# HI3: una aproximación integrada a la construcción de sistemas de Inteligencia Ambiental

Autor: Alejandro Paz López

Tese de doutoramento UDC / 2015

Directores:

Richard J. Duro Fernández

José Antonio Becerra Permuy

Programa de doutoramento en computación

UNIVERSIDADE DA CORUÑA

UNIVERSIDADE DA CORUÑA

**D. Richard J. Duro Fernández**, Catedrático de Universidad del Departamento de Computación de la Universidade da Coruña,

**D. José Antonio Becerra Permuy**, Contratado Doctor del Departamento de Computación de la Universidade da Coruña,

CERTIFICAN:

Que la memoria titulada:

"HI3: Una Aproximación Integrada a la Construcción de Sistemas de Inteligencia Ambiental"

ha sido realizada por **D. Alejandro Paz López** bajo nuestra dirección en el Departamento de Computación de la Universidade da Coruña, y constituye la Tesis que presenta para optar al grado de Doctor.

Fdo. Richard J. Duro Fernández
*Codirector de la Tesis Doctoral*

Fdo. José A. Becerra Permuy
*Codirector de la Tesis Doctoral*

# Resumen

El objetivo de la Inteligencia Ambiental (IAm) es mejorar la calidad de vida de las personas a través de la integración transparente de tecnologías en sus entornos. Dentro de la IAm, este trabajo se centra en facilitar el desarrollo de sistemas de IAm ubicuos y capaces de adaptarse dinámicamente a las características de distintos escenarios, poniendo énfasis en la adaptación a las necesidades de los usuarios y en la adaptación al entorno físico. Los entornos de IAm son altamente heterogéneos y abiertos. Esta diversidad hace que resulte muy difícil anticipar todas las condiciones en las que un sistema va a operar, complicando el desarrollo de aplicaciones capaces de adaptarse a distintos contextos.

Esta tesis aborda el proceso de construcción de sistemas de IAm desde una aproximación integrada y uniforme. Para ello, se propone un modelo conceptual y una arquitectura software que faciliten el desarrollo de aplicaciones de IAm aislando al desarrollador de la heterogeneidad de tecnologías y recursos que pueden estar presentes en estos entornos. Además, también se proporciona una implementación de la arquitectura que posibilita la creación de sistemas de IAm adaptables en base a la colaboración entre componentes interoperables.

# Resumo

O obxectivo da Intelixencia Ambiental (IAm) é mellorar a calidade de vida das persoas a través da integración transparente de tecnoloxías nas súas contornas. Dentro da IAm, esta tese céntrase en facilitar o desenvolvemento de sistemas de IAm ubicuos e capaces de adaptarse dinámicamente ás características de distintos escenarios, poñendo énfases na adaptación ás necesidades dos usuarios e na adaptación á contorna física. As contornas de IAm son altamente heteroxéneas e abertas. Esta diversidade fai que resulte moi difícil anticipar todas as condicións nas que un sistema vai operar, complicando o desenvolvemento de aplicacións capaces de adaptarse a distintos contextos.

Esta tese aborda o proceso de construción de sistemas de IAm desde unha aproximación integrada e uniforme. Para iso, proponse un modelo conceptual e unha arquitectura software que faciliten o desenvolvemento de aplicacións de IAm illando ao desenvolvedor de software da heteroxeneidade de tecnoloxías e recursos que poden estar presentes nestas contornas. Ademais, tamén se proporciona unha implementación da arquitectura que posibilita a creación de sistemas de IAm adaptables en base á colaboración entre compoñentes interoperables.

# Abstract

Ambient Intelligence (AmI) is about systems that assist people to improve their quality of life through the seamless integration of technologies in their environments. This work is focused on facilitating the development of ubiquitous AmI systems that are able to dynamically adapt to the characteristics of different scenarios, emphasizing the adaptation to the user needs and the physical environment. AmI environments are open and highly heterogeneous. This diversity makes it very difficult to anticipate all the conditions under which a system will operate, complicating the development of applications that can adapt to different contexts.

This thesis deals with the construction process of AmI systems from an integrated and uniform approach. For this purpose, a conceptual model and a software architecture that facilitates the development of AmI applications, isolating the developer from the heterogeneous technologies and resources that may be present in these environments, are proposed. In addition, an implementation of the architecture that enables the creation of adaptable AmI systems based on the collaboration between interoperable components is also provided.

# Agradecimientos

En primer lugar, quiero agradecerles a mis directores, Richard y José Antonio, su apoyo e inspiración a lo largo de estos años. Sin su confianza e iniciativa este trabajo nunca habría sido posible.

También me gustaría dar las gracias a todos los compañeros con los que he compartido esfuerzos, alegrías y reveses en el proceso de dar vida a una nueva línea de investigación en Inteligencia Ambiental dentro del Grupo. Muy especialmente a Gervasio, con el que he tenido la fortuna de colaborar desde el inicio y que siempre ha contribuido a hacer la tarea más amena y enriquecedora. Igualmente a Víctor, que ha contribuido a dar un notable empujón al desarrollo del proyecto HI3. Con él también he compartido innumerables discusiones y otras tantas batallas contra un interminable universo de tecnologías software no siempre amigables. Andrés, que ha aportado su visión y trabajo desde el ingrato lado del hardware y que siempre ha tenido unas palabras de ánimo cuando las he necesitado. Santi, por su energía y determinación en sacar muchas de nuestras ideas adelante. Alma, por su ayuda en el proceso de construir los entornos físicos experimentales utilizados en este trabajo. A todo el *sector hardware* del laboratorio, Álvaro, Félix y Martín, que han posibilitado la transformación de conceptos en dispositivos físicos. Sin olvidarme de Juan Carlos, por el gran diseño de la portada de este trabajo y por sus consejos en el diseño gráfico de las aplicaciones.

Por supuesto, quiero darle las gracias a todos los miembros del Grupo Integrado de Ingeniería por crear el mejor ambiente de trabajo que uno pueda desear. Por su apoyo, por sus ideas, por las risas... que a menudo se convierten en carcajadas. Por esos cafés en los que lo que importa es la conversación. En definitiva, por haberse convertido en algo más que compañeros de trabajo.

También me gustaría dar las gracias a las empresas cuyos proyectos de investigación han servido como casos de prueba para algunas de las tecnologías desarrolladas en esta tesis. MyTech Ingeniería Aplicada, por su colaboración en el desarrollo de la tecnología ONIZE, utilizada como base para la construcción de uno de los ejemplos de este trabajo. SCIO Innovation Technologies, por su colaboración en el

desarrollo del sistema OMNI, otro de los ejemplos utilizados en esta tesis.

Para finalizar me gustaría aprovechar para expresarle mi agradecimiento a las personas que han marcado una mayor diferencia en mi vida.

A mis padres, por tener mi educación como una prioridad en su vida y por enseñarme desde pequeño a valorar las cosas que realmente importan. Muy especialmente a mi madre, por darme siempre más de lo que tenía y porque su afecto es lo que en muchos aspectos me ha hecho ser lo que soy.

A María, por estar siempre a mi lado y por apoyarme en estos últimos meses como sólo ella sabe hacerlo.

A mis amigos, que también han hecho este camino conmigo y con los que seguro recorreré muchos más.

# Publicaciones

Durante la realización de esta tesis se han publicado los siguientes trabajos en congresos y revistas científicas:

- Gervasio Varela, Alejandro Paz-Lopez, Jose Antonio Becerra Permuy, Richard J. Duro Fernandez, Prototyping Distributed Physical User Interfaces in Ambient Intelligence Setups, Proceedings of the 2nd International Conference on Distributed, Ambient and Pervasive Environments (DAPI 2014), pp 76 - 85, 2014.
- Varela G., Paz-Lopez A., Becerra J.A. and Duro R.J, The Generic Interaction Protocol: Increasing portability of distributed physical user interfaces, Romanian Journal of Human - Computer Interaction, pp 249 - 268, 2013.
- A. Paz-Lopez, G. Varela, J.A. Becerra, S. Vazquez-Rodriguez, R.J. Duro, Towards ubiquity in ambient intelligence: User-guided component mobility in the HI3 architecture, Science of Computer Programming, pp 1971 - 1986, 2013.
- G. Varela, A. Paz-Lopez, J. A. Becerra , R. J. Duro, Decoupled Distributed User Interface Development Framework for Ambient Intelligence Systems, 3rd Workshop on Distributed User Interfaces: Models, Methods and Tools, DUI 2013, pp 23 - 26, 2013.
- F. López Peña, A. Paz-Lopez, G. Varela, R. J. Duro, Dynamic Maps of Urban Air Pollutants from Vehicle Based Opportunistic Sensor Network Data, Environmental Semeiotics, pp 31 - 39, 2012.
- A. Paz-Lopez, G. Varela, V. Sonora, J. A. Becerra, DAAF: a Device Abstraction and Aggregation Framework for Smart Environments, Proceedings of The Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, pp 739 - 744, 2012.
- G. Varela, A. Paz-Lopez, J. A. Becerra, S. Vazquez-Rodriguez, R. J. Duro, Towards Mobility in Ambient Intelligence: Component Migration and Adaptation Strategies in the HI3 Architecture, 5th International Symposium on Ubiquitous Computing and Ambient Intelligence (UCAmI 2011), 2011.

- Gervasio Varela, Alejandro Paz-Lopez, Jose Antonio Becerra, Santiago Vazquez-Rodriguez and Richard José Duro, UniDA: Uniform Device Access Framework for Human Interaction Environments, Sensors, pp 9361 - 9392, 2011.
- R. J. Duro, F. Bellas, A. Prieto, A. Paz-Lopez, Social Learning for Collaboration through ASiCo based Neuroevolution, Journal of Intelligent and Fuzzy Systems, pp 1 - 1, 2011.
- A. Paz-Lopez, G. Varela, S. Vazquez-Rodriguez, J. A. Becerra and R. J. Duro, Some Issues and Extensions of JADE to Cope with Multi-agent Operation in the Context of Ambient Inteligence, Advances in Intelligent and Soft Computing, pp 607 - 614, 2010.
- A. Paz-Lopez, G. Varela, S. Vazquez-Rodriguez, J. A. Becerra and R. J. Duro, Integrating Ambient Intelligence Technologies Using an Architectural Approach, Ambient Intelligence, pp 1 - 26, 2010.
- A. Paz-Lopez, G. Varela, J.Monroy, S.Vazquez-Rodriguez, R.J. Duro, HI3 Project: Software Architecture System for Elderly Care in a Retirement Home, Advances in Soft Computing, pp 11 - 20, 2008.
- A. Paz-Lopez, G. Varela, J. Monroy, S. Vazquez-Rodriguez, R. J. Duro, HI3 Project: General Purpose Ambient Intelligence Architecture, Proceedings of the 3rd Workshop on Artificial Intelligence Techniques for Ambient Intelligence (AITAmI'08), pp 77 - 81, 2008.

Además, se han promovido las siguientes patentes fruto de colaboraciones con empresas:

- A. Paz López, G. Varela Fernández , S. Vázquez Rodríguez, A. Faiña Rodrígez-Vila, J.A. Becerra Permuy, A. Deibe Díaz, R. Duro Fernández, F. López Peña. Terminal instrumentation node for a self-configuring and a secure building automation system. ES2400893, EP2592810, US2014148952, JP2014514885. MyTech Ingeniería Aplicada S.L. 2012.
- P. Bermúdez Pestonit, I. Raño Jares, B. Mosquera Collazo, A. Paz López, G. Varela Fernández, F. López Peña, A. Faiña Rodríguez, R. Duro Fernández. Sistema y método de tele-asistencia mediante televisor inteligente. P201431701. SCIO Innovation Technologies. 2014.

# Índice general

# Capítulo 1

# Introducción

La Inteligencia Ambiental (IAm) persigue la integración transparente de la tecnología en todos los entornos de las personas para asistirlas, anticipando sus necesidades, y para, en último término, mejorar su calidad de vida. Estos sistemas deben adaptarse al contexto del usuario y acompañarlo en sus tareas, basándose en la información que son capaces de obtener en tiempo real del entorno y en el conocimiento aprendido y acumulado. Esta tesis doctoral se centra en el problema de construir sistemas de IAm extensibles y adaptables a cualquier entorno, basándose en la colaboración dinámica entre componentes funcionales y la integración de tecnologías existentes. Para ello, se propone una visión integrada del proceso de desarrollo de sistemas de IAm adaptada a la realidad actual de un mundo conectado en el cada vez que los computadores están más integrados en el entorno, y dónde la movilidad y ubicuidad de los sistemas comienza a ser percibida como una necesidad. El proceso estará guiado por unos principios de diseño, unos modelos conceptuales y una arquitectura software abstracta, y estará soportado por una arquitectura concreta y un middleware de referencia.

El campo de la Inteligencia Ambiental conforma un área de investigación joven y multidisciplinar en la que tienen cabida áreas como inteligencia artificial, comunicaciones, interfaces naturales de usuario o electrónica. Las primeras ideas relacionadas con la IAm se remontan al concepto de *computación ubicua* y se plasman en el año 1991 en el artículo de Mark Weiser The Computer of the 21st Century [Weiser, 1991]. En dicho artículo, Weiser describía su visión de un mundo poblado de computadores que, integrados en el entorno, participarían activamente en la vida de las personas de forma natural y transparente para ellas. Siguiendo en gran medida estas ideas, en el año 1998 Philips acuña por primera vez el término Inteligencia Ambiental [Zelkha, 1998]. La palabra *inteligencia* hace referencia a la respuesta que el usuario espera del sistema en términos de proactividad, predictivilidad y

adaptatividad en sus comportamientos. Por otra parte, la palabra *ambiental* está relacionada con la necesidad de integración de la tecnología en los objetos cotidianos y en los entornos de forma no intrusiva. Esta visión de la IAm sitúa a las necesidades del usuario como elemento clave del desarrollo tecnológico; no es el usuario el que se adapta a la tecnología sino al revés. Todos estos conceptos teóricos comienzan a materializarse en mayor o menor medida en distintos proyectos que se realizan en la década de los 90 y que abarcan entornos tan variados como la creación de *hogares inteligentes*, la asistencia médica o la educación [Izso, 2009] [Larson, 2007] [Shi et al., 2003].

En último término, el principal objetivo de cualquier sistema de IAm debe ser el cumplimiento de las expectativas del usuario y la adaptación dinámica del sistema a sus necesidades, preferencias y hábitos. El usuario percibe los beneficios en cualquier de los entornos en que se desenvuelve, pero no percibe la tecnología. Es en este sentido en el que la IAm nos habla de la *tecnología que desaparece*. Así, desde la IAm, se imagina un proceso de desarrollo de aplicaciones independientes de la complejidad del acceso a los dispositivos del medio físico, capaces de moverse con el usuario trasladando su funcionalidad a dónde éste se encuentre, con un sistema dinámico para el descubrimiento y negociación con otros componentes, capaces de adaptar dinámicamente sus modos de interacción con el usuario en función de sus características y su entorno. Todas estas características hacen que los sistemas de IAm presenten algunos objetivos en común con otros paradigmas más maduros como los sistemas distribuidos o los sistemas de Inteligencia Artificial. Sin embargo, desde un punto de vista general, se trata de sistemas notablemente más complejos que éstos, debido a la naturaleza tan amplia y abierta de los entornos para los que han de diseñarse los sistemas de IAm.

Idealmente, los entornos de IAm abarcan a cualquier perfil de usuario y a cualquier espacio y contexto en el que las personas realicen sus actividades. No es posible restringirse a usuarios con unas determinadas habilidades tecnológicas y conocimiento. Tampoco es posible obviar la existencia de usuarios con distintas capacidades físicas y cognitivas; al contrario, las personas con limitaciones de este tipo deberían ser unas de las grandes beneficiadas de la utilización de la tecnología para disminuir su dependencia de otras personas. Por tanto, las aplicaciones deben ser capaces de adaptarse a las necesidades de las personas sin estar previamente programadas para un determinado tipo de usuario. Esto implica que los sistemas han de ser conscientes de las personas que hay en el entorno, identificarlas y aprender de sus comportamientos y costumbres. También implica que los sistemas deben ser capaces de descubrir y utilizar los recursos (componentes funcionales o dispositivos) que existan en el entorno en cada momento para interaccionar de la forma más adecuada posible con cada persona, para monitorizar su actividad o para extraer la información necesaria del entorno que permita a las aplicaciones formarse un contexto de la situación en base al que adaptar su operación. Ya no hablamos

de ordenadores que nos dan acceso a aplicaciones concretas para resolver tareas concretas, el acceso a los beneficios de la tecnología se obtiene de forma ubicua y natural, a través de los elementos del entorno en el que se encuentra el usuario en cada momento y a través de los dispositivos que lleva consigo.

Cuando hablamos de IAm nos enfrentamos a escenarios ubicuos, dónde los sistemas deben proporcionar su funcionalidad en cualquier momento y en cualquier lugar, soportando la movilidad del usuario y adaptándose en tiempo de ejecución a los diferentes entornos [Augusto, 2007]. Esto implica que en el momento de diseñar e implementar un sistema de IAm resulta muy difícil anticipar todas las condiciones en las que va a operar. Al contrario, habitualmente para completar sus tareas las aplicaciones necesitarán hacer uso de otros recursos funcionales que podrán variar de unos entornos a otros. Por tanto las aplicaciones deberán ser flexibles y tener cierta capacidad de adaptarse a los recursos disponibles. Para ello, no sólo es necesario que una aplicación sea capaz de encontrar otros recursos funcionales a priori desconocidos, también ha de ser capaz de entender las capacidades de estos, al igual que ha de entender el modo en que podrá interaccionar con ellos. Pero, como hemos anticipado, la complejidad de diseñar un sistema de IAm tan abierto y flexible no termina aquí. En entornos ubicuos, no es posible imponer o presuponer la existencia de una tecnología concreta, por ejemplo en aspectos relacionados con redes y protocolos de comunicación, plataformas de ejecución, redes de sensores/actuadores, paradigmas y medios de interacción con el usuario, etc. Se impone buscar la interoperabilidad entre sistemas y la integración de tecnologías. Se impone también adaptarse a contextos de ejecución en los que los recursos necesarios pueden no estar permanentemente disponibles, especialmente en el ámbito de la Computación Móvil.

Es por esta enorme complejidad y por los desafíos que presentan los entornos de IAm que a día de hoy la especificación e implementación de sistemas de este tipo se encuentra todavía en su infancia. Así, en los trabajos existentes nos encontramos con un gran número de enfoques que abordan la problemática a través de soluciones ad-hoc, que se restringen a problemas y entornos concretos y predefinidos [Tapia et al., 2004] [Brdiczka et al., 2009] [Bernardin et al., 2009] [Krumm et al., 2000]. Otros trabajos defienden la utilización de infraestructuras software que aporten soluciones reutilizables a algunos de los problemas comunes detectados en este ámbito. Este último es el caso de importantes proyectos desarrollados en los últimos años como AMIGO [Janse and Vink, 2008], SOPRANO [Sixsmith et al., 2009] y PERSONA [FP6-IST PERSONA, 2010] que proponen plataformas software basadas en los principios de la Computación Orientada a Servicios [Papazoglou et al., 2008]. También es el caso del proyecto [Soldatos et al., 2007] que propone una arquitectura para IAm en la que se utiliza una plataforma multi-agente para la representación y orquestación de los servicios del sistema. En cualquier caso, ninguna de las aproximaciones propuestas hasta el momento ha calado lo suficiente

en la comunidad científica como para establecerse como un punto de partida o una referencia clara a seguir a la hora de desarrollar sistemas de este tipo. Por el contrario, los intentos de desarrollo de soluciones concretas en este área han servido para poner de manifiesto de forma más precisa la enorme complejidad del ámbito y las dificultades tecnológicas para abordar dicha complejidad. Así, queda patente la necesidad de enfrentar de forma repetitiva problemas como el control de la sensorización y actuación del entorno; la obtención, integración y representación de la información contextual; la construcción de componentes reutilizables, interoperables y que se componen dinámicamente para resolver tareas. Por otra parte, se pone de manifiesto que otros aspectos como violaciones de la privacidad, interacción con el usuario excesivamente intrusiva o la deficiente integración de los elementos previamente presentes en el entorno, provocan en muchas ocasiones fuertes reticencias por parte del usuario a la hora de adoptar estas nuevas tecnologías.

En base al anterior diagnóstico, en esta tesis doctoral se propone una visión integradora del proceso de desarrollo de sistemas de IAm guiada por una arquitectura software de referencia, que hemos denominado como arquitectura HI3. Partiendo, para ello, de las lecciones aprendidas en investigaciones previas y centrando los objetivos tanto en facilitar a los desarrolladores la construcción de aplicaciones concretas en estos entornos como en mejorar la aceptación de soluciones de este tipo por parte de los usuarios. El planteamiento que se propone en esta tesis se guía por unas premisas muy claras. En primer lugar, se defiende que una arquitectura de referencia es el elemento de diseño central para dar respuesta a requisitos como escalabilidad, fiabilidad, modularidad, interoperabilidad, tolerancia a fallos, rendimiento, movilidad, ubicuidad, distribución o seguridad, todos ellos importantes a la hora de enfrentarse a entornos de enorme complejidad como son, con carácter general, aquellos en los que las personas realizan sus actividades cotidianas. En segundo lugar, la ausencia de una solución de referencia no debe llevarnos a descartar por completo modelos y sistemas previos, evitando erróneamente la posibilidad de utilizar e integrar modelos y tecnologías existentes que pueden aportar una base de partida razonable para distintos aspectos del problema. En tercer lugar, creemos que las soluciones propuestas deben ser lo más abiertas, adaptables y extensibles que sea posible, con el objetivo en mente de construir entornos amigables y adaptados a los usuarios, en los que se integren las tecnologías y elementos ya existentes, mejorando así la aceptación del sistema por parte de dichos usuarios. Por último, la proliferación de middleware en el campo de la IAm y en áreas relacionadas trae consigo un problema de middleware no interoperable, creando auténticas islas de middleware en todos los niveles en que puede estructurarse un sistema de IAm. Por ello, desde esta propuesta se buscará un equilibrio entre el desarrollo de soluciones integradoras, que busquen la interoperabilidad con tecnologías comunes, y mantener la arquitectura global suficientemente simple y asequible para los desarrolladores.

Una aproximación a la construcción de sistemas complejos que esté soportada por una arquitectura software bien fundamentada es una garantía a la hora de trasladar a las aplicaciones desarrolladas un conjunto de características deseadas, como pueden ser rendimiento, modularidad, reutilización, homogeneidad, tolerancia a fallos o seguridad [Bass et al., 2003]. Una arquitectura puede proporcionar modelos conceptuales comunes, principios de diseño y middleware que oculta la complejidad del entorno aislando a las aplicaciones de la gestión explícita de protocolos, fallos de comunicación, replicación de información o la gestión de la heterogeneidad en sistemas operativos, tecnologías de red, y lenguajes de programación. Por tanto, una arquitectura bien fundamentada no proporciona funcionalidades de IAm por sí misma, pero aporta un marco conceptual y una serie de herramientas para los diseñadores y desarrolladores que pueden agilizar el desarrollo y fomentar la interoperabilidad de las soluciones de IAm. Desafortunadamente, ninguno de los paradigmas de ingeniería del software existentes que están orientados al desarrollo de sistemas distribuidos, como la Computación Orientada a Servicios, Peer-To-Peer, o los Sistemas Multi-Agente, se encuentran en un estado de desarrollo y madurez suficientes como para responder a las necesidades de los sistemas de IAm. En consecuencia, existe una importante actividad investigadora que explora extensiones de los conceptos proporcionados por estos paradigmas y aproximaciones híbridas que combinan distintos paradigmas y distintas tecnologías.

