that adapts to its inhabitants. In *Proceedings of the AAAI Spring Symposium on Intelligent Environments* (1998), AAAI Press.

[186] Mylopoulos, J., Bordiga, A., Jarke, M., and Koubarakis, M. Telos: representing knowledge about information system. *ACM Trans. on Information systems 8*, 4 (October 1990), 325–362.

[187] Mynatt, E. D., Back, M., Want, R., Baer, M., and Ellis, J. B. Designing audio aura. In *CHI '98, Proceeding of the CHI 98 Conference on Human Factors in Computing Systems* (Los Angeles, California, USA, april 1998), ACM, pp. 566–573.

[188] Müller-Tomfelde, C., and Reischl, W. Communication chairs: examples of mobile roomware components. In *Conference on Human Factors in Computing Systems* (Los Angeles, California, United States, 1998), CHI'98, pp. 267–268.

[189] NAGEL, K., KIDD, C. D., O'CONNELL, T., DEY, A. K., AND ABOWD, G. D. The family intercom: Developing a context-aware audio communication system. In *Proc. of Ubicomp 2001: Ubiquitous Computing, Third International Conference* (Atlanta, Georgia, USA, September-October 2001), vol. 2201 of *Lecture Notes in Computer Science*, Springer, pp. 176–183.

[190] NAGUIB, H., COULOURIS, G., AND MITCHELL, S. Middleware support for context-aware multimedia applications. In *Proceedings of the Third International Working Conference on Distributed Applications and Interoperable Systems DAIS* (2001), pp. 9–22.

[191] NARAYANAN, A. K. Realms and states: A framework for location aware mobile computing. In *Proceedings of Workshop on International Conference on Mobile Computing and Networking (MOBICOM)* (Rome, Italy, 2001).

[192] NATALIA MARMASSE, C. S. Location-aware information delivery with commotion. In *Handheld and Ubiquitous Computing, Second International Symposium, HUC 2000* (Bristol, UK, september 2000), P. J. Thomas and H.-W. Gellersen, Eds., vol. 1927 of *LNCS*, Springer, pp. 157–171.

[193] NELSON, G. J. *Context-aware and location systems.* PhD thesis, University of Cambridge, 1998.

[194] NIETO, I., BOTÍA, J. A., AND GÓMEZ-SKARMETA, A. F. Information and hybrid architecture model of the ocp contextual information management system. In *I Simposio Computación Ubicua Inteligencia Ambiental, (UCAmI'05)* (Granada, Spain, 2005), Thomson.

[195] NIXON, P., DOBSON, S., AND LACEY, G. Smart environments: some challenges for the computing community. In *Proceedings of 1st International Workshop on Managing Interactions in Smart Environments* (London, december 1999), P.Ñixon, G. Lacey, and S. Dobson, Eds., MANSE'99, Dublin, Dec 1999, Springer-Verlag, pp. 1–4.

[196] NORMAN, D. *The Invisible Computer.* MIT Press, 1998.

[197] OGDEN, C., AND RICHARDS, I. *The Meaning of Meaning*, 8th ed. Routledge&Kegan Paul LTD, Broadway House, 68-74 Carter Lane, London, E.C.4, 1946.

[198] OLIVER, N. *Towards Perceptual Intelligence: Statistical Modeling of Human Individual and Interactive Behaviors.* PhD thesis, MIT Media Lab, 2000.

[199] OPPERMANN, R., AND SPECHT, M. Adaptive support for a mobile museum guide. In *In Proc. Workshop on Interactive Applications of Mobile Computing (IMC'98)* (Rostock, Germany, 1998).

[200] ORR, R. J., AND ABOWD, G. D. The smart floor: A mechanism for natural user identification and tracking. In *Proceedings of the 2000 Conference*

on Human Factors in Computing Systems (CHI 2000) (The Hague, Netherlands, April 2000).

[201] ORTEGA, M., REDONDO, M., PAREDES, M., SÁNCHEZ-VILLALÓN, P. P., BRAVO, C., AND BRAVO, J. Nuevos paradigmas de interacción en el aula del siglo xxi. In *II Congreso Internacional de Interacción Persona-Ordenador* (Salamanca, Spain, 2001), pp. 161–171.

[202] OXYGEN. http://oxygen.lcs.mit.edu/, 2000.

[203] PASCOE, J. Adding generic contextual capabilities to wearable computers. In *2nd International Symposium on Wearable Computers* (1998), pp. 92–99.