Uno de los proyectos más destacados, debido a los avances logrados a la hora de construir una arquitectura de IAm es el proyecto AMIGO (Ambient Intelligence for the Networked Home Environment) [Janse and Vink, 2008]. Propone una arquitectura orientada a servicios que hace uso de tecnologías provenientes de la Web Semántica para la descripción y composición de servicios, proporcionando además interoperabilidad con diversos protocolos de comunicación RPC. Incorpora mecanismos limitados para la abstracción del acceso a dispositivos, que se ciñen a la utilización de UPnP [Donoho et al., 2008] y la definición de servicios ad-hoc para la abstracción concreta de cada dispositivo. Otros proyectos provenientes del área del Internet de las Cosas (IoT, del inglés Internet of Things), como Sense2Web [De et al., 2012], se centran en solucionar el problema del acceso e integración transparente de los dispositivos en los entornos de las personas. Estos ejemplos de proyectos sirven para ilustrar que, como se verá claramente en el capítulo de revisión de trabajos previos, se puede afirmar que el desarrollo de infraestructuras software en el ámbito de la IAm todavía está en su fase inicial si se comparan los trabajos existentes con la ambiciosa visión teórica de este área formulada hace ya más de una década. No existe, por tanto, una arquitectura que aborde y resuelva de forma conjunta todos los requisitos planteados en este ámbito. Además, muchos de los esfuerzos realizados en este sentido, lejos de establecerse en el tiempo como soluciones de referencia, han quedado descontinuados y se han visto sobrepasados por los nuevos contextos tecnológicos que vienen siendo impuestos por la eclosión de la Compu-

tación Móvil, el IoT y el Internet del Futuro [Miorandi et al., 2012] [Baresi et al., 2012] [FutureInternet, 2015]. Por otra parte, esta proliferación de tecnologías y servicios accesibles a través de redes de comunicaciones globales pone todavía más de manifiesto que la integración e interoperabilidad de aplicaciones realizadas en distintas infraestructuras software, pregonada por la IAm y la Computación Ubicua, continúan siento un desafío más que una realidad.

Consecuentemente con lo expuesto hasta ahora, la propuesta que se describirá a lo largo de esta tesis parte de la aceptación del hecho de que actualmente existe un enorme salto entre la visión proporcionada por la IAm y las tecnologías actuales. Además, creemos que se ha dado una conjunción de varios factores que han jugado en contra de las aproximaciones a la construcción de sistemas de IAm basadas en modelos y arquitecturas de referencia. En este sentido podemos destacar la enorme complejidad inherente a los entornos de IAm, la propia complejidad de algunas de las soluciones middleware propuestas, la gran fragmentación de esfuerzos y de aproximaciones heterogéneas y la descontinuación de proyectos originalmente ambiciosos. A todos estos factores hay que añadir la rápida evolución de los sistemas computacionales que se está produciendo en los últimos años, impulsada principalmente por la Computación Móvil, IoT y la visión del Internet del Futuro como una red global de servicios conectados, que en ciertos aspectos ha dejado obsoletas a aproximaciones previas. Partiendo de este diagnóstico, en este trabajo se propone que, asumiendo la complejidad del área, se ha de identificar un conjunto de requisitos fundamentales más acotado sobre el que construir una aproximación integrada con capacidad para extenderse en el futuro. Asimismo, se plantea como objetivo prioritario que el middleware mantenga el nivel de simplicidad en su uso adecuado para que cumpla con su objetivo inicial de facilitar y estructurar el desarrollo de sistemas complejos, en vez de suponer una barrera más para el desarrollo de éstos (cayendo, en tal caso, en la paradoja de animar a los desarrolladores a la implementación de soluciones ad-hoc).

Este trabajo se enmarca en el contexto de un proyecto de investigación multidisciplinar del Grupo Integrado de Ingeniería de la Universidade da Coruña denominado Proyecto HI3. Se trata de un proyecto íntimamente relacionado con líneas de investigación como la Inteligencia Ambiental, la Computación Ubicua, las Interfaces Naturales de Usuario o la Inteligencia Artificial. Su objetivo central es la investigación de tecnologías que permitan la creación de espacios en los que la computación se integre de forma transparente en el día a día de las personas, de forma que los usuarios perciban servicios que se adaptan a sus necesidades sin ser conscientes de como éstos se llevan a cabo. Los desarrollos del Grupo en este ámbito también han servido para asentar las ideas que se acaban de plantear. Así, en este contexto, surge la necesidad de integrar trabajos de investigación en áreas transversales a la IAm que son heterogéneos, de forma que puedan interoperar y explotar las características de un entorno de IAm sin necesidad de enfrentar repetidamente los mismos

problemas.

Para afrontar el desafío, desde el prisma de la ingeniería de arquitecturas software, se propone una visión conceptual del proceso de desarrollo de sistemas de IAm que especifica los requisitos, los principios de diseño y la estructura que deben guiar dicho proceso. Partiendo de este punto, se plantea una arquitectura abstracta fundamentada en algunos de los principios de las Arquitecturas Orientadas a Servicios, que constituyen un buen punto de partida para dotar a los sistemas de IAm de características deseadas como distribución, adaptación, autonomía o interoperabilidad. No obstante, debido a los desafíos pendientes de resolver a los que se enfrenta actualmente el paradigma SOC [Baresi et al., 2012], una parte fundamental de este trabajo consiste en construir una arquitectura orientada a servicios que adapte los principios SOA a los requisitos de la Computación Ubicua. En particular, el middleware orientado a servicios propuesto pondrá especial énfasis en: i) paliar el problema de la heterogeneidad a todos los niveles (representación del conocimiento, descripción de servicios, modelos de interacción entre componentes, etc.); ii) proponer una visión ubicua de los servicios que permita su utilización en cualquier plataforma, más allá de la preponderancia actual de las tecnologías de Servicios Web; iii) ampliar las capacidades descriptivas de los modelos clásicos de descripción de servicios.

Sobre la base de la arquitectura SOA para entornos ubicuos propuesta, se introduce otro concepto clave para la aproximación al proceso de construcción de sistemas de IAm de esta tesis. Se trata de extender la idea proveniente de SOC relativa a la utilización de los servicios como componentes flexibles y autónomos en base a los que implementar las funcionalidades de los sistemas. Dicha extensión se hace en el sentido de considerar que todo el middleware (frameworks, modelos conceptuales y herramientas) desarrollado para facilitar a los desarrolladores la tarea de lidiar con la complejidad de los entornos de IAm, se abstraiga también, en la medida de lo posible, como servicios. De esta manera, lo que convierte a nuestra arquitectura SOA para entornos ubicuos en una arquitectura para IAm es que ésta proporcione a los desarrolladores una serie de servicios semánticamente descritos, autónomos, descubribles e interoperables, que abstraen y encapsulan diferentes soluciones a problemas comunes de entornos de IAm. Esta aproximación permite tratar en parte con la heterogeneidad y fragmentación de las soluciones existentes para problemas comunes de IAm, propias de un área inmadura como ésta. Para ello, los servicios semánticos de IAm propuestos por nuestra aproximación permitirán: i) abstraer e integrar el acceso a la funcionalidad de middleware heterogéneo; ii) abstraer los modelos heterogéneos utilizados por diferentes aproximaciones middleware para un mismo requisito de IAm en modelos comunes representados en forma de ontologías (dando soporte asimismo a múltiples lenguajes de ontologías). Buscamos en última instancia, que los desarrolladores puedan centrarse más en la implementación de nuevas funcionalidades para los usuarios que en lidiar con la complejidad de

los entornos de IAm y la complejidad y heterogeneidad del middleware existente.

Así, siguiendo esta aproximación, como parte de esta tesis también se proponen otras soluciones relacionadas con requisitos de más alto nivel que, con carácter general, se consideran prioritarios para la construcción de un entorno de IAm. En este sentido, las principales aportaciones están relacionadas con el problema del acceso e interacción transparente con el entorno físico, así como con el desarrollo de aplicaciones capaces de adaptarse a las características y necesidades del usuario. Estas soluciones, representadas en el nivel más alto de abstracción como servicios, se integran en la arquitectura HI3, contribuyendo a la construcción de una arquitectura global que proporcione a los desarrolladores soluciones homogéneas para la construcción de nuevas funcionalidades en este tipo de entornos.

Finalmente, como parte de este trabajo también se exploran las posibilidades de utilización del paradigma de Sistemas Multi-Agente para la implementación de funcionalidades en entornos de IAm. Los agentes tienen características como autonomía, razonamiento, proactividad y movilidad que los hacen apropiados para el desarrollo de sistemas dinámicos y distribuidos en entornos de IAm, que demandan capacidades de adaptación a los usuarios y a las características de entornos cambiantes. Sin embargo se trata de un paradigma que presenta una clara carencia de herramientas que faciliten la construcción de sistemas complejos y en que general está menos estandarizado que el paradigma SOC. Como parte de esta tesis se propone una aproximación para la integración de ambos paradigmas de modo que dentro de la arquitectura HI3 sea posible realizar implementaciones concretas de funcionalidades como sistemas multi-agente que posteriormente exporten su funcionalidad pública como servicios.

## Organización de esta memoria

De acuerdo con las contribuciones que se acaban de describir, se estructura el documento de la tesis como sigue:

- En el capítulo 2 se describe el objetivo global de esta tesis así como los subobjetivos que necesariamente deben completarse para alcanzarlo.
- En el capítulo 3 se hace un análisis de las áreas de investigación y los trabajos más relevantes para esta tesis. Se introducen los conceptos fundamentales de IAm que se manejan en este trabajo, describiendo para ello el marco teórico y los objetivos centrales que caracterizan a este área de investigación. Sobre esta base, se identifica la problemática común con la que se ha de lidiar a la hora de construir cualquier sistema de IAm y se evalúan las alternativas que el campo de la Ingeniería del Software nos ofrece para abordar la complejidad de estos entornos. Asimismo, se hace una revisión crítica de otros trabajos

relacionados con la construcción de arquitecturas software que tienen como objetivo facilitar el desarrollo e implantación de sistemas de IAm.

- En el capítulo 4 se presenta una visión global de la arquitectura HI3, incluyendo sus elementos principales y los fundamentos en los que se sustenta el proceso integrado para el desarrollo de sistemas de IAm propuesto en esta tesis.
- En el capítulo 5 se describen los detalles del middleware orientado a servicios para entornos ubicuos propuesto como base de la arquitectura HI3.
- En el capítulo 6 se propone un modelo para la implementación de funcionalidades en la arquitectura orientada a servicios HI3 utilizando el paradigma de Sistemas Multi-Agente. Además, se presenta una realización concreta utilizando la plataforma de agentes Jade.
- En el capítulo 7 se describe un conjunto de soluciones, integradas en la arquitectura HI3, que facilitan a los desarrolladores la construcción de entornos de IAm dónde los sistemas se adaptan dinámicamente al entorno físico y a los usuarios. Estas soluciones son soportadas por el middleware orientado a servicios de la arquitectura.
- En el capítulo 8 se hace una evaluación de las soluciones propuestas a través del desarrollo de dos casos de sistemas de IAm.
- En el capítulo 9 se presentan las principales conclusiones derivadas del trabajo realizado en esta tesis doctoral junto con las principales líneas de investigación y desarrollo que permanecen abiertas de cara al futuro.

# Capítulo 2

# Objetivos

El objetivo general de esta tesis es proporcionar un soporte para la construcción de sistemas de Inteligencia Ambiental (IAm) partiendo de una aproximación uniforme e independiente de entornos concretos y de las características particulares de los usuarios.

De acuerdo con este objetivo general, en primer lugar, esta tesis persigue facilitar que los desarrolladores puedan centrar sus esfuerzos en la construcción de nuevas funcionalidades de IAm, evitando que tengan que enfrentarse de forma repetida a problemáticas comunes de este tipo de sistemas. En particular, proporcionando mecanismos generales que los aíslen de la heterogeneidad de tecnologías y recursos que pueden estar presentes en los entornos de IAm. En segundo lugar, esta tesis tiene como objetivo central facilitar el desarrollo de aplicaciones ubicuas capaces de adaptarse dinámicamente a las características de distintos escenarios, poniendo especial énfasis en la adaptación a las necesidades de los usuarios y en la adaptación al entorno físico.

Con el fin de focalizar más el trabajo de esta tesis, se dividen estos objetivos centrales en una serie de objetivos más específicos que se detallan a continuación:

- Identificar y analizar un conjunto reducido de requisitos clave comunes a la mayoría de sistemas de IAm que permita abarcar más fácilmente la complejidad de este tipo de sistemas y entornos.
- Proporcionar a los desarrolladores un marco conceptual común y unos principios de diseño que guíen el proceso de desarrollo de cualquier sistema de IAm. Buscando, por tanto, la mayor generalidad posible tanto en los requisitos como en el ámbito de aplicación.
- Concebir mecanismos que faciliten la implementación de componentes funcionales utilizando los paradigmas y las plataformas computacionales que los

desarrolladores consideren más adecuados en cada caso y a la vez soportando la interoperabilidad entre componentes y sistemas.

- Facilitar el desarrollo de sistemas capaces de colaborar, para la resolución de sus tareas, con otros componentes funcionales a priori desconocidos. Este objetivo implica que han de proporcionarse distintos mecanismos abstractos de interacción adecuados para distintos escenarios y que la complejidad y heterogeneidad de los mecanismos concretos de comunicación utilizados han de quedar ocultos al desarrollador.
- Reducir el coste, en tiempo y esfuerzo, de desarrollar aplicaciones capaces de operar en diferentes entornos físicos donde las personas realizan sus actividades, con independencia de las tecnologías de dispositivos y de comunicaciones necesarias para interaccionar con los mismos.
- Facilitar el desarrollo de aplicaciones capaces de operar con usuarios con diferentes características y preferencias, en especial en lo referente a la interacción con los mismos.
- Soportar la ubicuidad y movilidad de los sistemas de IAm gracias a la adaptación dinámica de sus componentes a la situación sin necesidad de una programación específica para cada escenario. Dónde la situación puede estar definida por las características de los usuarios presentes, por los recursos funcionales y físicos disponibles en el entorno, por la ubicación y características del entorno, etc.
- Posibilitar la construcción de sistemas de IAm escalables, capaces de soportar un número a menudo impredecible de elementos distribuidos (dispositivos, usuarios, redes de comunicación, etc.).
- Proporcionar acceso público a un conjunto de especificaciones y herramientas software de referencia que permitan a la comunidad científica tanto evaluar y utilizar los resultados como construir nuevas soluciones e implementaciones compatibles con los mismos.

# Capítulo 3

# Marco teórico

En este capítulo se enmarca la presente tesis doctoral dentro de los campos de la Inteligencia Ambiental y la Computación Ubicua con el objetivo fundamental de mostrar qué aporta en estos ámbitos y de justificar la idoneidad de las aproximaciones en las que se basa. El capítulo se ha dividido en cinco secciones principales. Una primera sección en la que se realiza una revisión de los principales conceptos de la Inteligencia Ambiental (IAm) y de los desafíos de investigación más relevantes que permanecen abiertos en este campo. En la segunda sección se realiza una revisión de aquellos paradigmas de ingeniería del software que por sus características pueden adaptarse, al menos en cierta medida, a los requisitos de la IAm. Seguidamente se hace una breve introducción al Internet de las Cosas (IoT), un campo especialmente en auge actualmente y que interseca en buena parte de sus objetivos con la IAm. A continuación, en la cuarta sección, se incluye un estudio detallado de las principales iniciativas y proyectos existentes que, al igual que este trabajo, tratan de lidiar con la complejidad de los entornos de IAm a través del diseño de infraestructuras software/hardware de carácter general. Por último, la quinta sección está centrada en las conclusiones extraídas del estudio del área, que servirán de punto de partida para la definición y desarrollo de las soluciones arquitectónicas a los principales requisitos de la IAm que se presentan en esta tesis.

## 3.1. Inteligencia Ambiental

El objetivo de esta sección es realizar una revisión general actualizada del campo de la Inteligencia Ambiental, introduciendo las diversas tecnologías y áreas de conocimiento que abarca esta área multidisciplinar. En primer lugar se hace una introducción a sus fundamentos y a las características principales que la definen. A continuación se trata de establecer, en base a la experiencia de los trabajos previos

en el área, cuales son los principales requisitos y desafíos que es necesario abordar para construir entornos inteligentes según la ambiciosa visión de la IAm.

### 3.1.1. Conceptos básicos

Los avances de la tecnología, en especial la miniturización de los componentes electrónicos, está permitiendo que cada vez una mayor diversidad de dispositivos forme parte de nuestra vida diaria. Así, es posible adquirir, a precios muy asequibles, una gran variedad de sensores, actuadores y elementos con capacidad de procesamiento. Esta tecnología puede, además, incorporar capacidades de comunicación que permiten tanto su conexión en red como su uso coordinado y controlado desde sistemas software inteligentes capaces de interpretar los eventos recibidos del entorno y tomar decisiones sensibles al contexto en tiempo real o a posteriori. En este sentido, la Inteligencia Ambiental está íntimamente relacionada con el concepto de *disappearing computer* [Streitz et al., 2007] [Weiser, 1991] [Weiser, 1993]:

> *Las tecnologías más trascendentales son aquellas que desparecen. Se camuflan en la vida diaria hasta que son indistinguibles de ella.*

Hace algunas décadas resultaba difícil imaginar un escenario en el que los ordenadores estuviesen camuflados en cada entorno de forma indistinguible. Hoy en día es posible integrar distintos dispositivos en los entornos donde realizamos la vida diaria, de forma que los utilicemos sin apenas notar su presencia. Los computadores han evolucionado en las últimas décadas desde enormes ordenadores que ocupaban habitaciones enteras hacia pequeños dispositivos que pueden ser integrados en una gran variedad de espacios. Existen por tanto fundadas razones para pensar que nuestras vidas se transformarán en las próximas décadas a través de la introducción de un amplio rango de dispositivos computacionales en todos los entornos. Estos dispositivos tendrán que ser coordinados por sistemas software que realizarán la integración de los recursos disponibles para crear “entornos inteligentes”. Es decir, el potencial no está en los dispositivos que pueden utilizar los usuarios sino en la interacción inteligente entre todos ellos y con el propio usuario. Este enfoque nos dirige hacia una confluencia de áreas tecnológicas tan variadas como las redes de sensores, la Inteligencia Artificial o la Interacción Persona-Ordenador, dando lugar a la creación del área denominada “Inteligencia Ambiental”:

> *Un entorno digital que de forma proactiva y sensible al contexto apoya a las personas en su vida diaria.* [Augusto, 2007]

El término Inteligencia Ambiental comienza a ser ampliamente utilizado para describir este tipo de desarrollos hace aproximadamente una década y a día de hoy

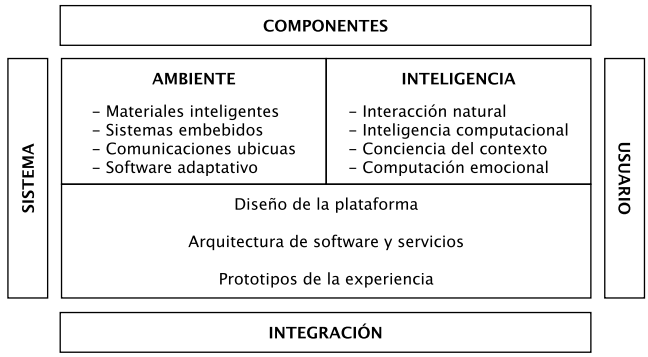



Figura 3.1: Requisitos de investigación en IAm.

se ha aceptado como un término para referirse a un área multidisciplinar que abarca una gran variedad de campos de las ciencias de la computación. Todas estas áreas de las que se alimenta son relevantes pero ninguna de ellas cubre conceptualmente por completo los requisitos que plantea la IAm. Es la IAm la que pone a colaborar todos estos recursos para proporcionar al usuario servicios flexibles e inteligentes dentro de su propio entorno. Por Inteligencia Ambiental nos referimos, por tanto, a los mecanismos que gobiernan el comportamiento del entorno, siendo sensible a las demandas del usuario y aprendiendo o conociendo sus preferencias para poder reaccionar de forma personalizada y consciente del contexto.

Más concretamente, según [Aarts et al., 2001], las principales características que debe tener un sistema de Inteligencia Ambiental son:

- **Discreción**: Los dispositivos deben integrarse de forma transparente en el entorno, siendo invisibles tanto física como psicológicamente.
- **Personalización**: Deben reaccionar de forma acorde con la situación y perfil de cada individuo.
- **Adaptabilidad**: Deben adaptarse a los cambios que sucedan en las personas y su entorno.
- **Proactividad**: Deben prever la mayor cantidad de procesos posibles en el entorno, anticipando las necesidades de las personas.

Para alcanzar esta ambiciosa visión de la IAm es necesario lograr un progreso significativo en diferentes tecnologías y áreas de conocimiento. Así, el Grupo Asesor de la Sociedad de las Tecnologías de la Información (ISTAG) de la Unión Europea plantea una especificación formal de los requisitos fundamentales de investigación en IAm [Ducatel et al., 2001] (ver Figura 3.1). Estos requisitos pueden dividirse en cuatro dominios, separados en dos ejes: en el horizontal se contrapone el usuario y el sistema (cada tecnología se acerca a uno de estos extremos), mientras en el eje vertical se enfrentan los componentes y la integración. Dentro de estos cuatro

extremos se ubican aquellas tecnologías en las que debe promoverse la investigación, las cuales se dividen en tres grupos de acuerdo al dominio al que estén más próximas.

Un entorno de IAm no podría ser parte de la cotidianidad de las personas si no cuenta con características que permitan su aceptación social [Ducatel et al., 2001], como son: facilitar el contacto humano, ayudar a generar conocimiento y habilidades laborales, inspirar confianza, ser consistente en todos los niveles y, una de las más importantes, que pueda ser utilizado por personas ordinarias sin un conocimiento especial en el ámbito tecnológico.

La IAm se asienta en las mismas bases que la Computación Ubicua y la Computación Pervasiva. No obstante, según la definición del ISTAG [Ducatel et al., 2003], la diferencia fundamental está en que estas últimas se orientan más hacia aspectos tecnológicos relacionados con la integración transparente de la tecnología en el entorno, mientras que la IAm da mayor relevancia al usuario y a la interacción natural y humanizada entre usuario y tecnología.

Todas estas características nos presentan a la Inteligencia Ambiental como un paradigma innovador y enormemente ambicioso desde su concepción teórica. La realidad actual nos muestra que sus objetivos son difíciles de conseguir en la práctica, y que necesitamos nuevos enfoques para acometer esta formidable tarea. No obstante, a lo largo de la última década se han realizado algunos progresos significativos como se refleja en la literatura [Aarts and Diederiks, 2007] [Aarts and Encarnaçao, 2006] [Augusto and Shapiro, 2007] [Preuveneers and Novais, 2012]. El desarrollo de estos sistemas y entornos nos muestra que el área de AmI permanece todavía en su infancia, como se aprecia al observar la enorme distancia existente entre la visión conceptual y el grado de concreción de los objetivos teóricos que actualmente se ha alcanzado. De hecho, uno de los problemas reconocidos que condicionan el desarrollo del área a día de hoy es la ausencia de modelos y metodologías de ingeniería del software que nos ayuden a analizar, diseñar, verificar y probar este tipo de sistemas [Coronato and De Pietro, 2010] [Penserini et al., 2010] [Becker, 2008] [Goertzel and Wang, 2006].

A pesar de las dificultades, en los últimos años han nacido numerosos proyectos que, buscando una aplicación práctica de las ideas propuestas, han llevado su aplicación a entornos tan diversos como casas inteligentes [Abowd and Mynatt, 2004] [Izso, 2009], monitorización de la salud y asistencia médica [Larson, 2007], educación [Shi et al., 2003] [Abowd, 1999], entornos de trabajo [Adler and Davis, 2004] [Stanford, 2005] o entornos que extraigan y exploten conocimiento a cerca de las actividades y las necesidad de las personas para su propio beneficio [Bosse et al., 2013]. Es importante señalar que ninguna de estas aplicaciones incorpora todas las características y requisitos planteados en los ambiciosos fundamentos teóricos de la IAm. No obstante, quizás lo más interesante de estas aplicaciones es que

pueden aportar experiencia para la creación de futuras soluciones que se acerquen cada vez más a la visión dada por la IAm.

El trabajo de esta tesis se guiará en gran medida por los requisitos generales que se han planteado para la IAm, apoyándose en la experiencia de los trabajos previos y defendiendo que la enorme complejidad inherente a estos sistemas nos obliga a abandonar la idea clásica de construir sistemas monolíticos no interoperables sustentados en una tecnología concreta. Al contrario, a día de hoy parece claro que el camino más razonable para avanzar de manera global hacia la visión proporcionada por la IAm es la construcción de modelos y arquitecturas generales que abstraigan en gran medida la complejidad inherente a estos entornos, proporcionando un soporte común para muchos de los requisitos generales de cualquier sistema de este tipo.

### 3.1.2. Desafíos y perspectivas de investigación

Algo más de una década de investigación en el área de la IAm nos ha dejado nuevos puntos de vista y puntos de acuerdo acerca de la forma en que los entornos de IAm deben ser diseñados y construidos para cumplir el requisito de ser realmente no intrusivos y adaptativos desde la perspectiva del usuario. En este sentido, recientemente se ha comenzado a aceptar la importancia de tener en cuenta elementos relacionados con la inteligencia social y el diseño para la realización de la visión aportada por la IAm. En esta sección se presenta un resumen de los principales aspectos de investigación y desafíos que permanecen abiertos en el área.

Si recordamos la visión dada por la IAm, el término Inteligencia se refiere a la respuesta que los usuarios esperan del sistema en términos de proactividad, predictivilidad y adaptatividad en su comportamiento. Por otra parte, el término Ambiental está relacionado con factores humanos y con la ubicuidad de un sistema no invasivo. De hecho, el principal objetivo debe ser el cumplimiento de las expectativas del usuario y la adaptación a sus necesidades, preferencias y hábitos.

Diversos estudios previos han tratado de definir las características comunes que deben presentar los entornos y sistemas para cumplir con la visión dada por la IAm. De acuerdo con [Saha and Mukherjee, 2003], escalabilidad, heterogeneidad, integración, invisibilidad, sensibilidad al contexto y gestión del contexto son todos los desafíos que deben abordarse. El artículo de Guruduth Banavar y sus compañeros de IBM imagina un proceso de desarrollo de aplicaciones independientes de la complejidad del acceso a los dispositivos, con un sistema dinámico de carga de componentes que incluye descubrimiento, negociación y selección dinámica de interfaces de usuario [Banavar et al., 2000]. Eila Niemela y Juhani Latvakoski proponen las siguientes características: interoperabilidad, heterogeneidad, movilidad, seguridad, adaptativilidad, auto-organización y realidad aumentada [Niemelä and Latvakos-

ki, 2004]. Los investigadores Roy Want y Tervor Pering de Intel proponen gestión energética, descubrimiento de componentes, adaptación de interfaces de usuarios y computación sensible a la localización [Want and Pering, 2005]. En [Da Costa et al., 2008] se propone el conjunto de características siguiente: heterogeneidad, escalabilidad, confianza/seguridad, privacidad, interoperación espontánea, mobilidad, sensibilidad al contexto, gestión del contexto, interacción transparente con el usuario e invisibilidad. Emili Aarts y Boris De Ruyter destacan la necesidad de incorporar características relacionadas con la inteligencia social [Aarts and Ruyter, 2009] (e.j., empatía, consciencia, adaptación a las convenciones sociales.).

A continuación se presenta una revisión en mayor detalle de los principales requisitos arquitectónicos y de diseño que por consenso podemos establecer para la construcción de sistemas de IAm. Además, en la Tabla 3.1 se presenta una visión resumida de los principales aspectos involucrados en cada requisito.

**Heterogeneidad.** Se refiere a la necesidad de que el software oculte las diferencias de infraestructura a los usuarios realizando las adaptaciones necesarias entre distintos entornos. Con el objetivo de facilitar la interconexión entre sistemas heterogéneos se deben utilizar estándares abiertos, publicar interfaces de acceso y facilitar la extensión de los sistemas. Para abordar la heterogeneidad inherente a estos entornos es necesario el desarrollo de middleware que independice la programación de las aplicaciones de variaciones en el entorno con el que se relacionan.

**Escalabilidad.** Los sistemas de IAm involucran a una gran variedad de elementos como usuarios, dispositivos, aplicaciones y redes de comunicaciones, presentes en una escala a menudo impredecible. Por tanto, en un contexto de esta naturaleza se debe evitar la centralización de las soluciones, fomentar la modularidad y prevenir los posibles cuellos de botella del sistema. Además, los componentes software deben ser instalados, cargados y gestionados automáticamente.

**Sensibilidad al contexto.** En los principios de la Computación Ubicua se establece la necesidad de que los sistemas utilicen la información relevante y los servicios del entorno. El contexto define todos esos elementos que deben ser observados para poder modelar el estado actual del entorno (situación) [Dey, 2001]. Las aplicaciones sensibles al contexto deben tener la capacidad de adaptar su comportamiento al contexto sin la intervención directa de los usuarios, con el objetivo de incrementar su usabilidad y mejorar la experiencia del usuario. A pesar de los avances en este campo, actualmente quedan numerosas barreras por superar tanto en la fase de adquisición de la información relevante del entorno como en las fases de representación e integración de dicha información [Baldauf and Dustdar, 2007] [Reichle et al., 2008].

**Interacción transparente y natural con el usuario.** A medida que los computadores se hacen más “inteligentes”, se hace obligado un incremento en la intensidad y calidad de la interacción humanizada [Saha and Mukherjee, 2003]. Una

visión ambiciosa de la interacción con el usuario debe centrarse tanto en los dispositivos de interacción como en el diseño de la interacción física en sí misma [Hornecker, 2005]. El primer desafío consiste en crear interfaces fácilmente integrables en los objetos del mundo real tomando en consideración los factores sociales y personales que pueden condicionar la experiencia de usuario. Por otra parte, durante la fase de diseño de los sistemas debemos crear interfaces de usuario abstractas y predecir los distintos tipos de interacción de modo que la decisión de qué interfaz debe ser utilizada se posponga al tiempo de ejecución.

**Movilidad.** Se trata de una característica clave para los objetivos de la Computación Ubicua, y se refiere a la capacidad del sistema para trasladar la funcionalidad de las aplicaciones de un dispositivo a otro siguiendo al usuario [Augustin et al., 2002]. Es necesario enfrentarse a una problemática de gran complejidad para proporcionar a los usuarios una experiencia de movilidad satisfactoria, lo que hace que permanezca como un tema abierto en el que únicamente se han logrado superar los primeros hitos [Adelstein, 2004] [Pan et al., 2010]. Algunos de los aspectos clave son: la interoperabilidad entre sistemas, la movilidad física de componentes, la adaptación de los componentes migrados al nuevo entorno y la privacidad.

**Invisibilidad.** Esta característica también forma parte de los aspectos centrales de la Computación Ubicua. Se refiere al objetivo de mantener la atención del usuario en la tarea que está realizando en vez de en las herramientas [Weiser, 1991], es decir, el usuario pasa a ser el recurso más importante del sistema [Garlan et al., 2002]. El primer paso para alcanzar la invisibilidad es el diseño de aplicaciones adaptables que requieran de la mínima intervención por parte del usuario [Saha and Mukherjee, 2003].

**Interoperabilidad.** Un componente de un sistema es interoperable si dispone de interfaces que le permiten trabajar con otros componentes y sistemas sin ninguna restricción de acceso o implementación [Niemelä and Latvakoski, 2004]. Podemos distinguir entre dos niveles de interoperabilidad: sintáctica y semántica. Dos o más sistemas exhiben interoperabilidad sintáctica si son capaces de comunicarse e intercambiar datos. La interoperabilidad semántica es la habilidad para interpretar la información intercambiada con el objetivo de producir los resultados esperados por ambos sistemas. Alcanzar ambos niveles de interoperabilidad entre sistemas desarrollados independientemente es uno de los objetivos de muchos investigadores y desarrolladores de diferentes áreas. Quizá los esfuerzos recientes más destacables son los que provienen de la Web Semántica y los Servicios Web [Booth et al., 2004] [Martin et al., 2007] [Dutta, 2008]. No obstante, a pesar de los abundantes desarrollos en esta área, el objetivo final de la interoperabilidad entre sistemas computacionales complejos está lejos de ser alcanzado [Bennaceur et al., 2010b]. Esto es así debido principalmente a la extrema heterogeneidad de tecnologías existentes y a que la comunicación espontánea tampoco es un tema resuelto.

**Interoperación espontánea.** Decimos que la interacción entre componentes es espontánea cuando los participantes en esta colaboración no están prefijados de antemano e incluso no es necesario que se conozcan previamente [Kindberg and Fox, 2002]. La interoperación espontánea es necesaria por la naturaleza volátil de los entornos de IAm, cuyos componentes interaccionan con distintos conjuntos de servicios según las necesidades de cada momento. Para diseñar un sistema que cumpla con este requisito es necesario gestionar de forma automática y uniforme el proceso de búsqueda y selección de los componentes participantes en una interacción.

**Integridad y seguridad.** En un ámbito tan complejo como el de la IAm resulta de vital importancia minimizar la necesidad de mantenimiento del sistema así como preservar la integridad del mismo. En términos de seguridad, es necesario garantizar la confidencialidad pero también la disponibilidad e integridad globales del sistema. Además, en este tipo de entornos a menudo resulta extremadamente difícil predecir y anticipar todas las posibles situaciones de fallo. En estos casos el sistema debe ser capaz de detectar los fallos y adaptarse a la situación manteniendo, aunque sea de forma limitada, la prestación de servicios al usuario. En la literatura se identifican, además, algunas de las consideraciones especiales que deben tenerse en cuenta para diseñar un sistema confiable y seguro en un entorno de IAm, como la sensibilidad al contexto o el uso de soluciones no intrusivas para el usuario [Dourish et al., 2004] [Robinson et al., 2005].

**Privacidad y confianza.** Ambos términos están íntimamente relacionados con aspectos de seguridad. Por la propia naturaleza de estos sistemas, de modo general, en cualquier entorno de IAm se maneja información sensible de los usuarios a través de distintas redes de comunicación y variedad de dispositivos. Debemos por un lado lidiar con los aspectos legales derivados del tratamiento de esta información y por otro lado asegurar que el sistema minimice el riesgo de exposición no deseada de la información sensible de los usuarios. En este tipo de entornos tan heterogéneos y dinámicos puede resultar muy complejo establecer a priori permisos y grados de confianza estáticos, al contrario, los niveles de confianza y privacidad pueden evolucionar con el entorno en función del contexto, de las interacciones que tienen lugar o de la localización de los usuarios [Cahill et al., 2003].

**Inteligencia social.** Distintos autores han puesto de relieve la necesidad de complementar la inteligencia de los entornos de IAm con inteligencia social. Para ello se plantea la necesidad de introducir en los entornos de IAm elementos que los conviertan en socializados, empáticos y conscientes [Aarts et al., 2007] [Aarts and Ruyter, 2009]. Socializados porque la interacción con el usuario se adecua a las reglas y convenciones sociales comúnmente aceptadas. Empáticos porque el entorno es consciente en algún grado del estado emocional del usuario y se adapta a él. Conscientes porque el sistema mantiene un estado interno suficientemente rico como para exhibir un comportamiento que pueda ser reconocido como consciente de la situación en sus interacciones con el usuario. Otro punto clave es el grado de

| Requisito | Aspectos clave |
|---|---|
| Heterogeneidad | Independecia del entorno físico.<br>Interconexión de sistemas heterogéneos. |
| Escalabilidad | Evitar la centralización.<br>Fomentar la modularidad.<br>Despliegue y gestión automática de componentes. |
| Sensibilidad al contexto | Adquisición e integración de la información del entorno.<br>Modelado de la situación.<br>Adaptación automática del comportamiento a la situación. |
| Interacción natural | Dispositivos fácilmente integrables en el entorno.<br>Generación dinámica de interfaces.<br>Interacción multimodal. |
| Movilidad | Acceso a aplicaciones y datos en cualquier momento y lugar.<br>Adaptación de los componentes migrados al nuevo entorno.<br>Seguridad y privacidad. |
| Invisibilidad | Mantener atención usuario en tarea y no en herramientas.<br>Diseño de aplicaciones adaptables.<br>Mínima intervención del usuario. |
| Interoperación expontánea | Asociación e interacción dinámica entre componentes.<br>Adaptación a los componentes disponibles en el entorno. |
| Interoperabilidad | Interoperoperabilidad sintáctica y semántica.<br>Comunicación espontánea. |
| Integridad y seguridad | Tolerancia a fallos y disponibilidad del sistema.<br>Confidencialidad y seguridad sensibles al contexto.<br>Mecanismos no intrusivos escalables a dispositivos heterogéneos. |
| Privacidad y confianza | Autenticación, encriptación y permisos.<br>Mecanismos adaptables al entorno y al contexto.<br>Aspectos legales y sociales. |
| Inteligencia social | Sistemas socializados, empáticos y conscientes.<br>Aceptación de los sistemas por parte de los usuarios. |

Tabla 3.1: Requisitos para la construcción de sistemas de IAm.

aceptación de este tipo de sistemas, dónde los usuarios podrían sentirse excesivamente monitorizados y controlados [Wright et al., 2008].

El análisis de este conjunto de requisitos pone todavía más de manifiesto, si cabe, la enorme complejidad de construir sistemas en entornos tan vastos y dinámicos como los definidos por los principios de a IAm y la computación ubicua. Como se ha visto, inmediatamente se abre un gran conjunto de desafíos tecnológicos comunes a prácticamente cualquier sistema de este tipo que se desee desarrollar:

- ¿Cómo se estructuran y organizan todos los componentes del sistema?
- ¿Dónde deberían estar localizados estos componentes?
- ¿Cómo interaccionan estos componentes entre sí?
- ¿Cómo se incorporan los nuevos componentes software/hardware al sistema

y como se organizan para integrar su funcionamiento en el conjunto?

- ¿Por qué canales fluye la información y qué garantías de seguridad y confianza ofrecen?
- ¿Cómo se pueden aprovechar, en determinado contexto, todas las fuentes de información y dispositivos existentes en el entorno?

La idea que subyace como respuesta a todas estas preguntas es el concepto de una arquitectura software y de una serie de modelos conceptuales que deberían proporcionar un soporte homogéneo a todos los componentes de un entorno de IAm y que deberían permitir el desarrollo integrado de nuevas funcionalidades.

Una arquitectura software puede proporcionar middleware capaz de ocultar la complejidad del entorno aislando a las aplicaciones de la gestión explícita de protocolos, fallos de comunicación, replicación de información, etc. El middleware también pude resolver los problemas de heterogeneidad relacionados con sistemas operativos, tecnologías de red o lenguajes de programación. Por tanto, una arquitectura proporciona un marco conceptual y una serie de herramientas para los diseñadores y desarrolladores que agilizan el desarrollo, despliegue, reutilización e interoperabilidad de las soluciones de IAm. Según esto, la aproximación de utilizar una arquitectura software común parece una buena forma de abordar la problemática descrita hasta ahora. No obstante, no existe un consenso real en la literatura respecto a este punto. Muchos investigadores ponen el foco en el desarrollo de funcionalidades de IAm concretas dejando de un lado los aspectos de infraestructura hardware/software [Brdiczka et al., 2009] [Stoettinger, 2004] [Rakotonirainy and Tay, 2004]. Así, en muchos casos, se ignoran los aspectos relacionados con el acceso a los dispositivos hardware, con la complejidad de las redes de comunicación y con la posibilidad de interoperación con componentes de terceros.

A pesar de las distintas aproximaciones que se pueden encontrar en la literatura a la hora de abordar la tarea de construir sistemas de IAm, parece claro que es necesario concebir arquitecturas software y modelos conceptuales que permitan abordar la problemática de forma global y metodológica. Siguiendo esta línea de pensamiento, numerosos investigadores han abierto líneas de trabajo conducentes al diseño de arquitecturas de IAm que proporcionen soporte para al menos un subconjunto de los requisitos comunes de la IAm. Es el caso del proyecto Amigo, que propone una combinación de tecnologías derivadas fundamentalmente de la Computación Orientada a Servicios [Janse and Vink, 2008]. AIT (Athens Information Technology) propone una plataforma basada en tecnologías de sistemas multi-agente a través del proyecto CHIL [Soldatos et al., 2007]. Siguiendo una aproximación diferente, el proyecto SOPRANO [Sixsmith et al., 2009] y el proyecto PERSONA [FP6-IST PERSONA, 2010] proponen plataformas orientadas a entornos de asistencia a personas mayores basadas en la plataforma OSGI [OSGi Alliance, 2015].

Las aproximaciones existentes basadas en arquitecturas y middleware son, a día

de hoy, incapaces de cubrir todas las características y requisitos presentados aquí. De hecho algunos autores afirman que el middleware está en crisis, siendo incapaz de cumplir algunas de sus principales promesas, como es la capacidad de ofrecer interoperabilidad real [Bennaceur et al., 2010b]. Podemos decir que la investigación y desarrollo de este tipo de sistemas se encuentra todavía en su infancia, no obstante algunas de los trabajos existentes suponen unos primeros pasos interesantes en el camino hacia la consecución de los ambiciosos objetivos de la IAm.

El conjunto de desafíos y necesidades que hemos analizado en este apartado y que todavía permanecen abiertos en mayor o menor medida, constituyen la motivación para el trabajo realizado en esta tesis, centrado en el avance en la construcción de modelos conceptuales y middleware de referencia que permitan dar respuesta a los requisitos comunes de un sistema de IAm desde una aproximación integrada. Para centrar aún más el contexto de este trabajo, en la sección 3.4 se incluye una discusión detallada sobre algunos de los trabajos previos que pueden considerarse como las iniciativas más relevantes dentro de esta línea de investigación.

## 3.2. Paradigmas de ingeniería del software más relevantes para IAm

En la sección anterior se identificaron una serie de propiedades y requisitos clave a los que debería dar respuesta una plataforma integral para el desarrollo de sistemas de IAm. En esta sección se hace una revisión de los paradigmas de ingeniería del software existentes que comparten, en su visión, algunas de las características deseadas para los entornos de IAm.

El nivel creciente de complejidad en los sistemas software en general ha estimulado el nacimiento de nuevos paradigmas de ingeniería del software. Ninguno de estos nuevos enfoques es lo suficientemente ambicioso como para lidiar con la complejidad propia de los entornos visionados por los principios de la Computación Ubicua o de la Inteligencia Ambiental. No obstante, existen dos paradigmas, la Computación Orientada a Servicios y los Sistemas Multi-Agente, en los que se apoyan muchas de las soluciones que se han desarrollado en los últimos años dentro de estos campos. En los siguientes puntos de esta sección se describirán los fundamentos de ambos paradigmas, con el fin de poner de relevancia sus bondades y limitaciones a la hora de ser aplicados en la construcción de entornos de IAm.

### 3.2.1. Computación Orientada a Servicios

La Computación Orientada a Servicios (SOC, del inglés Service Oriented Computing) es hoy en día un paradigma de referencia ampliamente aceptado para la

construcción de sistemas software distribuidos. De acuerdo con este paradigma, los componentes funcionales se abstraen como servicios con bajo acoplamiento, cuya lógica interna está oculta detrás de una interfaz de acceso al componente [Papazoglou and Georgakopoulos, 2003]. Ésto permite crear una red de servicios en base a la que construir procesos dinámicos, utilizando el patrón de interacción orientado a servicios. Siguiendo este patrón, el proveedor de servicios y el que demanda un servicio pueden ser sistemas no diseñados para trabajar juntos, sin embargo, el consumidor de servicios puede acceder a un registro a través del que descubrir aquellos componentes que necesita para realizar su tarea.

La promesa última de la Computación Orientada a Servicios es un mundo de servicios colaborativos, donde los componentes de las aplicaciones se ensamblan fácilmente obteniendo una red de servicios que interaccionan para crear nuevos procesos de forma flexible y dinámica. Idealmente, estos servicios son entidades computacionales autónomas que pueden ser utilizados con independencia de la plataforma subyacente. Así, los servicios pueden ser sintáctica y semánticamente descritos, publicados, descubiertos y dinámicamente interconectados, permitiendo desarrollar sistemas masivamente distribuidos e interoperables.

Una de las claves para alcanzar estos objetivos es el concepto de Arquitectura Orientada a Servicios (SOA, del inglés Service Oriented Architecture) [Arsanjani et al., 2009]. SOA puede ser vista como un método para el diseño y desarrollo de sistemas computacionales que proporciona una modelo bien definido para la utilización de servicios como soporte para los requisitos de negocio. La utilización de un Middleware Orientado a Servicios (SOM, del inglés Service Oriented Middleware) debe ayudar a crear sistemas más flexibles y dinámicos, fomentando la reutilización de las servicios existentes.

El campo de la Computación Orientada a Servicios es basto y complejo, involucrando numerosos conceptos y tecnologías que tienen sus orígenes en distintas disciplinas como sistemas distribuidos, redes de computadores, ingeniería del software, seguridad o Inteligencia Artificial. Como resultado, se puede afirmar que se trata de uno de los campos de investigación más activos hoy en día en el área de las tecnologías de la información, caracterizado por su naturaleza multidimensional, y en el que las actividades de investigación están muy fragmentadas. En los siguientes puntos se estudiará el estado actual y los principales desafíos de investigación en los aspectos SOC más relevantes para IAm.

#### Descripción de servicios

La descripción de servicios es un elemento fundamental en SOC, y determina la información que un servicio necesita exponer al entorno para posibilitar su identificación y uso. De forma general, los principios de diseño de una arquitectu-

ra SOA [Arsanjani et al., 2009] hablan de que una descripción de servicios podría contener información de los siguientes tipos:

- Las capacidades proporcionadas por el servicio y, opcionalmente, las capacidades requeridas de otros servicios.
- Conversaciones que permitan modelar la interacción entre servicios.
- Propiedades no-funcionales, referentes a la calidad de servicio por ejemplo.
- Información asociada a los servicios necesaria para invocación de los mismos, como el protocolo de acceso al servicio, formatos de mensaje o información a nivel de transporte y direccionamiento.