[204] PASCOE, J., MORSE, D. R., AND RYAN, N. S. Developing personal technology for the field. *Personal Technologies 2* (August 1998), 28–36.

[205] PATERNÒ, F., AND SANTORO, C. One model, many interfaces. In *Computer-Aided Design of User Interfaces III.* (Dordrecht, Hardbound, May 2002), C. Kolski and J. Vanderdonckt, Eds., CADUI, Kluwer Academic Publishers, pp. 143–154.

[206] PATTERSON, D. J., ETZIONI, O., FOX, D., AND KAUTZ, H. Intelligent ubiquitous computing to support alzheimer's patients: Enabling the cognitively disabled. In *Intelligent Ubiquitous Computing to Support Alzheimer's Patients: Enabling the Cognitively Disabled* (Gothenberg , Sweden, 2002).

[207] PENTLAND, A. Smart rooms. *Scientific American 274*, 4 (April 1996), 68–78.

[208] PENTLAND, A. S., AND CHOUDHURY, T. Face recognition for smart environments. *Computer 33*, 2 (2000), 50–55.

[209] PETERS, S., AND SHROBE, H. Using semantic network for knowledge representation in an intelligent environment. In *PerCom '03: 1st Annual IEEE International Conference on Pervasive Computing and Communications* (March 2003), Ft. Worth, TX, USA.

[210] PHILIPOSE, M., FISHKIN, K. P., PERKOWITZ, M., PATTERSON, D. J., FOX, D., KAUTZ, H., AND HÄHNEL, D. Inferring activities from interactions with objects. *IEEE Pervasive Computing 3*, 4 (October-December 2004), 50–57.

[211] PHILIPS. http://www.philips.com/Assets/Downloadablefile/LEAFLET8x-1504.pdf, 2002.

[212] PHILLIPS, B. *Metaglue: A Programming Language for Multi-Agent Systems.* PhD thesis, MIT, 1999.

[213] PONNEKANTI, S. R., LEE, B., FOX, A., HANRAHAN, P., AND WINOGRAD, T. ICrafter: A service framework for ubiquitous computing environments. *Lecture Notes in Computer Science 2201* (2001), 56–77.

[214] PUERTA, A., AND EISENSTEIN, J. XIML: A universal language for user interfaces. White Paper, 2001. http://www.ximl.org/Docs.asp.

[215] RAE. http://www.rae.es, 1992.

[216] RANGANATHAN, A., AND CAMPBELL, R. A middleware for context-aware agents in ubiquitous computing environments. In *Middleware 2003, ACM/IFIP/USENIX International Middleware Conference* (2003), vol. 2672 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 143–161.

[217] RANGANATHAN, A., CAMPBELL, R., RAVI, A., AND MAHAJAN, A. Conchat: A context-aware chat program. *IEEE Pervasive Computing* (July-Sept 2002), 52–58.

[218] RANGANATHAN, A., MCGRATH, R. E., CAMPBELL, R. H., AND MICKUNAS, D. Use of ontologies in a pervasive computing environment. *The Knowledge Engineering Review 18*, 3 (September 2003), 209–220.

[219] RAO, S. P., AND COOK, D. J. Predicting inhabitant actions using action and task models with application to smart homes. *International Journal of Artificial Intelligence Tools 13*, 1 (2004), 81–100.

[220] RATTAPOOM, T. *Security and Privacy in the Intelligent Room.* PhD thesis, MIT, 2002.

[221] RHODES, B. J. Using physical context for just-in-time information retrieval. *IEEE Transactions on Computers 52*, 8 (2003), 1011–1014.

[222] RHODES, B. J., AND STARNER, T. A continuously running automated information retrieval system. In *Proceedings of The First International Conference on The Practical Application Of Intelligent Agents and Multi Agent Technology* (1996), pp. 487–495.

[223] RIESENBACH, R. The ontario telepresence project. In *Conference on Human Factors in Computing Systems* (Boston, Massachusetts, United States, 1994), ACM Press, pp. 173–176.

[224] RIESGO, T. Wems: Una visión electrónica de los sistemas vestibles. In *Computación Ubicua e Inteligencia Ambiental, CAEPIA'03* (San Sebastian, Spain, 11 de Noviembre 2003).