De acuerdo con esto, se han propuesto diferentes lenguajes para soportar la descripción de servicios, algunos de los cuáles han alcanzado el estatus de estándar. La mayoría de iniciativas se centran en el concepto de Servicios Web [W3C, 2015b], convirtiéndose en la tecnología dominante para SOC. Adoptan un lenguaje de descripción basado en XML (WSDL, del inglés Web Service Description Language) junto con protocolos comunes (e.j., SOAP sobre HTTP), incluyendo únicamente una descripción sintáctica de los servicios y utilizando un modelo de interacción de tipo petición/respuesta [Booth et al., 2004]. Las organizaciones W3C y OASIS son las principales encargadas de la estandarización en esta área.

Paralelamente a los Servicios Web, la iniciativa de la Web Semántica [W3C, 2015a] junto con otras tecnologías relacionadas han producido un cambio significativo en la forma en que los servicios son descritos a través de la introducción de elementos semánticos. Emplear únicamente descripciones sintácticas obliga a un acuerdo previo a nivel sintáctico entre los proveedores y consumidores de servicios, a la vez que hace ambigua la semántica de los servicios. La semántica de los servicios se hace explícita a través de referencias a un vocabulario estructurado de conceptos (ontología) que representa un dominio concreto de conocimiento. El lenguaje OWL (del inglés Web Ontology Language) es el estándar establecido por el W3C para la descripción de ontologías [W3C, 2015c].

Si consideramos la descripción semántica de servicios o de sistemas completos (vistos como composiciones de servicios), las aproximaciones OWL-S [W3C, 2004], WSMO [Roman et al., 2006] y SAWSDL [W3C, 2007] son la más destacadas [Dutta, 2008] [Lanthaler et al., 2010].

El modelo propuesto por OWL-S se basa en gran medida en la idea de que un servicio exporta una funcionalidad descrita con un perfil compuesto por un conjunto de entradas, salidas, precondiciones y postcondiciones. La semántica de cada uno de estos elementos es definida con referencias a conceptos que pueden estar definidos en ontologías. Otro elemento fundamental de la descripción del servicio es la interfaz de acceso al mismo. OWL-S propone por defecto el uso de WSDL como lenguaje para la interacción con el servicio, aunque deja abierta la posibilidad de

que se puedan hacer extensiones del modelo para utilizar otras tecnologías.

SAWSDL es la recomendación del W3C para añadir anotaciones semánticas al lenguaje WSDL, que únicamente posee expresividad a nivel sintáctico. Dichas anotaciones pueden ser expresadas en cualquier lenguaje de ontologías y deben añadirse a las operaciones y a los parámetros de entrada/salida del esquema XML correspondiente a un descriptor WSDL.

WSMO (del inglés Web Service Modeling Ontology) es un modelo para los aspectos más relevantes de los servicios semánticos. Propone una descripción de las capacidades de los servicios en un estilo más funcional que OWL-S o SAWSDL, incluyendo la especificación de las capacidades en función de objetivos del cliente del servicio en vez de en función de entradas y salidas. Al contrario que las anteriores aproximaciones, contempla de forma explícita la inclusión de propiedades no-funcionales en la descripción de los servicios. Otra diferencia con OWL-S es que contempla que un servicio pueda proporcionar más de una funcionalidad. En conjunto, se puede decir que propone un enfoque más ambicioso que el resto de aproximaciones, sin embargo su desarrollo parece haber sido abandonado y las herramientas existentes están en un estado de implementación incompleto.

En lo referente a las propiedades no-funcionales, como seguridad, privacidad, confianza, disponibilidad, calidad de servicio, que caracterizan los resultados que un servicio proporciona, son habitualmente ignoradas en la descripción de servicios. Además, existe menos consenso en la comunidad SOC en lo referente a como deben ser identificadas y especificadas. No obstante, existen esfuerzos, especialmente en lo relacionado con la calidad de servicio, para alcanzar modelos generales para la descripción de propiedades no-funcionales [OASIS, 2015].

La emergencia de estos lenguajes de descripción de servicios, definidos en parte para abordar el problema de la heterogeneidad sintáctica en la definición de servicios, contribuye a una mayor heterogeneidad del middleware existente [Nagarajan et al., 2006] [Baresi et al., 2012]. Estos lenguajes emplean diferentes terminologías y formalismos para especificar elementos similares de los servicios, lo que dificulta la integración de servicios, especificados con lenguajes distintos, para la realización de una tarea común. Además, estos lenguajes introducen ambigüedades en la descripción de servicios debido a que la forma en que se realizan las anotaciones semánticas también varía de unos lenguajes a otros. La interoperabilidad puede alcanzarse en cierta medida si se realiza la traducción de las descripciones heterogéneas de servicios a un lenguaje común. Esto permitiría realizar el emparejamiento y composición de servicios de forma independiente a los lenguajes subyacentes. En los últimos años, podemos encontrar algunos intentos de formalizar una aproximación de este tipo [Tokuda et al., 2006] [Mokhtar et al., 2006] [Bennaceur et al., 2010a]. No obstante, ninguna de estas iniciativas ha alcanzado un nivel de madurez suficiente, al contrario, la interoperabilidad continúa siendo uno de los principa-

les desafíos cuando es necesario conectar espontáneamente sistemas heterogéneos complejos [Bennaceur et al., 2010a] [Autili et al., 2014].

La heterogeneidad no proviene únicamente de los modelos y lenguajes de descripción de servicios, la propia diversidad de lenguajes de ontologías y la falta de ontologías globales ampliamente aceptadas conlleva un problema reconocido de heterogeneidad a nivel de ontologías [Qu et al., 2006] [Avilés-López and García-Macías, 2009]. Por tanto, el problema de heterogeneidad que las ontologías pretendían resolver se ha trasladado a un nivel superior, el de las propias ontologías.

**Descubrimiento de servicios**

La publicación y descubrimiento de descripciones de servicios son dos de las funciones principales del patrón de interacción orientado a servicios. La aproximación más extendida para la organización y gestión de descripciones en registros es la propuesta en la especificación UDDI (del inglés Universal Description Discovery and Integration) [UDDI, 2006]. UDDI propone un sencillo modelo para la descripción de los servicios desde el punto de vista de los servicios de negocio ofertados por una empresa y desde un punto de vista técnico con referencias a la descripción WSDL del mismo. Una de las líneas de investigación en esta área ha sido el intento de enriquecer el sencillo modelo de UDDI con metadatos semánticos, con el fin de posibilitar búsquedas de servicios más ricas [Sivashanmugam et al., 2003] [Luo et al., 2006]. Desafortunadamente, actualmente las soluciones más extendidas de repositorios de servicios están prácticamente muertas y no parece existir actualmente un interés importante en continuar investigando sobre ellas.

En el ámbito del descubrimiento de servicios nos encontramos con un problema derivado de la potencial heterogeneidad de las descripciones. Aunque se trata de un problema que continua abierto, existe alguna propuesta de modelos de servicios comunes a los que mapear descripciones provenientes de tecnologías heterogéneas [Ben Mokhtar et al., 2010] [Georgantas et al., 2010].

En lo relacionado con los protocolos de descubrimiento de servicios, surge un nuevo problema de heterogeneidad de middleware. Lidiar con esta heterogeneidad implica un acuerdo en el protocolo a través del cuál los servicios se publicitan y se descubren. En este sentido, existen en la bibliografía algunas aproximaciones al problema que proponen adaptaciones desde algunos protocolos concretos de origen a un único protocolo común [Ben Mokhtar et al., 2010] [Raverdy et al., 2006] [Grace et al., 2003]. Por otro lado, la aproximación clásica de SOA es utilizar un único repositorio de servicios centralizado. No obstante, se trata de una solución que escala mal con respecto al creciente número de servicios y clientes demandados por áreas incipientes como la Computación Móvil o la IoT. Para abordar esta situación diversas aproximaciones proponen diseños de registros descentralizados en base a

la federación de registros, tanto en arquitecturas planas como jerárquicas [Chakraborty et al., 2006] [Gao et al., 2006] [Kassim et al., 2007].

Otro aspecto a considerar es el problema de emparejar las descripciones de servicios almacenadas con las peticiones (denominado habitualmente como proceso de *matching*). El problema principal aquí es que las descripciones en lenguajes existentes, como OWL-S o SAWSDL, cubren únicamente una parte de los aspectos necesarios para un razonamiento automático efectivo. En concreto, las propiedades no-funcionales de los servicios son comúnmente ignoradas.

**Interacción con los servicios**

A partir de una búsqueda de servicios por parte de un cliente y la consiguiente localización de algún servicio que encaje con la petición, y basándose en la información proporcionada por el servicio en su descripción, el acceso al servicio posibilita la interacción entre el cliente y el servicio. Teniendo en cuenta que, los servicios pueden ser implementados de formas distintas sin colaboración entre los desarrolladores, la uniformidad de las tecnologías y el middleware empleado no puede ser asumida en general. Esta diversidad de tecnologías es especialmente característica de los nuevos contextos relacionados con Computación Móvil e IoT, y propone nuevos desafíos para las tecnologías clásicas de SOC [Baresi et al., 2012]. Actualmente, los servicios, claramente dominados por las tecnologías de Servicios Web, utilizan comúnmente un modelo de interacción de tipo petición/respuesta soportado por WSDL. Como ya se mencionó, incluso tecnologías de servicios semánticos como OWL-S descargan en la práctica su modelo de interacción en WSDL.

Existen, sin embargo, otros modelos de interacción entre sistemas de información, como modelos basados en mensajes, modelos basados en eventos o modelos basados en memoria compartida. En primer lugar, si se persigue la generalidad de las arquitecturas SOA no se puede asumir la existencia de un único modelo. En segundo lugar, utilizar como base para la interacción un único modelo de tipo petición/respuesta limita de forma drástica la riqueza de la interacción entre servicios. No obstante, algunas aproximaciones intentan combinar diversos modelos de interacción utilizando abstracciones comunes y un sistema de mapeo a protocolos concretos [Autili et al., 2014] [Bencomo et al., 2013] [Avilés-López and García-Macías, 2009]. El paradigma ESB (del inglés Enterprise Service Bus) es la solución actualmente dominante para establecer puentes entre sistemas heterogéneos, utilizando los adaptadores necesarios [Schmidt et al., 2005]. Aún así, la mayoría de estas soluciones de interoperatividad son desplegadas de forma estática.

Aunque no existe una solución global al problema de la interoperabilidad entre modelos de interacción en entornos dinámicos, algunas propuestas recientes tratan de avanzar en este sentido [Autili et al., 2014] [Grace et al., 2011].

### Composición de servicios

En el ámbito SOC, la línea de investigación que se refiere a la composición de servicios trata de aportar soluciones para la agregación de múltiples servicios de modo que colaboren como un servicio compuesto en la resolución de una tarea. Se puede decir que se trata de uno de los aspectos claves dentro de este paradigma, dado que el potencial real de un desarrollo orientado a servicios solo se obtiene cuando las aplicaciones y procesos de negocio son capaces de cooperar para ofrecer nuevos servicios con valor añadido [Papazoglou et al., 2008].

En la última década, se ha trabajado en la definición de lenguajes basados en XML para la especificación de procesos de composición de servicios, siendo BPEL [Jordan and Alves, 2007] el más representativo en la línea de orquestación de servicios y WS-CDL [Kavantzas, 2005] el más relevante de los orientados a coreografía de servicios. No obstante, actualmente el consenso es que esta distinción entre orquestación y coreografía es difusa y debe confluir en un único lenguaje que impulse la automatización del proceso de composición dinámica de servicios [Papazoglou et al., 2008]. De hecho, uno de los mayores desafíos a superar para que las aproximaciones orientadas a servicios gocen de mayor aceptación es precisamente el desarrollo de tecnologías y métodos que soporten una composición de procesos distribuidos que sea realmente flexible, dinámica, eficiente y sencilla de utilizar [Papazoglou et al., 2008].

Actualmente las actividades de investigación en este campo se centran principalmente en mejorar la especificación de servicios de forma que las composiciones puedan ser evaluadas y verificadas automáticamente [Lohmann, 2010] y en el dinamismo y adaptación de las composiciones [Chan and Lyu, 2008] [Silva et al., 2008]. En el área de la Inteligencia Artificial existen también trabajos orientados a aplicar técnicas de planificación para automatizar la composición de servicios [Lazovik et al., 2006] [Pistore et al., 2005]. No obstante, en general, estos esfuerzos se encuentran todavía en estados preliminares de desarrollo, y únicamente algunas aproximaciones recientes comienzan a tener en cuenta la importancia de desarrollar aproximaciones que tengan en cuenta información relacionada con el contexto en que se realiza la composición [Ben Mokhtar et al., 2007] [Yu and Su, 2009] [Zhou and Riekki, 2010] [Bronsted et al., 2010].

### Discusión

Hemos visto que la visión propuesta por las Arquitecturas Orientadas a Servicios proporciona una serie de principios de diseño y herramientas que facilitan la construcción de sistemas distribuidos en base a servicios flexibles y desacoplados. Estas características redundan en un incremento en la capacidad de adaptación de los sistemas construidos con el paradigma SOC a los cambios que se producen en

su entorno de ejecución. Por tanto, desde sus principios, este paradigma se erige en una prometedora aproximación tecnológica para la construcción de sistemas como los que demandan los requisitos de IAm [Di Nitto et al., 2008] [Quitadamo et al., 2007].

A lo largo de la revisión hemos visto un paradigma bien asentado en la visión y en las tecnologías actuales de Internet. Sin embargo, también hemos visto un paradigma que, a pesar de los avances provenientes principalmente de las tecnologías de los Servicios Web y de la Web Semántica, se enfrenta a numerosos desafíos. Además, algunos de los desarrollos más ambiciosos en el área de los Servicios Semánticos, como WSMO, OWL-S, no disponen de implementaciones completas, se encuentran descontinuados o han dado lugar a implementaciones poco portables a los nuevos entornos de Computación Móvil e IoT. Asimismo, existen muchas otras propuestas teóricas que no se han materializado en una implementación de referencia bien documentada, mantenida y accesible que pueda ser utilizada por la comunidad de desarrolladores. Por todo ello, la ansiada interoperabilidad real entre servicios desarrollados independientemente continúa siendo una promesa más que una realidad.

En paralelo con la evolución del paradigma SOC, áreas como la IAm, la Computación Ubicua, la Computación Móvil y el IoT apuntan hacia una evolución de los sistemas de información y comunicaciones que hace énfasis en desafíos ya conocidos, relacionados con la heterogeneidad, la escalabilidad, la movilidad o la adaptación al contexto. De hecho, el Internet de los Servicios y el Internet de las Cosas comienzan a ser una realidad con la proliferación de servicios y dispositivos heterogéneos interconectados a través de múltiples redes de comunicación y protocolos. Por tanto, para facilitar a los desarrolladores la construcción de aplicaciones que hagan uso de esta abundancia de servicios, propios de lo que será el Internet del Futuro [FutureInternet, 2015], es necesario revisitar las soluciones de Middleware Orientado a Servicios desarrolladas para el Internet de hoy.

Por encima de todo, el middleware debe buscar un equilibro entre la creciente cantidad y heterogeneidad de información expuesta a través de servicios y la complejidad de procesar esa información para posibilitar la publicación, descubrimiento, adaptación, composición y acceso a los servicios, dentro de un ecosistema de dispositivos y tecnologías diversas. Muy particularmente, el paradigma debe abordar un problema generalizado de heterogeneidad de middleware no interoperable, que se extiende a muchos de los elementos principales de la computación SOC: ontologías, modelos de descripción de servicios, modelos de interacción, registros de servicios y protocolos de descubrimiento de servicios. En todos estos elementos el middleware que pretendía ocultar un problema de heterogeneidad de tecnologías ha trasladado el problema a un nivel superior.

En el caso concreto de la IAm, las aplicaciones deben adaptar su comportamien-

to de forma automática para responder a entornos que evolucionan y a cambios en su contexto. Para que esta adaptación sea dinámica es necesario soportar la composición de servicios que pueden tener diferencias en la forma en que su semántica es expuesta o en la forma en que son accedidos. Igualmente, es necesario expandir el ámbito de ejecución de servicios a virtualmente cualquier entorno computacional. La IAm demanda, por tanto, ese tránsito desde servicios web con modos de interacción estáticos y capacidades descriptivas insuficientes hacia servicios ubicuos y adaptables. Podríamos resumirlo en que el desafío es realizar una evolución desde las arquitecturas SOA actuales hacia arquitecturas SOA ubicuas.

Por tanto, podemos concluir que existe todavía un salto tecnológico importante entre el estado de desarrollo actual del paradigma de la Computación Orientada a Servicios, particularmente en lo referente al Middleware Orientado a Servicios existente, y los ambiciosos requisitos demandados por los principios de la IAm y por la evolución hacia el Internet del Futuro.

### 3.2.2. Sistemas Multi-Agente

Los primeros trabajos en Sistemas Multi-Agente surgen hacia finales de la década de los 70 y especialmente en los 80, fruto de trabajos en Inteligencia Artificial y resolución distribuida de problemas (DPS), que perseguían la cooperación de múltiples sistemas simples para resolver problemas complejos.

En los años 90 se pasa a una etapa de consolidación. Por un lado se asientan los conceptos teóricos de la mano de autores como Wooldridge, Etzioni, Nwana o Brene, mientras por otro lado surgen numerosos desarrollos que sirven como pruebas de concepto y a partir de los cuales se desarrollan metodologías y estándares. Además, el auge de Internet hacia finales de los 90 introduce numerosas áreas de aplicación para sistemas cooperantes. Búsqueda e integración de información o comercio electrónico se postulan como áreas perfectas para la aplicación práctica de este tipo de soluciones.

Los numerosos desarrollos llevados a cabo de forma independiente coinciden en la necesidad de estandarizar e integrar las tecnologías, y sobre todo de acercarlas al campo de la Ingeniería del Software para facilitar su aplicación a entornos comerciales y empresariales. Así a principios de los 2000 comienza el desarrollo de metodologías, modelos y estándares relacionados con la ingeniería del software de sistemas multi-agente.

#### Definición de Agentes Inteligentes y Sistemas Multi-Agente

De las múltiples y muy diversas definiciones para el concepto de Agente Inteligente que se pueden encontrar en la bibliografía una de las más citadas es la de

Wooldridge [Wooldridge, 2002], que dice:

> "Un agente es un sistema informático situado en un entorno y que es capaz de realizar acciones de forma autónoma para conseguir sus objetivos de diseño".

Para describir de forma más clara lo que es un agente inteligente se puede enumerar un conjunto de propiedades que deben exhibir [Wooldridge, 2002]:

- **Autonomía.** Un agente debe ser capaz de decidir qué acciones ejecutar o qué objetivos elegir sin un control exterior explícito.
- **Reactividad.** Un agente ha de ser capaz de reaccionar de forma instantánea ante los cambios que tengan lugar en su entorno.
- **Persistencia.** Los agentes se mantienen en el tiempo, a diferencia de otro tipo de programas.
- **Comunicación.** Un agente puede interactuar y comunicarse con otros agentes.
- **Adaptatividad.** Un agente tiene capacidad de aprendizaje y, de esta forma, es capaz de cambiar su comportamiento según su experiencia y adaptarse a un entorno que puede ser dinámico y sufrir cambios bruscos.
- **Proactividad.** Los cambios de un agente no son dirigidos únicamente por los cambios en el entorno. Un agente ha de ser capaz de cambiar en función de sus propios objetivos y tomar la iniciativa en un momento dado.
- **Movilidad.** Un agente tiene capacidad para moverse en el entorno.

Un sistema multi-agente no es más que un sistema computacionalmente complejo compuesto por una serie de agentes que interactúan entre sí. Los agentes que forman un sistema multi-agente son autónomos, independientes y generalmente heterogéneos. Surgen por la necesidad de construir sistemas que permitan la gestión inteligente de un sistema complejo coordinando los distintos subsistemas que lo componen. Para que la interacción se realice con éxito los agentes que componen el sistema han de ser capaces de cooperar, coordinar, negociar y resolver conflictos.

- **Cooperación.** Proceso mediante el cual los agentes actúan de forma conjunta para conseguir un fin común.
- **Coordinación.** Es necesaria cuando crece el número de agentes. La comunicación es la que permite a los agentes coordinar sus acciones y comportamientos, de esta forma el sistema resulta más coherente.
- **Resolución de conflictos.** Aparecen conflictos cuando en el problema que se va a resolver ocurre alguna de las siguientes circunstancias: el conocimiento local es incorrecto o incompleto; coexisten metas divergentes; hay distintos criterios de evaluación de las soluciones.

- **Negociación.** Proceso de comunicación entre los agentes para conseguir llegar a una solución aceptada por todos cuando aparecen conflictos o problemas en la cooperación.

Para que se manifiesten las características anteriores es imprescindible establecer modos y mecanismos de comunicación entre los agentes. En las primeras etapas de esta tecnología cada sistema desarrollado hacía uso de sus propias soluciones de comunicación. Como cada agente había sido desarrollado de forma independiente, surgía el problema de la falta de entendimiento entre unos y otros. Para solucionar este problema surgieron una serie de iniciativas que tratan de especificar una conjunto de estándares que garanticen una interacción real y homogénea entre sistemas multi-agente. Un ejemplo de esto es el estándar FIPA, que se describe en el punto siguiente.

#### Estándar FIPA

La fundación FIPA (del inglés, Foundation for Intelligent Physical Agents) [FIPA, 2015] es un asociación internacional sin ánimo de lucro de compañías y organizaciones que comparten el esfuerzo de crear unas especificaciones genéricas para las tecnologías de agentes.

FIPA establece que sólo debe ser especificado el comportamiento externo de los componentes del sistema, dejando los detalles de implementación y las arquitecturas internas a los desarrolladores de agentes. Así, especifica las reglas que permiten a una sociedad de agentes interoperar y proporciona un modelo de referencia para una plataforma de agentes. Principalmente identifica los papeles de algunos agentes clave necesarios para la gestión de la plataforma. El AMS (Agent Management System), o sistema de gestión de agentes, que se refiere al agente que ejerce el control de supervisión sobre el acceso y uso de la plataforma; es el responsable de la autenticación de los agentes residentes y del control de los registros. El ACC (Agent Communication Channel) es el encargado de proporcionar el camino para el contacto básico entre agentes dentro y fuera de la plataforma; es el método de comunicación por defecto que ofrece una rutina de servicio de mensajes fiable, ordenada y precisa. El DF (Directory Facilitator) es el agente que proporciona un servicio de páginas amarillas a la plataforma.

Como es lógico, el estándar también especifica el lenguaje de comunicación entre agentes ACL (Agent Communication Language) basado en el intercambio de mensajes. FIPA ACL especifica un lenguaje estándar de mensajes definiendo la codificación y la semántica de éstos. El estándar no establece un mecanismo para el transporte interno de mensajes, sin embargo, como distintos agentes pueden estar ejecutándose en distintas plataformas con distintas tecnologías de interconexión, FIPA especifica que los mensajes deben ser transportados entre las plataformas co-

dificados en forma de texto. La sintaxis de ACL es muy parecida a la del lenguaje KQML (Knowledge Query Manipulation Language) [Finin et al., 1994].

Este estándar proporciona formas comunes de conversaciones entre agentes a través de la especificación de protocolos de interacción, que son patrones de los mensajes intercambiados entre dos o más agentes. Otros aspectos especificados son la integración agente-software, la movilidad, la seguridad y la comunicación humano-agente.

**Discusión**

Los agentes tienen características como autonomía, razonamiento, proactividad y movilidad que los hacen apropiados para el desarrollo de sistemas dinámicos y distribuidos en entornos de IAm, que demandan capacidades de adaptación a los usuarios y a las características de entornos cambiantes [Penserini et al., 2010]. Recordemos que la idea clave de la IAm es "desarrollar tecnologías que se adapten a las personas", como opuesto a "personas que se adaptan a la tecnología".

En el contexto de la construcción de entornos inteligentes de IAm, se pueden adoptar distintas perspectivas a la hora de trasladar el concepto de agente a dichos entornos. Por un lado, tenemos que los sistemas complejos de un entorno de IAm requieren de un elevado número de componentes software. Cada uno de estos componentes, cuando involucran cierta autonomía e inteligencia, puede ser visto como un agente inteligente que contribuye a la arquitectura software del entorno. Por otro lado, los propios usuarios del entorno pueden ser vistos como agentes inteligentes. Entonces, el software necesita ser diseñado para razonar acerca de los usuarios e interaccionar con ellos, en el marco de un sistema multi-agente.

Está comúnmente aceptada la idea de que la característica de sensibilidad al contexto es uno de los elementos importantes para dotar de inteligencia a los sistemas de IAm [Coutaz et al., 2005] [Reichle et al., 2008]. En este sentido, las tecnologías de agentes son una herramienta relevante para el análisis y obtención de datos desde una red de sensores distribuidos, y para dotar a esos sensores de la habilidad de trabajar juntos y analizar situaciones contextuales complejas [Padovitz et al., 2008] [Ranganathan and Campbell, 2003].

Así, numerosos autores identifican a los sistemas multi-agente como una de las alternativas de arquitecturas distribuidas más relevantes, y asimismo como un paradigma útil para ayudar a describir y desarrollar sistemas de IAm distribuidos, dinámicos, escalables y ubicuos, tanto desde el punto de vista del software como del hardware [Cook, 2009] [Corchado et al., 2008] [Rui et al., 2009].

Desafortunadamente, la dificultad de desarrollar un sistema multi-agente es en general alta. Debido a la falta de herramientas de desarrollo especializadas para el

paradigma, los desarrolladores de agentes se enfrentan a tareas de diseño y programación complejas y poco estructuradas. Además, aunque los sistemas multi-agente presentan capacidades adecuadas para soportar la modularidad y la reutilización de sistemas, la integración es difícilmente alcanzada debido a la incompatibilidad entre las diferentes plataformas de agentes. Relacionado con esta última problemática, hay autores que han explorado formas de abordar la integración e interoperabilidad de sistemas multi-agente utilizando tecnologías provenientes del paradigma SOC [Tapia et al., 2009a] [Karaenke et al., 2012].

En la sección 3.4 se podrá comprobar como algunos de los desarrollos de middleware para IAm más relevantes se apoyan en conceptos y tecnologías de sistemas multi-agente.

## 3.3. Internet de las Cosas

La primera mención conocida a Internet de las Cosas (IoT, del inglés Internet of Things) se remonta a una presentación por parte de Kevin Ashton, integrante del MIT, en 1999. En ella, visionó que la incorporación de etiquetas RFID en los objetos cotidianos crearía un Internet de las Cosas [Ashton, 2009]. En su visión original, estas etiquetas RFID se relacionarían con bases de datos accesibles en Internet que contendrían información sobre los objetos etiquetados. La visión actual del Internet de las Cosas se refiere a la integración de un gran cantidad de objetos físicos en Internet, con el objetivo de posibilitar un alto nivel de interacción con el mundo físico que nos rodea. De hecho, a día de hoy, este tipo de dispositivos están teniendo una importante penetración en nuestras vidas, a través de la presencia creciente de sensores y actuadores en nuestros teléfonos móviles, coches, casas e incluso en forma de dispositivos *wearables*.

Tras un análisis de la bibliografía, en [Miorandi et al., 2012] se concluye que el IoT actual se compone de tres capas principales: una red de objetos inteligentes conectados a través de tecnologías que extienden el actual Internet; una capa física que soporta múltiples tecnologías de dispositivos; y una serie de aplicaciones que aprovechan esas tecnologías. Además, los autores identifican un grupo de características que un objeto inteligente del IoT debe poseer: ser un objeto físico, tener capacidades de comunicación, tener un nombre y dirección entendible por una máquina, tener un identificador único, tener capacidades de computación, y tener capacidades de sensorización o actuación.

Por tanto, el área de IoT está directamente relacionada con la visión de la IAm en cuanto al objetivo de integrar los entornos físicos en el contexto del usuario, a través de monitorizar sus estados y actuar sobre ellos con el fin de adaptarlos dinámicamente a las necesidades y preferencias de las personas. Para conseguirlo, al

igual que la IAm, el IoT debe lidiar con una creciente proliferación de tecnologías heterogéneas y no interoperables, a lo que hay que incorporar la dimensión añadida por el enorme crecimiento de la Computación Móvil. Las Cosas se convierten en Cosas móviles con capacidades para comunicarse inalámbricamente y ser conscientes de sus cambios de localización, además de incorporar cada vez un mayor número de sensores. Así, los desarrolladores también deben enfrentarse a los desafíos que acarrea el traslado de paradigmas y patrones de operación en entornos con conectividad y operación permanente a entornos de computación móvil. En los siguientes subapartados se hace una introducción a las principales aportaciones del área de la IoT en este tipo de aspectos.

### 3.3.1. Abstracción de tecnologías

El problema de la heterogeneidad puede ser visto desde diferentes niveles:

- Heterogeneidad de hardware de sensorización/actuación de diferentes fabricantes, de diferentes generaciones y con diferentes propiedades.
- Heterogeneidad de redes y protocolos de sensorización/actuación a través de las que se accede a los dispositivos.
- Heterogeneidad de recursos de computación, APIs y plataformas de desarrollo disponibles en diferentes dispositivos.
- Heterogeneidad en el formato y tipo de datos que producen los sensores/actuadores.

Uno de los tipos de aproximaciones que nos encontramos en la bibliografía para abordar esta heterogeneidad son aquellas que se centran en proporcionar una descripción semántica de los dispositivos, sus funcionalidades y de los tipos de datos manejados por sus sensores/actuadores. Así, un grupo importante de las ontologías para IoT derivan de esfuerzos en el área de las Redes de Sensorización Inalámbricas, siendo la contribución más influyente proveniente de esta área la ontología SSN [Compton et al., 2012], que proporciona una ontología para redes de sensores sobre la que se puede razonar.

Recientemente se han hecho nuevos esfuerzos para extender las ontologías de sensores con información semántica que las acerque más al concepto de dispositivo propio de IoT. Por ejemplo, Sense2Web [De et al., 2012] proporciona una ontología que modela dispositivos, las funcionalidades que éstos proporcionan y el servicio a través del que un recurso puede ser accedido. Otras aproximaciones se orientan más a la descripción de entornos físicos concretos junto con los dispositivos que contienen. Es el caso de la ontología DogOnt [Bonino and Corno, 2008] [Bonino and Corno, 2010], que proporciona un modelo orientado a la descripción de espacios residenciales, incluyendo tanto elementos arquitectónicos pasivos como dispositivos

controlables.

Más allá de las ontologías que proporcionan una taxonomía de dispositivos, describiendo sus propiedades y capacidades, recientemente se están realizando numerosos esfuerzos para homogeneizar el acceso a dispositivos desde un nivel de abstracción más elevado, principalmente a través de servicios. Las aproximaciones de este tipo existentes pueden clasificarse en dos grandes grupos: i) aquellas que utilizan un Servicio Web descrito en WSDL para encapsular el acceso a las funcionalidades de un dispositivo, típicamente accesible utilizando el protocolo HTTP y el envío de datos en formato XML; ii) aquellas que utilizan Servicios REST [Richardson and Ruby, 2007]. Estas últimas proporcionan un menor grado de interoperabilidad al no utilizar un modelo estándar para especificar la interfaz del servicio, sin embargo permiten construir soluciones más ligeras, aptas para dispositivos con menores capacidades computacionales.

Utilizando ambas tecnologías de servicios, en primer lugar existen aproximaciones que proponen abstraer directamente las funcionalidades que proporciona un dispositivo como un servicio [Guinard et al., 2011]. En segundo lugar, otras aproximaciones proponen una virtualización de los dispositivos, de modo que los servicios acceden a una representación virtual del dispositivo que almacena los datos del dispositivo real que previamente han sido procesados [Aberer et al., 2007].

Incluso teniendo los dispositivos uniformemente representados, y servicios creados a través de los que acceder a los dispositivos, sigue pendiente de resolver la heterogeneidad a nivel de red en el descubrimiento y acceso a los servicios. Ésta problemática puede darse tanto en un entorno de red local (redes como Bluetooth, ZigBee, UPnP, WiFi, etc.) como en el caso de interacciones remotas a través de Internet (servicios SOAP, servicios REST, 6LowPAN, etc.). En ambos casos, una posible aproximación al problema sería incorporar al middleware de servicios plugins para cada tecnología de red que se quiera abstraer.

### 3.3.2. Escalabilidad

Viendo el IoT desde una perspectiva global, existe otro aspecto muy relevante a tener en cuenta que puede transformar en totalmente insatisfactoria cualquier de las aproximaciones anteriores: la escalabilidad de la solución. En la visión de IoT de un mundo de objetos interconectados, el número de dispositivos puede crecer de forma vertiginosa, y el middleware de IoT debe estar preparado para obtener, procesar y almacenar enormes cantidades de datos en tiempo real. Este contexto puede invalidar una aproximación orientada a servicios con acceso directo a los dispositivos por problemas de rendimiento.

Se impone así, el estudio de soluciones para la agregación de información proveniente de múltiples sensores potencialmente redundantes [Mietz et al., 2013] [Rana

et al., 2010] [Schmidt et al., 2011], para el pre-procesado distribuido de la información [Dean and Ghemawat, 2008], o incluso soluciones provenientes del almacenamiento *en la nube* tanto para almacenar la información proveniente de los sensores como los metadatos de los dispositivos [Mitton et al., 2012] [Mell and Grance, 2011].

### 3.3.3. Adaptación a entornos móviles

A día de hoy, el IoT asume, de forma general, que se trabaja con un entorno físico controlado o con una movilidad de dispositivos limitada y predeterminada. No obstante, la tendencia será hacia que la movilidad sea la regla, y cada vez más dispositivos estén vinculados a los patrones de movilidad de las personas [Teixeira et al., 2013].

En concreto, la sensorización móvil, puede dividirse en dos categorías [Lane et al., 2008]: sensorización parcitipativa y sensorización oportunística. En el primer caso se involucra directamente a la persona en el control del dispositivo, mientras que en la sensorización oportunística es el dispositivo móvil el que determina cuándo y dónde debe realizar su tarea.

En general, un esquema de sensorización oportunística demanda que los dispositivos estén localizados permanentemente para que el sistema pueda tomar la iniciativa sin intervención de las personas [Lu et al., 2010] [Guo et al., 2013]. La aproximación habitual en este tipo de escenario es que los dispositivos registren la información actualizada de su localización en un repositorio, generalmente con almacenamiento en la nube.

Estos escenarios relacionados con la IoT móvil están poco explotados más allá de soluciones ad-hoc para problemas concretos. En este contexto, asunciones como la existencia de una conectividad permanente, el acceso en tiempo real a la información o el conocimiento previo de las capacidades de interacción con el entorno disponibles, dejan de ser válidas.

### 3.3.4. Discusión

Como hemos visto, los desafíos principales del IoT se centran lidiar con la naturaleza heterogénea de dispositivos y redes de dispositivos, y construir una solución escalable que pueda manejar eficientemente un entorno dinámico poblado por un gran número de dispositivos. No obstante, de esta revisión se concluye que en la actualidad las soluciones disponibles no proporcionan una aproximación integral a todas las problemáticas planteadas.

Con el fin de manejar la heterogeneidad de tecnologías de dispositivos se han

creado numerosas ontologías para modelar dispositivos. Asimismo, se han explotado tecnologías provenientes de la Computación Orientada a Servicios para abstraer el acceso a los dispositivos en base a servicios. Así, soluciones existentes basadas en Servicios REST proporcionan un sistema ligero y adecuado para un gran número de dispositivos, pero que presenta deficiencias en aspectos como la interoperabilidad o el descubrimiento automático de servicios.

Centrándonos específicamente en los requisitos de la IAm, y más concretamente en las necesidades de interacción de los sistemas de IAm con el entorno físico, el campo de la IoT junto con las Redes de Sensorización Inalámbrica y en general las tecnologías de abstracción del acceso a dispositivos de sensorización/actuación, contribuyen de manera directa con soluciones para este requisito. Sin embargo, la ausencia de soluciones de referencia de IoT hace que cualquier sistema de carácter general en el ámbito de IAm deba lidiar con un problema de heterogeneidad de los modelos de abstracción de dispositivos y de heterogeneidad del middleware de IoT.

Asimismo, no parece conveniente que una aplicación de IAm tenga que lidiar con aspectos particulares derivados de la movilidad de los dispositivos o los procesos de transformación y agregación de información proveniente del entorno. Por tanto, en la actualidad la IAm demanda un nivel mayor de abstracción que facilite la integración de diferentes tecnologías de dispositivos en general, y de IoT en particular.

## 3.4. Arquitecturas software para IAm

Como se ha puesto de relevancia a lo largo de este capítulo, los sistemas de Inteligencia Ambiental incorporan una serie de nuevos desafíos al desarrollo de software, como movilidad, adaptabilidad, heterogeneidad o interoperabilidad. Muchas de estos aspectos son comunes y por tanto deberían ser gestionados por una infraestructura de IAm común en vez de por cada aplicación de forma individual. En esta tesis se defiende este tipo de aproximación, basada en la definición de un vocabulario, conceptos y abstracciones comunes como el primer paso hacia la construcción de sistemas interoperables, reutilizables, adaptables y más fáciles de diseñar. Sin embargo, como en el caso de otros paradigmas de ingeniería del software, un modelo conceptual para sistemas de IAm no facilita por sí mismo la tarea de los desarrolladores de aplicaciones. Con el fin de agilizar la construcción de sistemas de IAm, los desarrolladores necesitan, además, middleware que de soporte a los modelos conceptuales, y que les proporcione una serie de herramientas y utilidades que les permitan centrarse en la lógica de las funcionalidades deseadas en vez de en la interacción e integración de las diferentes tecnologías subyacentes.

No obstante, dado que la IAm es un campo relativamente reciente (comienza a

popularizarse hace poco más de una década), la mayoría de los proyectos, especialmente los primeros, dirigen sus objetivos hacia la obtención de funcionalidades para dominios concretos, inclinándose por un diseño ad-hoc y por implementaciones destinadas a un determinado entorno muy restringido que reducen drásticamente las posibilidades de adaptar esos proyectos a otros entornos, objetivos o tecnologías. Algunos ejemplos de este tipo de aproximación pueden encontrarse en [Bobick et al., 1999] [Krumm et al., 2000] [Stoettinger, 2004] [Tapia et al., 2004] [Rakotonirainy and Tay, 2004] [Brdiczka et al., 2009] [Bernardin et al., 2009].

Frente a esto, han surgido otros proyectos motivados por el objetivo de construir soluciones middleware de referencia que faciliten el desarrollo de sistemas de IAm en entornos reales. Ejemplos destacados son proyectos como Oxygen [OXYGEN, 2005], centrado en entornos de trabajo; CHIL [Waibel et al., 2004], que proporciona una infraestructura para la integración de tecnologías perceptivas, interfaces multi-modales y servicios para asistir a las personas en entornos de trabajo fundamentalmente; AMIGO [FP6-IST Amigo, 2008a], que proporciona una arquitectura versátil con importantes avances en aspectos relativos a la interoperabilidad entre tecnologías; PERSONA [FP6-IST PERSONA, 2010], enfocado a desarrollar una arquitectura que facilite la construcción de entornos que incrementen la independencia de la gente mayor; SOPRANO [Sixsmith et al., 2009], que, centrándose en escenarios de cuidado de personas mayores, proporciona abstracciones de ese entorno y un sistema semántico basado en ontologías para la orquestación de la colaboración entre componentes.

Como parte del trabajo de esta tesis se han realizado estudios sucesivos de un gran números de proyectos que, de una forma más o menos rigurosa, siguen este último enfoque. En una última iteración, se han filtrado aquellos proyectos comparativamente menos relevantes para obtener la lista final de trabajos que son considerados más significativos de acuerdo con los objetivos que persigue esta tesis. En lo que sigue de esta sección, se realizará un recorrido en orden cronológico por esta lista de proyectos, intentando poner de relevancia tanto sus virtudes como sus debilidades en cuanto al objetivo de construir infraestructuras software de referencia en el campo de la IAm. Cabe mencionar que en el caso del proyecto Oxygen su relevancia ha de situarse en el contexto de la fecha en que se inició su desarrollo, ya que constituye uno de los primeros esfuerzos encaminados a definir modelos conceptuales de referencia para entornos de IAm.

### 3.4.1. Oxygen

El proyecto Oxygen [OXYGEN, 2005] fue desarrollado por el MIT (Massachussets Institute of Technology) entre los años 1999 y 2005. Sus objetivos se enfocaban al desarrollo de sistemas pervasivos y centrados en el usuario a través de la com-

binación de diversas tecnologías y áreas de conocimiento. Así, el proyecto en su conjunto debe ser entendido más como un cúmulo de ideas y conceptos que contribuyen a la computación pervasiva y no intrusiva, que como un único sistema software. No obstante, entre los resultados de Oxygen están una serie de frameworks reutilizables como Metaglue [Coen et al., 1999], GOALS [Saif et al., 2003] e Hyperglue [Peters et al., 2003].

La infraestructura de soluciones propuesta por Oxygen gira en torno al desarrollo de tres grandes tipos de tecnologías: dispositivos centrados en el usuario, redes de comunicación flexibles y sistemas software adaptables:

- Tecnologías de dispositivos centrados en el usuario, que permiten crear espacios inteligentes en los entornos en que desenvuelven su actividad las personas, como oficinas, hogares o vehículos.
  - Dispositivos de sensorización y actuación integrados en el entorno físico. Utilizan fundamentalmente tecnologías en audio y vídeo, y tienen la capacidad de reconfigurarse para soportar múltiples protocolos de comunicación.
  - Dispositivos portátiles, encargados de procesar y comunicar información referente al usuario y a su entorno dentro de un espacio inteligente controlado por dispositivos de sensorización y actuación.
- Tecnologías de red flexibles y descentralizadas. Proporcionan descubrimiento automático de recursos, y proporcionan acceso seguro a esos mismos recursos disponibles en la red.
- Tecnologías software que se adaptan a los cambios de requisitos del entorno o de los usuarios. En particular, el desarrollo de arquitecturas software que proporcionan mecanismos para construir aplicaciones en base a componentes distribuidos y reutilizables.

En lo referente a la interacción con el usuario, el proyecto Oxygen busca métodos de interacción natural que sustituyan a teclados y ratones, centrándose principalmente en sistemas de voz y visión. Asimismo, se estudian formas de integración multimodal que incrementen la efectividad de estas tecnologías perceptuales.

Con el fin de transformar la visión de la computación centrada en las personas en una realidad, como parte del proyecto Oxygen se desarrollaron distintos sistemas prototipo en cada una de estas líneas principales. Nos centramos aquí en los sistemas Metaglue, GOALS e Hyperglue, que proporcionan conjuntos de frameworks para facilitar el desarrollo de aplicaciones en entornos de computación pervasiva.

El framework Metaglue [Coen et al., 1999] proporciona una plataforma de agentes software implementada en Java. La plataforma se basa en un modelo de comunicación RPC, utilizando Java RMI para las llamadas remotas entre agentes. Además, la arquitectura Metaglue también incluye un sistema para la gestión de

recursos y un lenguaje que facilita la programación de agentes utilizando una serie de conceptos de alto nivel que posteriormente son traducidos automáticamente al código Java necesario para implementar dichos agentes. El sistema de gestión de recursos permite a los agentes buscar en la red los agentes que mejor cumplen con una interfaz que se especifica en una petición de descubrimiento de agentes. Este mecanismo favorece la construcción de agentes desacoplados.

Extendiendo el framework Metaglue, el middleware GOALS [Saif et al., 2003] proporciona soporte para introducir ciertas capacidades de sensibilidad al contexto en la resolución de tareas dentro de la plataforma de agentes Metaglue. Con este objetivo, GOALS proporciona un lenguaje para la especificación de planes compuestos por una serie de nodos o procesos por los que fluye la ejecución de la tarea, en función de información contextual que define la situación actual del usuario y el entorno.

Posteriormente se desarrolló el framework Hyperglue [Peters et al., 2003] [Peters, 2006], con el objetivo principal de proporcionar un nuevo modelo de comunicación y descubrimiento de agentes que permita coordinar las acciones de los agentes a más alto nivel. Así, en Hyperglue, se desarrolla el concepto de sociedad de agentes añadiendo un agente interlocutor de cada sociedad. Este interlocutor es el responsable de entender las peticiones de los agentes de su sociedad y buscar, a través de un sistema de directorio, los agentes interlocutores de otras sociedades con las que es necesario colaborar para resolver una tarea.

Asimismo, Hyperglue también propone el uso de un modelo de contexto muy acotado para facilitar la construcción de componentes que se adapten a las necesidades del usuario. Este modelo únicamente tiene en cuenta tres dimensiones: el momento del día, la localización del usuario y la tarea que se estaba realizando cuando se realiza una nueva petición de servicio.

Como parte del proyecto Oxygen se realizaron importantes esfuerzos conducentes al desarrollo de prototipos de sistemas pervasivos que permitan la construcción de diversos espacios inteligentes, en especial en entornos de trabajo colaborativo como salas de reuniones y habitaciones inteligentes [AIRE, 2003]. No obstante, como en el caso de otros proyectos de este tipo desarrollados en estos años y que supusieron los primeros pasos serios en la definición de arquitecturas de referencia para entornos de IAm, la arquitectura software de Oxygen adolece de falta de características que hoy se consideran fundamentales como interoperabilidad, servicios semánticos de alto nivel o interoperación espontánea de componentes.

### 3.4.2. CHIL

El proyecto CHIL (Computers in the Human Interaction Loop) [Waibel et al., 2004] tiene como principal objetivo el desarrollo e integración de tecnologías per-

ceptivas, interfaces multi-modales y servicios concretos para asistir, de forma no intrusiva, a las personas en actividades y en entornos interiores (e.j., en reuniones, presentaciones, oficinas). Se trata de un proyecto desarrollado entre los años 2004 y 2007 por un consorcio de 15 grupos de investigación, y financiado en parte por la Unión Europea.

Frente a la opción de desarrollar servicios ad-hoc para la resolución de problemas concretos relacionados con este objetivo, el proyecto CHIL aborda el diseño e implementación de una arquitectura software que proporcione métodos comunes de interacción con sensores y actuadores, integración de tecnologías heterogéneas, e integración y composición de servicios sensibles al contexto. Tanto el proyecto global como la arquitectura ponen especial énfasis en la integración de componentes perceptivos provenientes de una variedad de proveedores tecnológicos, así como en el descubrimiento y gestión de los mismos.