[225] R.M., T. Context-sensitive reasoning for autonomous agents and cooperative distributed problem solving. In *IJCAI-93 Workshop on Using Knowledge In Its Context* (Chambéry, France, 1993).

[226] ROMÁN, M., HESS, C. K., CERQUEIRA, R., RANGANATHAN, A., CAMPBELL, R. H., AND NAHRSTEDT, K. Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing* (Oct-Dec 2002), 74–83.

[227] ROMÁN, M., HESS, C. K., RANGANATHAN, A., MADHAVARAPU, P., BORTHAKUR, B., VISWANATHAN, P., . CERQUEIRA, R., CAMPBELL, R. H., AND MICKUNAS, M. D. Gaiaos: An infrastructure for active spaces. Technical Report UIUCDCS-R-2001-2224 UILU-ENG-2001-1731, University of Illinois, Urbana-Champaign, 2001.

[228] RUSSEL, S., AND NORVIG, P. *Artificial Intelligence: A Modern Approach.* Prentice Hall International, 1995.

[229] RYAN, N. Contextml: Exchanging contextual information between a mobile client and the fieldnote server. http://www.cs.kent.ac.uk/projects/mobicomp/fnc/ConteXtML.html, 1999.

[230] RYAN, N. S., PASCOE, J., AND MORSE, D. R. Enhanced reality fieldwork: the context-aware archaeological assistant. In *Computer Applications in Archaeology 1997* (Oxford, October 1998), V. Gaffney, M. van Leusen, and S. Exxon, Eds., British Archaeological Reports, Tempus Reparatum.

[231] SANCHO, G. D.-A., RIOJA, R. M. G., AND VÁZQUEZ, M. C. C. Design of a fipa compliant agent platform for limited devices. In *MATA 2003 5th International Workshop on Mobile Agents for Telecommunications Applications* (Morocco, October 8-10 2003), vol. 2881 of *Lecture Notes in Computer Science.*

[232] SATO, K. Context sensitive interactive systems design: A framework for representation of contexts. In *10th International Conference on Human-Computer Interaction, HCI International 2003* (Crete, Greece, June 22-27 2003).

[233] SATYANARAYANAN, M. Pervasive computing: Vision and challenges. *IEEE Personal Communications 8*, 4 (August 2001), 10–17.

[234] SCHILIT, B., ADAMS, N., AND WANT, R. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications* (Santa Cruz, CA, US, 1994).

[235] SCHILIT, B., AND THEIMER, M. Disseminating active map information to mobile hosts. *IEEE Network 8*, 5 (1994), 22–32.

[236] SCHILIT, B.Ñ. Mega-utilities drive invisible technologies. *IEEE Computer 36*, 2 (february 2003), 97–99.

[237] SCHILIT, B.Ñ., THEIMER, M. M., AND WELCH, B. B. Customizing mobile applications. In *Proceedings of USENIX Mobile & Location-Independent Computing Symposium* (Cambridge, Massachusetts, 1993), USENIX Association, pp. 129–138.

[238] SCHILIT, W. *A system architecture for context-aware mobile computing.* PhD thesis, Columbia University, 1995.

[239] SCHMIDT, A. Implicit human computer interaction through context. *Personal and Ubiquitous Computing 4*, 2/3 (2000).

[240] SCHMIDT, A., AIDOO, K. A., TAKALUOMA, A., U.TUOMELA, LAERHOVEN, K. V., AND DE VELDE, W. V. Advanced interaction in context. In *Handheld and Ubiquitous Computing* . (september 1999), H.-W. Gellersen, Ed., vol. 1707 of *LNCS*, First International Symposium. HUC'99. Karlsruhe, Germany, Springer - Verlag, pp. 89–101.

[241] SCHMIDT, A., BEIGL, M., AND GELLERSEN, H.-W. There is more to context than location. *Computers and Graphics 23*, 6 (1999), 893–901.

[242] SCHMIDT, A., DE VELDE, W. V., AND KORTUEM, G., Eds. *Situated Interaction in Ubiquitous Computing* (The Hague, The Netherlands, April 2000), Computer Human Interaction 2000 (CHI 2000). http://www.teco.edu/chi2000ws/.

[243] SCHMIDT, A., AND LAERHOVEN, K. V. How to build smart appliances. *IEEE Personal Communications 8*, 4 (August 2001), 66–71.