Sobre la arquitectura desarrollada se construyen una serie de prototipos de servicios orientados a la asistencia natural y no intrusiva a las personas en la realización de ciertas actividades. Para ello se aborda el desarrollo de una serie de componentes comunes que tratan de ofrecer soluciones humanizadas para procesos de identificación de personas, reconocimiento de acciones y actividades, o anticipación de las necesidades de los usuarios durante la realización de una tarea. De forma general, todos estos componentes se apoyan en una serie de interfaces multi-modales que permiten "observar" a los usuarios en su entorno a través del procesamiento de múltiples señales de audio y vídeo (reconocimiento del habla, reconocimiento de gestos, reconocimiento de caras, identificación de objetos, etc.).

#### Arquitectura software general

La arquitectura CHIL [Soldatos et al., 2006] [Soldatos et al., 2007] proporciona una serie de principios de diseño, especificaciones y APIs que gobiernan el desarrollo de servicios y aplicaciones en entornos heterogéneos y distribuidos. La construcción de esta arquitectura parte de un modelo conceptual que propone una división en tres capas:

- **Servicios de usuario.** Los componentes de este nivel son responsables de gestionar la interacción con el usuario y de proporcionar servicios concretos a los usuarios asegurando la reusabilidad y colaboración de los mismos.
- **Modelado de la situación.** En este nivel se procesa la información contextual recibida de los sensores para modelar la situación. Esta información contextual permite a los servicios de usuario responder de forma más adecuada ante variaciones en las actividades de los usuarios y antes cambios en el entorno.
- **Tecnologías perceptivas.** En este nivel se encuentran los componentes encargados de proporcionar una sensorización transparente del entorno. Como

ya se ha mencionado, CHIL se centra fundamentalmente en sensorización basada en audio y vídeo.

Esta arquitectura conceptual se concreta a través de una serie de decisiones tecnológicas. Se mantiene la división en tres capas principales, donde la capa superior se implementa utilizando tecnologías de sistemas multi-agente (en concreto se utiliza el framework de agentes JADE). La utilización de este framework facilita el desarrollo de servicios distribuidos, que se comunican entre sí utilizando un conjunto de mensajes especificado en el estándar FIPA.

En este diseño, la comunicación entre las tres capas se realiza a través de componentes desarrollados por terceros. Concretamente, los componentes perceptivos acceden a los sensores utilizando el middleware NIST Smart Flow [Rosenthal and Stanford, 2000], que proporciona un alto rendimiento en la transferencia distribuida de flujos de audiovisuales y facilita el procesamiento descentralizado de este tipo de flujos. La comunicación entre los componentes perceptivos y los servicios de usuario basados en agentes se implementa utilizando la librería CHILIX de IBM. Esta librería facilita el intercambio de información independiente de la plataforma, basado en XML sobre TCP.

Siguiendo esta organización, la información fluye a lo largo de la arquitectura de la siguiente forma:

- Los sensores visuales y acústicos capturan flujos de audio y video, que son transferidos a los componentes perceptivos a través del sistema NIST Smart Flow. Alternativamente, otros sensores que producen directamente información contextual (e.j., sensores de temperatura, sensores de movimiento) añaden esta información directamente al contexto.
- Los componentes perceptivos procesan los flujos de audio y vídeo, buscando, por ejemplo, identificar y localizar a personas y objetos del entorno.
- La capa de servicios mantiene un modelo contextual de alto nivel, que es construido en base a la información contextual recibida desde los sensores y desde los componentes perceptivos a través del sistema CHILIX.

En una arquitectura distribuida y basada en componentes como esta resulta también fundamental disponer de algún mecanismo para localizar a los componentes y servicios disponibles. En este sentido, CHIL descarta el uso de mecanismos y protocolos de directorio existentes (e.j., UPnP, SLP, UDDI) argumentando que no son apropiados para lidiar con la heterogeneidad y diversidad propia de los entornos de IAm. Estos protocolos proporcionan mecanismos para registrar y descubrir recursos y servicios en una red. No obstante, dichos protocolos están demasiado orientados a unos tipos de dispositivos y contenidos concretos, y no son adecuados para gestionar el amplio rango de componentes e información que pueden estar presentes en un entorno de IAm. Motivado por estas limitaciones, CHIL incluye un

mecanismo de directorio que se apoya en una base de conocimiento en la que se registran especificaciones de los componentes accesibles en el sistema. Para la especificación de estos componentes se utilizan ontologías descritas en lenguaje OWL. El acceso remoto a este directorio se realiza a través de protocolos RPC como SOAP o RMI.

#### Sensores y componentes perceptivos

En lo referente a la abstracción del acceso a dispositivos heterogéneos, la arquitectura CHIL proporciona un modelo basado en la definición de un conjunto de APIs genéricas. En este modelo, para cada clase o tipo de dispositivo se define una API que abstrae el control y acceso de todos los dispositivos de ese tipo con independencia del fabricante. Así, con la propia arquitectura se incluye también una serie de APIs que proporcionan un conjunto de operaciones comunes para el acceso a sensores de audio y vídeo (distintos tipos de cámaras y micrófonos principalmente). Además, la arquitectura utiliza un agente proxy para cada clase de dispositivo definida, de modo que éstos pueden ser registrados y descubiertos en el registro de agentes de la plataforma JADE.

La solución de CHIL para el acceso a dispositivos facilita el desarrollo de componentes que necesitan interaccionar con el hardware, ya que estos únicamente necesitan conocer una API para controlar todos los dispositivos de una misma clase. No obstante, esta solución se queda a medio camino de proporcionar un modelo completo de acceso uniforme a dispositivos, dado que se mantiene la heterogeneidad entre distintas clases de dispositivos.

#### Sensibilidad al contexto

Como ya se ha mencionado, tanto los componentes sensores como los componentes perceptivos pueden generar información contextual que alimenta la base de conocimiento de un sistema CHIL. No obstante, para construir aplicaciones complejas es probable que sea necesario realizar composiciones de distintos componentes perceptivos. Para conseguir esto, la arquitectura utiliza un modelo basado en la construcción de grafos de transición de estados. Estos modelos de situación se acompañan de una tabla de verdad, que permite determinar cuando una combinación concreta de valores perceptivos dispara una transición a otro estado dentro de la situación contextual actual. Esta aproximación se caracteriza por ser una solución rígida debido a que está basada en la definición de grafos de estados estáticos a la vez que introduce un grado de complejidad elevado en la definición y configuración de un sistema CHIL para un entorno concreto.

#### Servicios

Los servicios que proveen la funcionalidad de un sistema CHIL se implementan como agentes de la plataforma JADE. La arquitectura CHIL también soporta el registro y descubrimiento de servicios utilizando el mismo mecanismo de proxies que se describió para el caso de los componentes perceptivos. Así, cada servicio tiene un agente proxy que permite su registro en la base de conocimiento de acuerdo con una especificación del servicio descrita en lenguaje OWL. Esta solución no contempla la utilización de otros lenguajes y modelos de descripción de servicios existentes, por lo que la interoperabilidad del sistema es limitada.

Un tipo especial de servicio son los servicios de actuación, que se encargan de controlar la interacción entre el entorno y el usuario. A mayores, la arquitectura obliga a la implementación de unos agentes especiales que interceptan las peticiones de los servicios para actuar sobre el entorno y seleccionan el servicio de actuación concreto se va a utilizar en cada momento. Una característica que le resta potencia y flexibilidad a este mecanismo es que la lógica que gobierna la elección de un servicio de actuación concreto tiene que ser codificada de forma estática en un algoritmo del agente intermediario.

#### Tolerancia a fallos

La arquitectura CHIL incorpora mecanismos de recuperación ante fallos en la capa de agentes. Así, un agente especial del sistema monitoriza el estado del resto de agentes para detectar cuando alguno de ellos muere. Ante un fallo de este tipo, el sistema distingue entre dos metodologías de recuperación: recuperación sin estado y recuperación manteniendo el estado. La recuperación sin estado únicamente ejecuta de nuevo el agente en la plataforma. Por el contrario, la recuperación que mantiene el estado del agente recupera de una base de datos la versión serializada del agente que contiene su estado en un momento concreto. La arquitectura no incluye este tipo de mecanismo fuera de la capa de agentes.

Además, se utiliza el Directory Facilitator (DF) de JADE como punto de registro y consulta de todos los agentes del sistema. No obstante, cabe mencionar que la experiencia demuestra que la utilización del DF de JADE con un elevado número de agentes puede suponer una importante degradación en el rendimiento de la plataforma de agentes.

### 3.4.3. LAICA

El proyecto LAICA [Cabri et al., 2008] explora las posibilidades de aplicación de tecnologías de sistemas multi-agente en la creación de infraestructuras software

que permitan lidiar con la complejidad de los entornos de IAm. El proyecto explota las características de este paradigma para modelar los dispositivos del entorno y para la orquestación de los componentes que proporcionan la funcionalidad de alto nivel en un sistema de IAm. Se trata de un proyecto que involucró a varias universidades y empresas, y que se enfoca hacia un contexto urbano. De hecho, su principal aplicación es el control de los flujos de personas y vehículos a través de las calles, con el fin de monitorizar dichos flujos, detectar posibles situaciones de peligro y actuar ante ellas.

#### Arquitectura conceptual LAICA

Para lidiar con la complejidad del entorno al que se orienta este proyecto, LAICA propone una arquitectura basada en los siguientes componentes principales:

- **Agentes de sensorización.** Agentes encargados de gestionar la información obtenida de los dispositivos de tipo sensor.
- **Agentes de actuación.** Agentes que pueden realizar acciones sobre el entorno a través de dispositivos de tipo actuador.
- **Portal web.** Permite el control y configuración de todo el sistema a través de tecnologías Web.
- **Base de datos.** Toda la información relevante extraída por el sistema se almacena en una base de datos, con el fin de mantener un seguimiento de lo que pasa en el entorno.
- **Central operativa.** Es un componente centralizado que constituye el núcleo de la arquitectura. Es el elemento encargado de la gestión y monitorización de todos los demás componentes del sistema, incluyendo el registro de los componentes disponibles en cada momento.

Las función principal del middleware LAICA es gestionar y facilitar la ejecución coordinada de servicios y el intercambio de datos entre componentes del sistema. En concreto, soporta la localización de los servicios demandados en el sistema así como la adaptación de datos y comandos entre agentes y dispositivos heterogéneos.

El entorno al que se dirige el proyecto LAICA es complejo y extenso, e implica la presencia de una infraestructura de red que es pervasiva a lo largo de una ciudad y que permite la interacción entre componentes. No obstante, las interacciones entre componentes no pueden estar predefinidas debido al dinamismo y heterogeneidad del entorno. En este contexto, el middleware LAICA soporta estas interacciones facilitando la localización dinámica y transparente de componentes y realizando las traducciones necesarias de protocolos y formatos.

Por otra parte, LAICA trata de mantener una visión global de todo lo que está pasando en el sistema en cada momento, realizando un seguimiento de las interac-

ciones y los datos obtenidos del entorno. No obstante, debido a la extensa red de sensores que puede manejar un sistema de este tipo, el middleware de LAICA también debe encargarse de ejecutar filtros inteligentes sobre la información recibida con el fin de almacenar en la base de datos únicamente la información relevante.

**Realización concreta del arquitectura LAICA**

Desde un punto de vista global, la arquitectura LAICA pretende ser una plataforma proveedora de servicios distribuidos implementados utilizando el paradigma de Sistemas Multi-Agente. En un ecosistema como este, la función principal de la arquitectura es definir y gestionar un modelo común escalable para la interacción colaborativa entre los distintos tipos de componentes (agentes principalmente) existentes en el sistema.

El middleware LAICA puede aceptar dos tipos de información de entrada: comandos y datos. Los datos representan información proveniente de los agentes sensores y pueden ser almacenados en la base de datos o enviados como entrada para otros componentes, después de una posible transformación o filtrado. Los comandos son utilizados por los componentes para obtener un servicio concreto del sistema. El middleware proporciona interfaces distintas para procesar comandos y datos.

Los datos de entrada se pasan directamente a la lógica de conversión, que decide si los datos deben ser almacenados y si deben ser transformados de alguna forma. Esta lógica de conversión analiza a su vez si la información puede ser agregada con otros datos disponibles.

Los comandos recibidos en el sistema pueden llegar directamente al repositorio (lookup table), que contiene información sobre todos los componentes de un sistema LAICA concreto. Es decir, el repositorio conoce donde se encuentra un determinado servicio, y puede conocer también otro tipo de información relevante para la interacción, como por ejemplo el protocolo que es necesario adoptar para colaborar con un determinado agente. Con toda esta información, el repositorio puede elegir y buscar el agente que contiene el servicio adecuado para responder a un comando recibido. Además, el repositorio también ofrece funcionalidades para mantener información de las colaboraciones entre agentes que están teniendo lugar en un determinado momento dentro del sistema.

El núcleo del middleware LAICA es un componente centralizado que se encarga de la gestión global de todos los demás componentes de la arquitectura, con el fin de mantener un funcionamiento óptimo de todo el sistema. Para ello, monitoriza de forma continua el comportamiento de los componentes críticos del sistema y puede tomar decisiones respecto a su configuración y modo de operación para mantener la estabilidad del sistema. Por ejemplo, el núcleo puede controlar las interfaces de entrada y salida del middleware con el fin de aislar al sistema de agentes que están

funcionando incorrectamente. El rol de este elemento es el de un supervisor, y debe mantener un nivel de intrusismo adecuado para no comprometer el rendimiento del sistema con sus propias actuaciones.

El middleware también presenta una interfaz de salida, que es utilizada para informar de los resultados de la ejecución de un comando a otros componentes LAICA.

En lo referente al formato de los datos y los comandos, para ambos se utiliza una codificación basada en el lenguaje XML, pero con dos esquemas diferentes, dadas las distintas características de datos (en su mayor parte datos multimedia en el caso de LAICA) y comandos. La utilización de XML proporciona un alto grado de independencia entre los distintos componentes a la hora del intercambio de datos. No obstante, con el fin de generalizar el uso de esta arquitectura, sería interesante mejorar este esquema incluyendo un marco semántico común e interoperable sobre el que abstraer el acceso a cualquier tipo de dispositivo y red de sensorización (e.j., utilizando una ontología para la descripción de las operaciones, características y datos involucrados en una red de sensorización heterogénea).

Basándonos únicamente en el diseño descrito hasta ahora, LAICA presenta una limitación fundamental relacionada con la escalabilidad y tolerancia a fallos del sistema, debido a la presencia de una serie de componentes únicos y centralizados que son esenciales para el funcionamiento de todo el sistema. No obstante, para paliar este problema en cierta medida, LAICA proporciona soporte para la ejecución distribuida de varias instancias de su middleware en diferentes nodos computacionales.

Un caso especial es el del componente que actúa como repositorio del sistema (lookup table). Con el fin de que no se convierta en un punto crítico del sistema, LAICA propone la posibilidad de su distribución en diferentes nodos del sistema. En concreto, recomienda la implementación de un repositorio jerarquizado y distribuido con la posibilidad de mantener réplicas sincronizadas del mismo. No obstante, y a pesar de la importancia de este componente en el sistema, LAICA no provee ninguna implementación concreta del mismo con este tipo de características.

En su conjunto, la arquitectura software LAICA proporciona una plataforma de agentes y un modelo de colaboración entre agentes interesante, que está orientado fundamentalmente al control avanzado de un entorno de sensorización altamente dinámico y distribuido. Sin embargo, carece de un modelo avanzado para la creación de componentes funcionales que permita la interoperación espontánea entre los mismos y la interoperabilidad entre componentes desarrollados en distintas plataformas software. Además, la ausencia de una implementación de un repositorio distribuido y redundante introduce un importante punto débil en el núcleo de la arquitectura.

### 3.4.4. Amigo

El proyecto Amigo [FP6-IST Amigo, 2008a] [FP6-IST Amigo, 2008b] junto con algunos de los trabajos derivados de él, constituyen uno de los mayores esfuerzos realizados en el ámbito del desarrollo de middleware para entornos de IAm. Es el resultado del esfuerzo conjunto de quince compañías europeas y varios centros de investigación especializados en tecnologías móviles, redes digitales para el hogar y electrónica de consumo. Su objetivo central es el desarrollo de middleware para la integración dinámica de sistemas heterogéneos con el fin de alcanzar un alto grado de interoperabilidad entre servicios y dispositivos. El proyecto se centra especialmente en entornos domésticos, donde la falta de interoperabilidad entre tecnologías de distintos fabricantes y la ausencia de servicios realmente atractivos y adaptados al usuario hace que la tecnología esté lejos de proporcionar una mejora en la calidad de vida de las personas en un modo similar al visionado por la IAm.

#### Arquitectura Amigo

El proyecto Amigo se fundamenta en una arquitectura orientada a servicios compuesta por tres elementos fundamentales:

- **Capa Base de Middleware.** Contiene la funcionalidad necesaria para facilitar la construcción de un entorno de servicios y dispositivos conectados.
- **Capa de Servicios Inteligentes de Usuario.** Contiene la funcionalidad necesaria para facilitar la creación de un entorno adaptado a los usuarios.
- **Framework de Programación y Despliegue.** Contiene módulos y APIs que facilitan a los desarrolladores la creación de servicios Amigo, proporcionando soporte para interoperabilidad, seguridad y gestión de información contextual.

El Middleware de la Capa Base de Amigo es una solución para entornos domésticos que permite integrar un importante número de tecnologías existentes en términos de plataformas de servicios y protocolos de comunicación. La solución está basada en la utilización de mecanismos de descubrimiento y comunicación de servicios y dispositivos disponibles en una red, incluyendo soporte para varios protocolos estándar existentes. Este middleware facilita la interoperabilidad e integración de plataformas de servicios heterogéneas, soportando el uso de tecnologías semánticas para la descripción de características tanto funcionales como propias de la arquitectura. A través de estas tecnologías semánticas se proporcionan las bases para la resolución de la problemática del acceso a dispositivos y servicios heterogéneos, descubrimiento de servicios y consciencia del contexto. Por último, esta capa también proporciona mecanismos de seguridad para autenticación, autorización y encriptación.

A través del Framework de Programación y Despliegue, la Capa Base de Middleware ofrece módulos que facilitan el desarrollo de servicios Amigo tanto en la plataforma Java como en la plataforma .NET. Como se ha mencionado, Amigo soporta distintos protocolos de comunicación y descubrimiento de servicios. Por tanto, los desarrolladores pueden seleccionar el protocolo que deseen por medio de las APIs que les proporciona este framework, manteniendo la interoperabilidad con el resto de servicios. Los protocolos soportados incluyen: protocolos de interacción orientada a servicios (SOAP [SOAP, 2007], UPnP [Donoho et al., 2008] y RMI [RMI, 2015]); y protocolos de descubrimiento de servicios (SLP [Guttman et al., 1999], UPnP [Donoho et al., 2008] y WS-Discovery [WS-Discovery, 2009]).

Los Servicios Inteligentes de Usuario se ubican entre los usuarios y los proveedores de servicios, y proporcionan información contextual, combinan múltiples fuentes de información, crean perfiles de usuario y facilitan la adaptación del sistema a la situación del usuario y su entorno. Concretamente, esta capa proporciona los siguientes servicios principales:

- **Servicio de Gestión del Contexto.** Este servicio trata con la colección de información disponible con el fin de establecer la información contextual y transformarla a un formato adecuado que pueda ser procesado posteriormente por los servicios. El servicio utiliza ontologías para representar el conocimiento y para razonar y realizar inferencias sobre el mismo.
- **Servicio de Modelado y Perfilado de Usuario.** Facilita la personalización de los servicios ofrecidos al usuario a través de la combinación de información contextual con información de los perfiles de usuario.
- **Servicio de Consciencia y Notificación.** Facilita el desarrollo de aplicaciones que permiten al usuario ser consciente de cambios significativos en su contexto con un mínimo esfuerzo. Para ello, este servicio permite que las aplicaciones se subscriban para ser automáticamente notificadas de cambios en el contexto. Desde el punto de vista de los usuarios, es posible crear un perfil que define el tipo de información de la que desean ser notificados, así como la frecuencia y forma de la notificación.
- **Servicio de Interfaz de Usuario.** Este servicio implementa un framework de interacción que permite la combinación de múltiples dispositivos de modo que estos puedan ser gestionados de forma sensible al contexto, por ejemplo, seleccionando los dispositivos más apropiados dependiendo de sus capacidades de presentación de la información.

De acuerdo con este modelo conceptual, Amigo propone un framework de seguridad que proporciona mecanismos de privacidad y autorización. El primero se ocupa de la verificación de las identidades que acceden al sistema, y se trata de una solución basada en Kerberos [Neuman and Ts'o, 1994]. El segundo se encarga de controlar el acceso a los recursos a través de un sistema de control de acceso.

**Arquitectura de servicios interoperables**

Los trabajos realizados dentro del proyecto Amigo conducentes al desarrollo de una arquitectura orientada a servicios que proporcione soluciones avanzadas de interoperabilidad son probablemente el aspecto más destacado del desarrollo global realizado. Al contrario que la mayoría de arquitecturas desarrolladas para entornos de IAm, la aproximación de Amigo no impone una tecnología de middleware concreta, al contrario, soporta distintas tecnologías que son integradas, estableciendo mecanismos de interoperabilidad entre ellas.

Por tanto, el objetivo de Amigo no es desarrollar una plataforma de servicios nueva imponiendo un middleware homogéneo para todos los dispositivos y servicios del sistema, sino introducir una arquitectura de servicios abstracta con la que se pueden representar distintas plataformas de servicios a través de la abstracción de sus características fundamentales.

La arquitectura Amigo [Georgantas et al., 2005] está basada en una arquitectura SOA típica. Incluye un modelo y un lenguaje para la descripción de servicios, mecanismos de emparejamiento entre servicios y un repositorio para el registro y descubrimiento de servicios. Para soportar la interoperabilidad, Amigo propone una solución middleware denominada AmIi [Georgantas et al., 2010], que soporta la integración de tecnologías heterogéneas existentes en cada uno de estos aspectos propios de una arquitectura SOA. Las soluciones de interoperabilidad se basan en mecanismos de traducción entre distintas tecnologías.

En primer lugar, el middleware AmIi propone un modelo de servicios abstracto que permite integrar descriptores de servicios de otros lenguajes, como OWL-S, WSDL o SAWSDL. El modelo de Amigo se compone de dos partes: un *profile* y un *grounding*. El profile de un servicio se describe como un conjunto de capacidades funcionales y un conjunto de propiedades no-funcionales, mientras que el grounding del servicio describe la forma de acceso al servicio. Una capacidad puede además tener una conversación asociada, que permite especificar la forma de realizar funcionalidades compuestas en base a la combinación de otras funcionalidades. Por último, las propiedades no-funcionales están relacionadas con información de contexto y de QoS del servicio.

Para la realización concreta de este modelo, la arquitectura Amigo define un lenguaje de descripción de servicios propio (ASDL). Para su implementación combina dos lenguajes ya existentes, SAWSDL [Kopecky et al., 2007] y WS-BPEL [Jordan and Alves, 2007]. Se emplea SAWSDL para describir las capacidades de los servicios, mientras que WS-BPEL se utiliza para describir las conversaciones asociadas con las capacidades, aprovechando las características de este lenguaje para la especificación de flujos de procesos.

AmIi también incluye un sistema de plugins para soportar protocolos de distin-

tos registros de servicios, de modo que se habilita la búsqueda de servicios descritos utilizando distintas tecnologías. Para ello se utiliza un registro central en el que se almacenan las descripciones de servicios heterogéneas traducidas a un lenguaje común. Finalmente, un motor de emparejamiento permite seleccionar los servicios que se corresponden con una búsqueda hecha por un cliente, pudiendo combinarse tanto descripciones semánticas como descripciones sintácticas.

Por otro lado, AmIi proporciona soporte para el manejo de servicios con distintos protocolos de comunicación, de nuevo basándose en un mecanismo de traducción entre protocolos [Thomson et al., 2008]. Esta solución asume la existencia de un repositorio universal que proporciona los elementos necesarios para realizar la traducción al protocolo concreto de un servicio. No obstante, Amigo únicamente se centra en un modelo de interacción entre servicios de tipo petición/respuesta, soportando interoperabilidad entre protocolos RPC como RMI, CORBA y SOAP.

Este modelo orientado a servicios desarrollado por Amigo constituye la solución de interoperabilidad más completa de entre las que podemos encontrar en los principales proyectos de IAm que abordan la complejidad de estos entornos partiendo de la construcción de una arquitectura software de carácter general. No obstante, Amigo defiende el modelo de comunicación RPC como el mecanismo esencial para la comunicación en la Computación Orientada a Servicios. Esto deja fuera a otros modelos de interacción asíncrona como el modelo publicación/subscripción que favorecen el desacoplamiento entre componentes y el comportamiento autónomo de los mismos; siendo ambas características demandadas por los sistemas de IAm. Por otro lado, la integración de SAWSDL y WS-BPEL como lenguajes para la descripción de servicios semánticos enriquecidos nos ciñe a la utilización de la base del lenguaje WSDL con una serie de añadidos para soportar tanto anotaciones semánticas (SAWSDL) como descripciones de procesos (WS-BPEL). En parte por el diferente objetivo de ambos lenguajes y en parte porque WSDL es un lenguaje creado únicamente para la descripción sintáctica de Servicios Web, el resultado es una sintaxis compleja y poco intuitiva que dificulta la definición de servicios no triviales.

#### Integración de dispositivos comunes

En el área de la integración de dispositivos, Amigo apuesta por la utilización de UPnP [Donoho et al., 2008] como la tecnología principal, centrada en dispositivos multimedia. No obstante, también propone la utilización de componentes proxy concretos para encapsular el acceso a dispositivos basados en distintas tecnologías. Así, en su arquitectura se incluyen de forma predefinida proxys para algunas tecnologías domóticas habituales. Por tanto, se mantiene la heterogeneidad entre proxys de distintas clases de dispositivos, dado que la arquitectura Amigo no proporciona ninguna interfaz común que uniformice la operación de cualquier dispositivo. Por tanto las aplicaciones concretas necesitan una programación y un conocimiento

específicos para interaccionar con dispositivos de diferentes ámbitos.

La ausencia de un modelo de acceso a redes de dispositivos totalmente uniforme traslada parte de la complejidad de tratar con un entorno complejo y heterogéneo a los servicios de más alto nivel y a las aplicaciones, que idealmente deberían tener una visión más abstracta del entorno físico en el que operan.

### 3.4.5. PERSONA

El proyecto europeo PERSONA [FP6-IST PERSONA, 2010] (PERceptive Spaces prOmoting iNdependent Aging) tiene como principal objetivo el desarrollo de soluciones sostenibles que promocionen la inclusión social y la independencia de las personas mayores, a través del desarrollo de una plataforma basada en estándares abiertos, y la definición de un conjunto de servicios adaptados a las necesidades de los usuarios mediante la armonización de tecnologías y conceptos AAL (Ambient Assisted Living). Se trata de un proyecto que se ha realizado entre los años 2007 y 2010, y ha estado financiado en parte por fondos de la Comisión Europea.

#### Arquitectura conceptual

La arquitectura propuesta por PERSONA se diseña en base a una serie de requisitos que se corresponden en gran medida con los requisitos generales para la construcción de sistemas de IAm. No obstante, en este caso se restringe el entorno a ámbitos domésticos de AAL y pone especial énfasis en que la arquitectura favorezca el desarrollo e integración modular y flexible de componentes de alto nivel.

Desde un punto de vista más técnico, la arquitectura de PERSONA parte de una serie de bases tecnológicas y de diseño que en buena parte están fundamentadas en otros trabajos previos:

- Utilización del middleware SODAPOP [Hellenschmidt and Kirste, 2004], que proporciona una arquitectura basada en distintos buses de comunicación.
- Se adopta una aproximación orientada a servicios inspirada en el modelo de descubrimiento y composición de servicios del proyecto Amigo [Janse et al., 2008] [Thomson et al., 2008].
- Inclusión de un conjunto de servicios básicos del sistema inspirado principalmente en las propuestas del proyecto Amigo [Janse et al., 2008].
- Adopción del framework OSGi [OSGi Alliance, 2015] para la gestión del ciclo de vida de los servicios del sistema.
- Utilización de un modelo de red ad-hoc basada en unos protocolos de descubrimiento concretos, como UPnP y Bluetooth.

- Utilización de la Máquina Virtual de Java como plataforma independiente del sistema operativo.

El hecho de que la arquitectura de PERSONA utilice un modelo basado en SOA no significa que todas las funcionalidades se abstraigan como servicios. Así, PERSONA contempla dos tipos de funcionalidades especiales que quedan fuera del modelo de servicios, se trata de aquellas relacionadas con la información contextual en forma de eventos y la interacción con el usuario. En consecuencia, y siguiendo el modelo de SODAPOP, la arquitectura define los siguientes tipos de buses de comunicación: bus de contexto, buses de interacción con el usuario (input y output) y bus de servicios. Los dos primeros siguen un modelo de comunicación basado en eventos mientras que el último sigue un modelo RPC.

Un sistema PERSONA se organiza en nodos de ejecución. Todas las instancias del middleware ejecutándose en distintos nodos tienen capacidades para encontrarse unas a otras, de modo que los componentes tienen la visión de un modelo virtual en el que todos los buses de comunicación están directamente conectados. Así, un componente simplemente se registra en un conjunto de buses especificando el rol que va a jugar en cada uno de ellos. Los roles posibles para un componente registrado en un bus de eventos son publicador y subscriptor; los roles posibles en un bus de servicios son llamador (o cliente de servicios) y receptor de llamadas (o proveedor de servicios). Esto, junto con la posibilidad de definir ontologías asociadas a los buses, ayuda a garantizar la coherencia del sistema en lo referente al flujo de información entre sus componentes.

La arquitectura incluye una serie de servicios de sistema que cubren una serie de funcionalidades que son necesarias para el propio funcionamiento de la arquitectura:

- **Dialog Manager.** Componente centralizado que accede a la información de una base de conocimiento central con el fin de determinar la acción (servicio) a ejecutar ante una situación contextual dada.
- **Contex History Entrepot (CHE).** Un componente centraliazado único que se encarga de almacenar toda la historia de eventos del contexto generados en el sistema en un repositorio central.
- **Situation Reasoner.** Utiliza la base de datos del CHE para inferir nueva información contextual utilizando el lenguaje SPARQL [Prud'hommeaux and Seaborne, 2008].
- **Service Orchestator.** La arquitectura permite definir meta-servicios estáticos que se construyen como una combinación de servicios atómicos reales. El Service Orchestator es el componente encargado de interpretar la información que describe los meta-servicios y realizar las acciones necesarias para la ejecución de los servicios atómicos que los componen. Estos meta-servicios

deben ser definidos y añadidos al sistema por un administrador a través de una interfaz de usuario y se almacenan en un BD centralizada.

- **Privacy-aware Identity and Security Manager (PISM).** Proporciona mecanismos de autenticación, permisos y encriptación.
- **Space Gateway.** Se trata de un gateway que permite el acceso externo a servicios de la arquitectura a través de una URL fija.

Esta arquitectura destaca por proponer un modelo conceptual elaborado que impone una metodología para la construcción de sistemas de IAm en torno a una serie de componentes bien definidos y a un esquema de interacción claramente estructurado. Aún así, en el modelo de PERSONA se pueden identificar algunos puntos débiles, como el carácter centralizado de algunos componentes básicos de la arquitectura o la ausencia de mecanismos automáticos para la construcción de servicios compuestos. Además, la definición estática de los meta-servicios por parte de un usuario administrador puede ser útil en entornos relativamente pequeños y controlados, pero carece del dinamismo que requiere la adaptación a entornos complejos y abiertos típicos de IAm.

#### Middleware PERSONA

El modelo conceptual descrito se concreta en un middleware organizado en capas y compuesto por una serie de bundles OSGi:

- **Abstract Connection Layer (ACL).** Es la capa inferior y es la responsable de la conexión entre distintos nodos (instancias del middleware) del sistema. Esta capa soporta internamente varios protocolos de descubrimiento y envío de mensajes, en concreto UPnP, R-OSGi y Bluetooth. Una abstracción común de estas tecnologías facilita la interconexión transparente entre distintos nodos y distintos protocolos.
- **SodaPop Layer.** Implementa los conceptos de *bus* (tanto basado en eventos como basado en invocaciones) y *mensaje*, incluyendo una interfaz para la serialización de mensajes a través de los buses.
- **PERSONA-Specific Layer.** En esta capa se implementan los tipos concretos de bus contemplados en el modelo conceptual: bus de contexto, bus de servicios y bus de interacción con el usuario.

El bus de contexto proporciona un canal de comunicación ligero que es utilizado para publicar eventos de cambio en el contexto. Los elementos publicadores de información contextual deben encontrar en primer lugar el servicio OSGi encargado de proporcionar acceso al bus de contexto y registrarse en él. Los suscriptores de contexto deben adicionalmente especificar un filtro para indicar los eventos del

contexto en los que están interesados. La estrategia del bus de contexto es realizar un simple broadcast de todos los eventos recibidos a todos los componentes subscritos.

En el bus de servicios, los proveedores de servicios proporcionan una descripción de los servicios disponibles utilizando perfiles de OWL-S [W3C, 2004], en el momento en que se registran los servicios en el sistema. Los clientes de servicios hacen sus peticiones a través de la creación de consultas sobre perfiles de servicio concretos. La utilización de un lenguaje semántico como OWL-S para la descripción de servicios junto con el modelo de interacción basado en buses, proporciona un importante nivel de desacoplamiento entre los servicios y los clientes de esos servicios. No obstante, PERSONA no contempla opciones de interoperabilidad fuera de su plataforma. Además, al utilizar la plataforma OSGi para la construcción de sus componentes, éstos están atados a ser implementados sobre la plataforma Java.

#### Abstracción de la sensorización del entorno

El middleware de PERSONA incluye un framework que soporta la integración de distintas redes de sensorización inalámbrica, denominado SAIL (del inglés, Sensors Abstraction and Integration Layer) [Girolami et al., 2008]. Esta capa se ha diseñado con el objetivo de integrar únicamente dispositivos con unas capacidades de cómputo razonables, quedando excluidos dispositivos de reducidas capacidades como los propios de la domótica o componentes *wearable*. Para este tipo de dispositivos se propone la creación de gateways ad-hoc que adapten las tecnologías no soportadas por SAIL.

La arquitectura del framework SAIL está basada en una división en tres capas, denominadas Capa de Acceso, Capa de Abstracción y Capa de Integración, construidas sobre OSGi. Los controladores de red residen en la Capa de Acceso para manejar protocolos de comunicación específicos de cada red de sensorización soportada. La Capa de Abstracción de SAIL define un modelo de sensores común para todas las tecnologías soportadas. La realización de esta capa se basa en el patrón de interfaces SPI [Motorola, 2003]. Los controladores de red concretos quedan ocultos a través de un modelo común y una API para la capa superior (la Capa de Integración). En esta última capa se proporciona una visión lógica de los sensores adaptada a las necesidades de las aplicaciones.

La información proveniente de sensores se publica en forma de eventos a través del bus de contexto. La actuación sobre dispositivos se realiza utilizando invocaciones RPC a través del bus de servicios.

Por tanto, en lo tocante a la sensorización del entorno, PERSONA también destaca por proponer un modelo avanzado que va más allá de la alternativa común consistente en implementar un conjunto de servicios heterogéneos ad-hoc para encap-

sular el acceso a una serie de dispositivos concretos. Sin embargo, también presenta ciertas carencias, como son los elevados requisitos computacionales que impiden su utilización directa en pequeños dispositivos, y la falta de un modelo uniforme de dispositivos descrito en un lenguaje neutral que posibilite la interoperabilidad.

### 3.4.6. SOPRANO

SOPRANO (Service-oriented Programmable Smart Environments for Older Europeans) [Sixsmith et al., 2009] es un proyecto enfocado al desarrollo de entornos tecnológicos basados en el concepto de AAL (Ambient Assisted Living) que mejoren la calidad de vida de las personas mayores, proporcionándoles un mayor grado de independencia en sus hogares. Se trata de un proyecto desarrollado entre los años 2007 y 2010 por un consorcio de empresas y grupos de investigación, y que ha sido en parte financiado por fondos de la Unión Europea.

El núcleo tecnológico del proyecto es el desarrollo de un middleware que facilite la construcción de soluciones integradas en un entorno tan complejo como son los entornos de AAL. Para ello, la aproximación de SOPRANO se basa en la utilización de ontologías para la gestión de la información del entorno y la utilización de una arquitectura SOA para el acceso a los dispositivos del entorno. Esta aproximación propone una división del sistema en base a tres aspectos fundamentales: sensores/actuadores, información de contexto, y comportamiento/funcionalidad del sistema.

La función principal del middleware SOPRANO es recoger dinámicamente la información contextual del entorno, a través de diferentes sensores, y transferirla, en el nivel de abstracción adecuado, a la ontología que representa el conocimiento del entorno y a un sistema de gestión de reglas [Wolf et al., 2008]. Para ello la arquitectura SOPRANO define los siguientes componentes principales:

- **Semantic Service Registry.** El acceso a los sensores y actuadores del entorno se realiza a través de servicios que ocultan la complejidad de la interacción con hardware heterogéneo. Estos servicios incorporan una descripción semántica de los datos que esperan recibir o que generan. Además, la arquitectura proporciona un registro de servicios que facilita la localización de los servicios en un sistema SOPRANO.
- **Context Manager.** Este componente proporciona funcionalidades para explotar la información contextual de que dispone el sistema en un momento dado. Así, el Context Manager analiza constantemente la información procedente del entorno e intenta deducir información contextual de más alto nivel semántico.
- **Procedural Manager.** Este componente es responsable de gestionar las reacciones del sistema ante cambios en el contexto o ante peticiones explícitas de

los usuarios. Así, como resultado de analizar una nueva situación, el Procedural Manager compila una descripción abstracta de un proceso encargado de manejar dicha situación. Este proceso abstracto se compone en base a un repositorio de plantillas de procesos. Estas plantillas describen peticiones abstractas de servicios en vez de referencias a servicios concretos.

- **Composer.** Este componente recibe y procesa los planes del Procedural Manager. A partir de un plan, el Composer busca, compone y parametriza servicios concretos con el objetivo de ejecutar el proceso definido en el plan.
- **Ontología del Contexto.** SOPRANO utiliza como elemento central del sistema una ontología para representar el contexto. Esta ontología sirve como nexo de unión entre los principales componentes del sistema dado que proporciona diferentes niveles semánticos de abstracción de la información de acuerdo con los requisitos de los distintos componentes.

La plataforma SOPRANO se apoya en una plataforma OSGi orientada a servicios que le proporciona las herramientas básicas para la gestión distribuida de los componentes del sistema. Esto implica que todos los componentes software son integrados como servicios implementados en Java. El framework OSGi es el encargado de dar soporte a la comunicación entre componentes, de gestionar el ciclo de vida completo de los servicios y de proporcionar el registro para la búsqueda de los servicios del sistema.

El descubrimiento y ejecución de servicios se fundamenta principalmente en la utilización de un modelo de descripción semántica de servicios que facilite la automatización de estas tareas. SOPRANO se apoya en un lenguaje de descripción de servicios propio llamado DIANE [Klein et al., 2005] [Küster et al., 2007]. La ontología de descripción de servicios de DIANE se basa en muchos de los conceptos de OWL-S, a la vez que incorpora soluciones para el manejo de información difusa y con incertidumbre en la especificación de servicios. Así, las funcionalidades que ofrece un servicio son descritas en base a un conjunto parametrizable de resultados que produce la ejecución del servicio, mientras que una petición de servicio se describe como un conjunto difuso de resultados deseados.

Dentro de la arquitectura de SOPRANO, la operación de todos los componentes gira en gran medida en base a su relación con el contexto y la ontología que SOPRANO define para la descripción de dicha información contextual [Klein et al., 2007] [Schmidt and Schmidt, 2006]. El middleware proporciona, al resto de componentes del sistema, una API para la interacción con el contexto a distintos niveles de abstracción semántica.

Los sensores y actuadores deben ser incorporados al sistema a través de servicios OSGi. Además, estos servicios deben proporcionar o manejar información de acuerdo con el vocabulario definido por la ontología de bajo nivel de SOPRANO. De esta forma se garantiza que la información obtenida por los sensores puede ser

procesada por el sistema e incorporada al contexto. De forma similar a como ocurre en el proyecto PERSONA, este modelo de abstracción de dispositivos tampoco es interoperable al estar ligado a la plataforma Java-OSGi.

En el siguiente nivel de abstracción, SOPRANO ofrece una API de acceso a funcionalidades de consulta y escritura del contexto. Su objetivo es posibilitar que los desarrolladores puedan incorporar componentes que realicen procesos de combinación y agregación de información contextual. De forma general, en este nivel existen servicios que procesan información almacenada en las ontologías de bajo nivel y la transforman en información propia de la ontología de alto nivel, definida en el dominio de asistencia a personas mayores en el hogar. De igual manera, existen otros servicios que transforman información de la ontología de alto nivel en peticiones concretas a los servicios de bajo nivel que representan a actuadores del entorno. Este modelo no impone ningún formalismo para el desarrollo de los procesos de razonamiento sobre la información del contexto.

En un último nivel de abstracción semántica, se gestiona el conocimiento procedural que permite la definición de procesos adaptables a un entorno concreto y a un usuario concretos. Esta descripción del comportamiento del sistema a alto nivel es manejada en SOPRANO utilizando un mecanismo basado en la definición de procesos plantilla parametrizables. Para ello, el formalismo utilizado está basado en el lenguaje de modelado de procesos WS-BPEL. Con esta estructura, los responsables de configurar el sistema pueden crear instancias de las plantillas de procesos y concretarlas con servicios específicos en aquellos puntos donde sean parametrizables. Por tanto, se puede decir que las plantillas de procesos representan los casos de uso soportados por un sistema SOPRANO concreto.

Esta arquitectura facilita el diseño de funcionalidades configurables en base a servicios distribuidos en un entorno limitado y definido por una ontología concreta. No obstante, la necesidad de realizar la composición de servicios de forma manual como parte de la configuración del sistema para un entorno y usuario concreto, junto con la ausencia de interoperabilidad con otros sistemas, le resta capacidades para soportar entornos más extensos, abiertos y heterogéneos. Por otra parte, los procesos de inferencia, que han de hacerse sobre ontologías a todos los niveles del sistema, implicarán una importante degradación del rendimiento del sistema en un entorno que requiera de la gestión de un contexto amplio y complejo.

### 3.4.7. Comparación de arquitecturas software para IAm

Después de analizar algunas de las soluciones arquitectónicas para entornos de IAm existentes, se pone de manifiesto la dificultad de realizar una comparación exhaustiva entre ellas. Los objetivos de estos proyectos no son los mismos, los ámbitos a los que se orientan divergen, el grado de concreción de los distintos modelos

teóricos en implementaciones concretas es muy distinto y el nivel de detalle en la documentación disponible acerca de estos proyectos es también muy diferente. No obstante, en este apartado se presenta una comparación entre los proyectos más relevantes con el fin de clarificar donde se posicionan estos trabajos dentro de la carrera por alcanzar los ambiciosos objetivos de la IAm.

La comparación comienza por establecer el conjunto de requisitos fundamentales que, idealmente, cada arquitectura de IAm debería satisfacer. Este conjunto está basado en el estudio del estado del arte y en nuestra propia experiencia en el desarrollo de soluciones para entornos de IAm. Así, se parte del conjunto de requisitos que ya fueron identificados y descritos en la sección 3.1.2: heterogeneidad, escalabilidad, sensibilidad al contexto, interacción natural, movilidad, invisibilidad, interoperación espontánea, interoperabilidad, integridad-seguridad, privacidad-confianza, inteligencia social.

A mayores de estos requisitos fundamentales se ha decidido incluir en la comparativa tres nuevos requisitos relacionados con la generalidad de las soluciones y con las posibilidades y facilidades reales para reutilizar los resultados de estos proyectos en la construcción de nuevas soluciones en el campo de la IAm. El primero de los requisitos que se ha añadido se refiere a las facilidades destinadas tanto a los desarrolladores como a los instaladores de un sistema concreto. En este sentido, se valorará la inclusión de herramientas que faciliten la creación de servicios y componentes del sistema, la disponibilidad de documentación de diseño detallada o la inclusión de herramientas gráficas que faciliten la configuración y monitorización del sistema. Asimismo, se tendrá en cuenta el grado en el que la arquitectura permite a los desarrolladores centrar sus esfuerzos en la implementación de nuevas funcionalidades sin verse afectados por la estructura y complejidad tecnológica del middleware. El segundo de los requisitos añadidos se refiere al nivel de generalidad de la solución. Idealmente, una arquitectura de IAm debería diseñarse de forma que pudiese ser aplicada a todos los entornos donde las personas realizan sus actividades cotidianas, con el fin de mejorar su calidad de vida. Por tanto, evitando imponer de antemano restricciones referentes al tipo de usuario o al tipo de entorno físico con los que se va lidiar. El tercer requisito añadido se refiere a la accesibilidad y vigencia de las soluciones. En este sentido, en primer lugar se valorará la libre distribución de especificaciones y del código fuente de implementaciones concretas de las arquitecturas. En segundo lugar se valorará la vigencia de las soluciones en cuanto a su adecuación a los entornos computacionales y las tecnologías más comunes en el presente.

Tomando como base este conjunto de requisitos, en la tabla de la Figura 3.2 se muestra una visión esquemática del resultado de la comparación entre arquitecturas realizada.