[244] SCHUCKMANN, C., KIRCHNER, L., SCHUMMER, J., AND HAAKE, J. Designing object-oriented synchronous groupware with coast. *Computer Supported Cooperative Work* (1996), 30–38.

[245] SCHULTZ, T., WAIBEL, A., BETT, M., METZE, F., PAN, Y., RIES, K., SCHAAF, T., SOLTAU, H., WESTPHAL, M., YU, H., AND ZECHNER, K. The isl meeting room system. In *Proceedings of the Workshop on Hands-Free Speech Communication (HSC-2001)* (Kyoto Japan, April 2001).

[246] SELKER, T., ARROYO, E., AND BURLESON, W. Chameleon tables: using context information in everyday objects. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems* (2002), ACM Press, pp. 580–581.

[247] SENDÍN, M., GRANOLLERS, T., LORÉS, J., AGUILÓ, J., AND BALAGUER, A. Un modelo interactivo ubicuo aplicado al patrimonio natural y cultural del montsec. *Revista Iberoamericana de Inteligencia Artificial*, 16 (2002).

[248] SENDÍN, M., LORÉS, J., AGUILÓ, C., AND BALAGUER, A. Un modelo interactivo ubicuo aplicado al patrimonio natural y cultural del área del montsec. *Revista Novática*, 153 (January 2001), 22–25.

[249] SHAFER, S. Ten dimensions of ubiquitous computing. In *Proceedings of 1st International Workshop on Managing Interactions in Smart Environments* (London, 1999), P.Ñixon, G. Lacey, and S. Dobson, Eds., MANSE'99, Dublin, Dec 1999, Springer-Verlag, pp. 5–16.

[250] SMAILAGIC, A., MARTIN, R., RYCHLIK, B., ROWLANDS, J., AND OZCERI, B. Metronaut: A wearable computer with sensing and global communication capabilities. *Personal and Ubiquitous Computing 1*, 3 (1997).

[251] SMITH, M. Smartcards: Integrating for portable complexity. *IEEE Computer*, 115 (August 1998), 110–112.

[252] STAJANO, F. *Security for Ubiquitous Computing.* John Wiley and Sons, Febreary 2002.

[253] STANFORD, V. Using pervasive computing to deliver elder care. *IEEE Pervasive Computing 1*, 1 (January 2002), 10–13.

[254] STEFIK, M., FOSTER, G., BOBROW, D. G., KAHN, K., LANNING, S., AND SUCHMAN, L. Beyond the chalkboard: computer support for collaboration and problem solving in meetings. *Communications of the ACM 30*, 1 (1987), 32–47.

[255] STIEFELHAGEN, R. Tracking focus of attention in meetings. In *ICMI '02: Proceedings of the 4th IEEE International Conference on Multimodal Interfaces* (Pittsburgh, Pennsylvania, October 2002), IEEE Computer Society.

[256] STRANG, T., AND LINNHOFFPOPIEN, C. A context modeling survey, 2004.

[257] STREITZ, N. A., GEIßLER, J., HOLMER, T., KONOMI, S., MüLLER-TOMFELDE, C., REISCHL, W., REXROTH, P., SEITZ, P., AND STEINMETZ, R. i-land: an interactive landscape for creativity and innovation. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems* (1999), ACM Press, pp. 120–127.

[258] STREITZ, N. A., GEISSLER, J., AND HOLMER, T. Roomware for cooperative buildings: Integrated design of architectural spaces and information space. In *Proceedings of the First International Workshop on Cooperative Buildings, Integrating Information, Organization, and Architecture* (London, UK, 1998), LNCS, Springer-Verlag, pp. 4–21.

[259] SUMI, Y., ETANI, T., FELS, S., SIMONET, N., KOBAYASHI, K., AND MASE, K. C-map: Building a context-aware mobile assistant for exhibition tours. In *Community Computing and Support Systems: Social Interaction in Networked Communities* (November 1998), T. Ishida, Ed., no. 1519 in Lecture Notes in Computer Science, Springer, pp. 137–154.

[260] SUMI, Y., AND MASE, K. Conference assistant system for supporting knowledge sharing in academic communities. *Interacting with Computers 14*, 6 (2002), 713–737.

[261] SUSPERREGIA, L., MAURTUA, I., TUBÍO, C., SEGOVIA, I., PÉREZ, M., AND SIERRA, B. Inteligencia ambiental en fabricación: prototipo amico, agentes sensibles al contexto para nuevos entornos de trabajo. In *I Simposio Computación Ubicua Inteligencia Ambiental, UCAmI'05* (Granada, Spain, 2005), Thomson, pp. 295–304.

[262] TANDLER, P. The beach application model and software framework for synchronous collaboration in ubiquitous computing environment. *Journal of Systems and Software 69*, 3 (January 2004), 267–296. Special issue: Ubiquitous computing.

[263] TANDLER, P., STREITZ, N. A., AND PRANTE, T. Roomware-moving toward ubiquitous computers. *IEEE Micro 22*, 6 (2002), 36–47.

[264] TANG, J. C., YANKELOVICH, N., BEGOLE, J., KLEEK, M. V., LI, F. C., AND BHALODIA, J. R. Connexus to awarenex: extending awareness to mobile users. In *CHI2001 Proceedings of the SIG-CHI on Human factors in computing systems* (Seattle, WA, USA, 2001), ACM, pp. 221–228.

[265] TAPIA, E. M., INTILLE, S. S., AND LARSON, K. Activity recognition in the home using simple and ubiquitous sensors. In *Pervasive Computing, Second International Conference, PERVASIVE 2004* (Vienna, Austria, April 2004), vol. 3001 of *Lecture Notes in Computer Science*, Springer, pp. 158–175.

[266] TRIVEDI, M., HUANG, K., AND MIKIC, I. Intelligent environments and active camera networks. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, volume 2* (2000), pp. 804–809.

[267] TRUONG, K., AND ABOWD, G. Inca: A software infrastructure to facilitate the construction and evolution of ubiquitous capture & access applications. In *In the Proceedings of Pervasive 2004: The Second International Conference on Pervasive Computing* (Linz,Austria, April 2004), pp. 140–157.

[268] UNCLE ROY ALL AROUND YOU. http://www.uncleroyallaroundyou.co.uk/, 2003.

[269] URIBARREN, A., PARRA, J., URIBE, J. P., ZAMALLOA, M., AND MAKIBAR, K. Middleware para servicios distribuidos y aplicaciones móviles. In *I Simposio Computación Ubicua Inteligencia Ambiental, UCAmI'05* (Granada, Spain, 2005), Thomson, pp. 241–250.

[270] VERDEJO, M., BARROS, B., READ, T., AND RODRIGUEZ-ARTACHO, M. A system for the specification and development of an environment for distributed cscl scenarios. In *Intelligent Tutoring Systems, 6th International Conference* (2002), S. Cerri, G. Gouardères, and F. Paraguaçu, Eds., LNCS, Springer-Verlag.

[271] VOELKER, G. M., AND BERSHAD, B.Ñ. Mobisaic: An information system for a mobile wireless computing environment. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications* (Santa Cruz, California, 1994), IEEE Computer Society Press, pp. 185–190.

[272] VOICE, D. http://www.dementiavoice.org.uk/Projects/Projects_GloucesterProject.htm, 2000.

[273] W3C. Composite capability/preference profiles (cc/pp). W3C Recommendation http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/, World Wide Web Consortium, 2004.

[274] WALDO, J. The jini architecture for network-centric computing. *Communications of the ACM* (July 1999), 76–82.

[275] WANG, X. H., DONG, J. S., CHIN, C., AND HETTIARACHCHI, S. R. Semantic space: an infrastructure for smart spaces. *IEEE Pervasive Computing 3*, 3 (July-Sept 2004), 32–39.

[276] WANG, X. H., ZHANG, D. Q., GU, T., AND PUNG, H. K. Ontology based context modeling and reasoning using owl. In *Proceedings of PerCom 2004* (Orlando, FL, USA, march 2004), pp. 18–22.

[277] WANT, R., BORRIELLO, G., PERING, T., AND FARKAS, K. I. Disappering hardware. *IEEE Pervasive Computing 1*, 1 (January-March 2002), 36–47.

[278] WANT, R., HOPPER, A., FALCÃO, V., AND GIBBONS, J. The Active Badge location system. *ACM Transactions on Information Systems 10*, 1 (January 1992), 91–102.

[279] WANT, R., PERING, T., DANNEELS, G., KUMAR, M., SUNDAR, M., AND LIGHT, J. The personal server: Changing the way we think about ubiquitous computing. In *In Proceedings of the 4th international conference on Ubiquitous Computing (Ubicomp 2002)* (Göteborg, Sweden, 2002), G. Borriello and L. E. Holmquist, Eds., vol. 2498 of *LNCS*, Springer-Verlag, pp. 194–209.

[280] WANT, R., SCHILIT, B., ADAMS, N., GOLD, R., PETERSEN, K., GOLDBERG, D., ELLIS, J., AND WEISER, M. An overview of the parctab ubiquitous computing experiment. *IEEE Personal Communications 2*, 6 (december 1995), 28–43.

[281] WARD, A., JONES, A., AND HOPPER, A. A new location technique for the active office. *IEEE Personal Communications 4*, 5 (October 1997), 42–47.

[282] WARD, A. M. R. *Sensor-driven computing.* PhD thesis, University of Cambridge, 1999.

[283] WARNEKE, B., LAST, M., LIEBOWITZ, B., AND PISTER, K. S. J. Smart dust: Communicating with a cubic-millimeter computer. *IEEE Computer 34*, 1 (2001), 44–51.

[284] WEAL, M. J., MICHAELIDESA, D. T., THOMPSON, M. K., AND , D. C. D. R. The ambient wood journals - replaying the experience. In *Proceedings of ACM Hypertext'03, The fourteenth conference on Hypertext and Hypermedia* (Nottingham, UK, 2003).

[285] WEBSPHERE. http://www.ibm.com/websphere, 2003.

[286] WEISER, M. The computer for the 21st century. *Scientific American* (September 1991).

[287] WEISER, M. Some computer science issues in Ubiquitous Computing. *Communications of ACM 36*, 7 (July 1993), 75–84.

[288] WEISER, M. The world is not a desktop. *ACM Interactions 1*, 1 (1994), 7–8.

[289] WEISER, M., AND BROWN, J. S. Designing calm technology. *PowerGrid Journal 1*, 1 (1996).

[290] WEISER, M., AND BROWN, J. S. *Beyond Calculation: The Next Fifty Years of Computing.* Springer-Verlag, New York, 1997, ch. The Coming Age of Calm Technology, pp. 75–86.

[291] WEISER, M., GOLD, R., AND BROWN, J. S. The origins of ubiquitous computing research at PARC in the late 1980s. *IBM Systems Journal 38*, 4 (1999), 693–696.

[292] WINOGRAD, T. Architectures for context. *Human-Computer Interaction 16*, 2,3 & 4 (2001), 401–419. Lawrence Erlbaum Associates.

[293] WINOGRAD, T., AND FLORES, F. *Understanding Computers and Cognition : A New Foundation for Design.* Addison-Wesley Professional, 1987.

[294] WOOD, K. R., RICHARDSON, T., BENNETT, F., HARTER, A., AND HOPPER, A. Global teleporting with java: Toward ubiquitous personalized computing. *IEEE Computer 30*, 2 (1997), 53–59.

[295] WORDNET. http://wordnet.princeton.edu/, 1997.

[296] THE WORKING GROUP FOR WLAN STANDARDS. *IEEE 802.11. Wireless LAN Médium Access Control and Physical Layer Specification*, 1997.

[297] WYCKOFF, P., MCLAUGHRY, S., LEHMAN, T., AND FORD, D. Tspaces. *IBM Systems Journal 37*, 3 (1998), 454–474.

[298] X-10 INC. *The X10 Specification.* USA, 91 Ruckman Rd., Box 420, Closter, NJ 07624.

[299] YAN, H., AND SELKER, T. Context-aware office assistant. In *Proceedings of the 2000 International Conference on Intelligent User Interfaces* (New Orleans, LA, USA, January 2000), IUI 2000, pp. 276–279.

[300] YANG, J., YANG, W., DENECKE, M., AND WAIBEL, A. Smart sight: a tourist assistant system. In *Proceedings of the 3rd International Symposium on Wearable Computers* (San Francisco, California, October 1999), pp. 73–78.

Reunido el tribunal que suscribe en el día
de la fecha, acordó calificar la presente Tesis
doctoral con SOBRESALIENTE CUM LAUDE
Madrid, 18 de Mayo de 2006

PRESIDENTE:

JULIO ABASCAL GONZÁLEZ

SECRETARIO

FRANCISCO GÓMEZ ARRIBAS

VOCAL 1

MIGUEL GEA MEGÍAS

VOCAL 2

JOSÉ BRAVO RODRÍGUEZ

VOCAL 3

FRANCISCO JOSÉ BALLESTEROS