Como hemos visto, Amigo es uno de los proyectos más amplios y con un enfo-

| | AMIGO | CHIL | LAICA | OXYGEN | PERSONA | SOPRANO |
|---|---|---|---|---|---|---|
| Heterogeneidad | Media | Media | Baja | Baja | Media | Media |
| Escalabilidad | Alta | Media | Alta | Alta | Media | Baja |
| Sensibilidad al contexto | Media | Baja | No | Baja | Media | Media |
| Interacción natural | Baja | Media | No | Media | Baja | No |
| Movilidad | Baja | No | Baja | No | No | No |
| Invisibilidad | Media | Media | Baja | Media | Media | Alta |
| Interoperación espontánea | Media | Baja | No | Baja | Baja | Media |
| Interoperabilidad | Media | Baja | No | No | Baja | Baja |
| Integridad y seguridad | Baja | Media | Media | Baja | Baja | Baja |
| Privacidad y confianza | Media | Baja | No | Media | Baja | No |
| Inteligencia social | No | No | No | No | No | No |
| Facilidades desarrollador | Media | Media | Baja | Baja | Baja | Media |
| Acceso y vigencia | Baja | Baja | Baja | Baja | Baja | Baja |
| Generalidad | Alta | Media | Baja | Media | Media | Baja |

Tabla 3.2: Comparación de las arquitecturas de IAm más relevantes.

que más general en el ámbito de la IAm. Proporciona una arquitectura ambiciosa que abarca un importante número de los requisitos fundamentales de cualquier sistema de IAm y, aunque en sus objetivos originales se orienta principalmente hacia entornos del hogar, sus soluciones pueden considerarse las más genéricas de todas las evaluadas en esta comparativa. Destaca, además, tanto por la detallada documentación técnica disponible (manuales para desarrolladores, manuales de usuario y ejemplos de implementación) como por proporcionar a la comunidad gran parte de sus implementaciones con licencias de código abierto. Su modelo de servicios semánticos distribuidos incluye herramientas que facilitan la automatización de los procesos de descubrimiento y composición de servicios, al mismo tiempo que incluye mecanismos que proporcionan interoperabilidad entre servicios definidos utilizando distintos modelos de descripción de servicios existentes. Sin embargo, su modelo de interacción basado únicamente en protocolos de comunicación RPC limita notablemente las posibilidades de interacción e interoperación espontánea entre servicios. Otro punto destacable de Amigo es que ya desde su diseño inicial se tuvieron en cuenta algunos aspectos relacionados con la seguridad y la privacidad,

finalizando con el desarrollo de un sistema de autenticación basado en Kerberos y un sistema RBAC (autorización basada en permisos y roles), que se integraron con buena parte de los componentes de la arquitectura. Por otro lado, entre los puntos débiles de Amigo hay de que destacar la ausencia de mecanismos que permitan la movilidad de componentes, así como la ausencia de mecanismos de tolerancia a fallos y herramientas para la monitorización del sistema, que en conjunto redunden en una mayor integridad y confianza del mismo. Además, aunque Amigo proporciona una serie de proxys que encapsulan el acceso a varias tecnologías de dispositivos (e.j. UPnP y algunas redes domóticas), no proporciona un modelo común que garantice un acceso uniforme a la potencial heterogeneidad de un entorno físico de IAm real. Por último, es necesario reseñar que a pesar de que Amigo proporciona una serie de frameworks muy bien documentados, su utilización para la implementación de sistemas concretos se vuelve compleja por el hecho de que el desarrollador se ve obligado a codificar parte de la lógica de acceso a los mecanismos de seguridad, sensibilidad al contexto o composición de servicios en sus propios servicios. Esto se ve agravado, además, por la utilización de librerías que a día de hoy están obsoletas y que dificultan la extensión de la arquitectura.

El proyecto CHIL utiliza una aproximación basada en tecnologías de sistemas multi-agente especialmente orientada a la sensorización/actuación y a la interacción natural con el entorno. En este sentido, destacan sus aportaciones en la construcción de una arquitectura distribuida, adaptable y con capacidades de tolerancia a fallos para la sensorización uniforme de entornos físicos heterogéneos. No obstante, de forma similar a como ocurre con Amigo, su modelo de acceso a los dispositivos físicos del entorno no es todo lo uniforme que sería deseable. También es destacable la aportación de CHIL a la construcción tanto de interfaces naturales (principalmente basadas en imagen y voz), como de sistemas de interacción multimodal dinámicamente adaptables en función del contexto. La solución incluye un modelo declarativo que facilita a los desarrolladores la definición de servicios plantilla, así como una serie de herramientas para la gestión global del sistema. Por el contrario, entre los puntos débiles de CHIL cabe destacar la ausencia de soluciones de movilidad, la ausencia de mecanismos de seguridad y privacidad y las casi nulas capacidades para la interoperación espontánea entre servicios así como las reducidas posibilidades de interoperabilidad con componentes de otras plataformas.

En el caso del proyecto LAICA, su principal aportación es la creación de una infraestructura distribuida y robusta, basada en tecnologías de sistemas multi-agente, que facilita el despliegue de una red de sensorización y monitorización colaborativa del entorno. El proyecto está claramente orientado a entornos exteriores urbanos, lo que le resta generalidad a la solución, no obstante proporciona unas bases interesantes que podrían ser extensibles a otros entornos. También merece la pena destacar que la arquitectura proporciona una serie de herramientas que le proporcionan cierto grado de tolerancia a fallos y que permiten la monitorización global de

todo el sistema. Esta solución presenta un modelo de interacción entre sus componentes sencillo basado en el intercambio de comandos y datos codificados en XML. Aunque este modelo proporciona cierta independencia entre los componentes, no posibilita ni la interoperación espontánea entre componentes del propio sistema ni la interoperabilidad con otros sistemas. Finalmente, LAICA tampoco provee de ningún mecanismo explícito que garantice la privacidad y confianza a lo largo de todo el sistema.

El proyecto Oxygen agrupa a numerosos desarrollos en los ámbitos de los sistemas pervasivos y la computación ubicua, pero pone el acento en la interacción natural y en la construcción de dispositivos multiprotocolo y autónomamente adaptables para la sensorización de entornos de trabajo. No obstante, también hay que destacar el desarrollo de distintos frameworks que facilitan la construcción de sistemas de IAm basados en componentes distribuidos y reutilizables que colaboran entre sí. En este sentido, el proyecto proporciona una plataforma de agentes sobre la que se definen los servicios del sistema. Estos servicios incorporan ciertas capacidades de interoperación en base a un sistema de descubrimiento basado en restricciones. El sistema también incorpora mecanismos para garantizar la privacidad y seguridad en las comunicaciones entre sus componentes. Entre las debilidades principales de las soluciones de Oxygen destacan la ausencia de mecanismos avanzados para la gestión del contexto, la ausencia de soluciones completas de movilidad de componentes y la ausencia de mecanismos que proporcionen algún grado de interoperabilidad con otros sistemas. Además, en lo referente al requisito de heterogeneidad, Oxygen no proporciona ningún modelo uniforme de acceso a dispositivos, ya que todas sus soluciones en este aspecto son realizadas ad-hoc sobre tecnologías concretas.

En el caso del proyecto PERSONA, quizá su mayor aportación respecto a otras aproximaciones como Amigo es la definición de un modelo conceptual de referencia más elaborado y un modelo de interacción entre componentes más rico. Así, haciendo uso de soluciones parciales previas, PERSONA crea una arquitectura en torno a la utilización de servicios y canales de comunicación semánticamente anotados a través de los que se gobierna la interacción entre los distintos componentes del sistema. Además, a pesar de que el proyecto está orientado a entornos de asistencia a personas dependientes, la arquitectura que propone es fácilmente generalizable para ser utilizada en otros entornos. En lo referente al contexto, la arquitectura PERSONA define una serie de mecanismos y componentes que permiten la gestión global e inferencia de información contextual en base a la utilización de un lenguaje de descripción de ontologías y un lenguaje semántico de consulta. No obstante, algunos de estos elementos fundamentales del núcleo de la arquitectura son centralizados, lo que repercute en la escalabilidad de la arquitectura en su conjunto. También es de destacar el modelo de abstracción de dispositivos (SAIL) que propone PERSONA para lidiar con la heterogeneidad del entorno físico. En este sentido,

SAIL define un modelo más elaborado que el de Amigo, por ejemplo, pero deja fuera de su solución uniforme a pequeños dispositivos con escasos recursos computacionales. En lo referente a las mayores debilidades de esta arquitectura, esta carece de soluciones de movilidad y aporta muy reducidas capacidades para la interoperabilidad con otros sistemas y para la interoperación espontánea entre servicios. De hecho, PERSONA contempla la composición de servicios semánticos en su modelo de referencia, pero esta ha sido implementada de forma estática. Otra limitación de la arquitectura es su sistema de seguridad y privacidad que únicamente implementa mecanismos para el cifrado de datos y autenticación de usuarios, pero no incluye ningún sistema de permisos. Por último, la arquitectura se caracteriza por unos requisitos de configuración elevados y por la ausencia de documentación técnica detallada accesible públicamente.

El proyecto SOPRANO persigue objetivos muy similares a los de PERSONA, pero propone una arquitectura más condicionada por el dominio concreto al que se orienta (asistencia a personas dependientes) y por tanto se trata de una aproximación menos generalizable. El modelo conceptual que propone gira en torno a la información contextual almacenada en una ontología que contiene distintos niveles de abstracción del conocimiento. Incluye mecanismos para la obtención de la información contextual a partir de a sensorización del entorno y mecanismos para realizar inferencias sobre el contexto a distintos niveles. En este sentido, la arquitectura propone un modelo de gestión del contexto interesante, pero a su vez hace que prácticamente todos los procesos que se ejecutan en un sistema SOPRANO tengan una fuerte dependencia del sistema de gestión del contexto y de las inferencias que en este se realizan. Esta circunstancia puede tener una influencia negativa tanto en la escalabilidad como en el mantenimiento de sistemas concretos donde el entorno sea extenso y complejo. Otro aspecto a destacar de SOPRANO es el modelo de descripción semántica de servicios que propone. En este caso se trata de un modelo propio, llamado DIANE, que, aunque inspirado en OWL-S, incluye mecanismos interesantes para el manejo de incertidumbre e información difusa en la descripción y descubrimiento de servicios. Este modelo potencialmente proporciona posibilidades de interoperación espontánea entre los servicios SOPRANO, aunque en la implementación realizada se reduce a una composición de servicios relativamente sencilla basada en la definición de plantillas de composición parametrizables. En lo referente al requisito de heterogeneidad, SOPRANO implementa un modelo de abstracción de dispositivos basado en servicios OSGi que encapsulan el acceso a los dispositivos físicos, similar al proporcionado por Amigo. No obstante, como en el caso de Amigo y PERSONA, este modelo no puede implementarse directamente en pequeños dispositivos con escasos recursos. Otras debilidades de la arquitectura SOPRANO son la ausencia total de mecanismos de movilidad, la ausencia de mecanismos que garanticen la privacidad y seguridad global del sistema y la ausencia de soluciones de interoperabilidad con componentes de otras plataformas

(únicamente soporta su propio modelo de gestión de servicios, DIANE).

Poniendo el punto de atención en el requisito *inteligencia social*, observamos que ninguna de las arquitecturas estudiadas lo contempla como un requisito independiente, y en general involucra aspectos poco abordados en los sistemas de IAm actuales. No obstante, este requisito está muy relacionado con aspectos como la capacidad de adaptación del sistema. la sensibilidad al contexto o la invisibilidad. Por tanto, aquellas arquitecturas que proporcionan mejores soluciones en estos ámbitos también proporcionan una mejor base para la inclusión de características relacionadas con lo que entendemos como inteligencia social de un sistema de IAm.

A modo de resumen, este análisis no hace sino enfatizar algunos de los motivos por los que ninguna de las arquitecturas de IAm existentes se ha conseguido establecer como una arquitectura de referencia en el área. Las soluciones estudiadas utilizan distintas aproximaciones conceptuales y distintas tecnologías, y además en algunos casos sitúan el foco en requisitos concretos de los entornos de IAm o directamente se centran en dominios restringidos. No obstante, todas comparten un éxito muy limitado en el cumplimiento de los requisitos impuestos para la creación de entornos inteligentes según la ambiciosa visión de la IAm. Así, aspectos como la interoperabilidad real entre sistemas, la movilidad de componentes o la gestión eficiente del contexto, constituyen líneas de investigación que todavía permanecen en su infancia. Además, la mayoría de estos proyectos no proporcionan suficientes facilidades para que otros desarrolladores los utilicen o los extiendan. De hecho pocos distribuyen libremente su código fuente y también son pocos los que publican documentación de diseño suficientemente detallada. Asimismo, en muchos casos, los proyectos parecen abandonados, sin ningún tipo de mantenimiento o continuidad. Una excepción es el proyecto Amigo, que, como se ha mencionado, mantiene accesible buena parte de su código fuente y una detallada documentación. Sin embargo, aún en este caso, nos encontramos con el hándicap de que su implementación mantiene dependencias de tecnologías actualmente desfasadas.

Además, una características común a la mayoría de estos proyectos es que las ambiciosas propuestas de sus modelos conceptuales iniciales se han visto notablemente reducidas a la hora de tomar decisiones de diseño y de elección de tecnologías concretas. En general, afectando principalmente a requisitos como la interoperabilidad, la interoperación espontánea y la escalabilidad.

## 3.5. Discusión

Como se ha puesto de manifiesto a lo largo de este capítulo, la Inteligencia Ambiental es un área de investigación enormemente prometedora que encierra una gran cantidad de desafíos. Así, aunque la visión de la IAm se introdujo hace más

de una década, el desarrollo e implantación de sistemas de IAm concretos continúa en sus infancia [Aarts and Ruyter, 2009]. Esto es debido fundamentalmente a la enorme distancia que todavía existe entre las ideas y conceptos que introduce la visión de la IAm y el estado de desarrollo de áreas tecnológicas involucradas.

Para crear entornos inteligentes como los visionados por la IAm, donde la tecnología se adapta a las necesidades, costumbres y deseos de las personas, y las asiste de forma no intrusiva con el fin de mejorar su calidad de vida, los sistemas tienen que enfrentarse a una serie de desafíos y problemáticas comunes. Este escenario da como resultado una serie de características y requisitos que, con carácter general, son aplicables a todas las soluciones de IAm. Algunas de las características más destacadas son heterogeneidad, sensibilidad al contexto, interacción natural, movilidad, invisibilidad, interoperabilidad, integridad, privacidad e inteligencia social. Intentar cumplir estos requisitos complica enormemente la ingeniería de soluciones de IAm.

Para abordar este escenario existen fundamentalmente dos caminos: el desarrollo de soluciones particulares ad-hoc que proporcionan funcionalidades concretas en un determinado entorno, y el desarrollo de modelos conceptuales y middleware que proporciona soluciones homogéneas y reutilizables a problemas comunes. Se ha visto que no existe un consenso absoluto respecto a la aproximación más conveniente a seguir. No obstante, después de realizar un extenso estudio de este ámbito, en este trabajo se defiende que las arquitecturas son el elemento de diseño fundamental para conseguir proporcionar características complejas, como las propias de los sistemas de IAm, de forma global a todo un sistema. Además, en sistemas software complejos, las arquitecturas software juegan un papel determinante a la hora de garantizar la calidad global de los mismos. Desafortunadamente, ninguno de los paradigmas existentes en ingeniería del software, incluidos los más avanzados como la Computación Orientada a Servicios o los Sistemas Multi-Agente, cubren por completo los requisitos propios de los sistemas de IAm. En consecuencia, se investigan distintas aproximaciones híbridas, que combinan aportaciones provenientes de los paradigmas clásicos y soluciones de otros campos como la Web Semántica o la Inteligencia Artificial. No obstante, a día de hoy no existe ninguna arquitectura de referencia comúnmente aceptada para sistemas de IAm.

Se ha realizado un estudio exhaustivo de un gran número de trabajos existentes en los ámbitos de la Inteligencia Ambiental, la Computación Ubicua, los Sistemas Pervasivos y el Internet de las Cosas que tratan de abordar la problemática inherente a este tipo de entornos partiendo de la construcción de arquitecturas y middleware de referencia, y se han identificados aquellos proyectos más relevantes e innovadores. De forma general, la base tecnológica de este tipo de soluciones es una arquitectura por capas distribuida en nodos computacionales de distinta complejidad y sofisticación. Típicamente, las capas inferiores se encargan de obtener información del entorno a través de una serie de sensores y de la modificación del entorno a tra-

vés de dispositivos actuadores. Las capas superiores normalmente se materializan en nodos que se encargan de procesar adecuadamente la información contextual y de la toma de decisiones. Además, los nodos de alto nivel pueden proporcionar distintos servicios de valor añadido que proporcionan funcionalidades avanzadas al usuario. Por último, con el fin de que los desarrolladores puedan centrarse en la programación de aplicaciones inteligentes que se adapten a las necesidades de los usuarios, es necesario una plataforma middleware que actúe a modo de unión entre las diferentes capas y componentes, proporcionando una serie de interfaces de programación y protocolos que oculten la heterogeneidad inherente a los sistemas y entornos de IAm.

Del análisis de las principales arquitecturas de IAm existentes se puede concluir, asimismo, que todas están lejos de proporcionar una solución "todo en uno" para el conjunto de características que definen a los sistemas de IAm. Más aún, aspectos como la interoperabilidad entre sistemas complejos, la representación y gestión de información contextual, la movilidad de componentes o la interacción natural constituyen líneas de investigación que permanecen abiertas a día de hoy. El acceso uniforme a la variedad de tecnologías y dispositivos que permiten sensorizar y actuar sobre el entorno tampoco está completamente resuelto en ninguna de las arquitecturas más relevantes, lo que restringe de forma importante la ubicuidad, invisibilidad y universalidad de las soluciones. Igualmente, otros aspectos importantes como la privacidad, seguridad y la tolerancia a fallos del sistema son a menudo obviados o abordados únicamente de forma marginal. Además, se ha detectado que una característica común a gran parte de los proyectos existentes en este ámbito es la ausencia de facilidades para su reutilización y extensión por parte del resto de la comunidad científica. Y esto es así debido a que en gran parte de los casos no se proporciona acceso libre a las implementaciones realizadas, falta documentación detallada de las mismas o los proyectos se encuentran por completo discontinuados y sin mantenimiento de ningún tipo.

Como se ha establecido previamente, todo lo dicho hasta aquí explica la situación actual en la que no existe ninguna arquitectura de IAm que pueda considerarse de referencia. Por tanto, este escenario deja abiertas muchas posibilidades de investigación en relación con la definición y construcción de modelos conceptuales y arquitecturas que permitan a los desarrolladores lidiar con la complejidad inherente a los entornos de IAm.

# Capítulo 4

# Arquitectura HI3

## 4.1. Introducción

Una vez realizado un análisis crítico del marco teórico en el que encaja el trabajo de esta tesis, en este capítulo se proporciona una visión global de la aproximación al proceso de desarrollo de sistemas de Inteligencia Ambiental (IAm) propuesta. Se trata de una aproximación integrada guiada por una arquitectura software que hemos denominado como arquitectura HI3. Así, en lo que sigue, se hará una introducción a los conceptos y elementos más relevantes de la arquitectura HI3, que incluyen principios de diseño, modelos conceptuales y herramientas software que organizan y facilitan el proceso de construcción de sistemas de IAm.

A lo largo de la revisión bibliográfica previa se ha destacado la interdisciplinaridad inherente al campo de la IAm y las aportaciones al mismo que se nutren de áreas como la Computación Orientada a Servicios (SOC), el Internet de las Cosas (IoT), la Inteligencia Artificial o la Interacción Persona-Computador. Asimismo, se ha puesto de manifiesto como en su momento se produjo una proliferación de propuestas para el diseño de arquitecturas software de IAm sin llegar ninguna a establecerse como una referencia en el campo. Así, este trabajo parte de las tendencias actuales de investigación en estas áreas afines a la IAm y de las lecciones aprendidas a través de la evaluación de soluciones middleware previamente diseñadas de forma explícita para este tipo de sistemas.

Por un lado, los avances en los últimos años en las áreas de IoT y SOC están contribuyendo a distribuir la computación hasta la mayoría de entornos en los que las personas realizan sus tareas y a que las empresas puedan hacer accesibles cada vez más procesos y funcionalidades a través de Internet [Miorandi et al., 2012] [Wu et al., 2011]. Todo esto apunta en el sentido de algunos de los principios de la

Computación Ubicua y la IAm. No obstante, la realidad actual es que hechos como la proliferación de middleware de IoT no interoperable, la escasa penetración de la Computación Orientada a Servicios lejos del ámbito de los Servicios Web o la casi nula repercusión de las tecnologías de Servicios Semánticos en entornos reales nos revelan que estas tecnologías están todavía lejos de ser la base que permita articular soluciones adecuadas a los ambiciosos requisitos establecidos para los sistemas de IAm. Asimismo, el auge de la Computación Móvil ha contribuido a acercar cada vez más servicios y funcionalidades al usuario en cualquier momento y lugar, a la vez que ha generado nuevas problemáticas no resueltas [Baresi et al., 2012].

Por otro lado, revisando la bibliografía se ha comprobado como el auge de aproximaciones basadas en arquitecturas de referencia para IAm ha sufrido un parón en favor de esfuerzos más concretos en campos como los mencionados previamente. Creemos que este particular se debe en buena parte a la enorme dificultad de abordar los ambiciosos objetivos de la IAm desde un punto de vista global y al grado de interdisciplinaridad que ello requiere, con el agravante del nivel de inmadurez en que se encuentran muchas de las áreas involucradas. Además, gran parte de las arquitecturas diseñadas previamente se han visto sobrepasadas en pocos años por los avances producidos en los sistemas de información y telecomunicaciones en general, al probarse actualmente como desarrollos difícilmente trasladables a entornos de computación móvil, dispositivos *wearables* o dispositivos de IoT.

Partiendo de este análisis, el presente trabajo propone dar un pequeño paso atrás para re-evaluar las posibilidades de plantear una visión holística de los sistemas de IAm desde el prisma de la ingeniería de arquitecturas software y partiendo de un núcleo de requisitos fundamentales más acotado y fácilmente abordable. Para ello, se propone partir de los avances más recientes en áreas como Computación Orientada a Servicios (SOC), Internet de las Cosas (IoT) o Computación Móvil, integrándolos bajo un mismo paradigma y dotándolos de características que los acerquen un poco más a los principios de la Computación Ubicua y la IAm. En particular, se perseguirá la definición de una arquitectura software capaz de expandirse a la mayor parte de plataformas computacionales existentes hoy en día y con posibilidades de ser extendida progresivamente con nuevas capacidades.

Este trabajo está enmarcado en el contexto de un proyecto de investigación multidisciplinar del Grupo Integrado de Ingeniería de la Universidade da Coruña denominado Proyecto HI3. Se trata de un proyecto íntimamente relacionado con líneas de investigación como la IAm, la Computación Ubicua, las Interfaces Naturales de Usuario o la Inteligencia Artificial. Su objetivo central es la investigación de tecnologías que permitan la creación de espacios en los que la computación se integre de forma transparente en el día a día de las personas, de forma que los usuarios perciban el servicio sin ser conscientes de como éste se lleva a cabo. Más concretamente, el Proyecto HI3 persigue la creación de entornos Humanizados, Inteligentes, Interactivos e Integrados. En un entorno Humanizado las personas no

necesitan un entrenamiento especial para hacer uso de las tecnologías, éstas son transparentes para el usuario y son capaces de adaptarse a sus características. Un entorno Inteligente exhibe un comportamiento proactivo, adaptable, predictivo y autónomo. Por último, estos entornos deben posibilitar la Integración transparente de diferentes tecnologías, incluyendo tecnologías de sensorización/actuación, redes de comunicaciones o plataformas computacionales con recursos heterogéneos.

Dentro del contexto del Proyecto HI3 surge la oportunidad y la necesidad de integrar trabajos de investigación en áreas transversales a la IAm, como las mencionadas previamente, bajo una visión común. Como eje central de esta aproximación holística, se hace inmediatamente patente la conveniencia de articular un modelo conceptual y una arquitectura software que facilite la construcción progresiva de entornos de IAm cada vez más amplios y ambiciosos sin necesidad de enfrentar repetidamente los mismos problemas. Este trabajo se corresponde con dicha vertiente del Proyecto HI3, y pretende, por tanto, avanzar en la construcción de las herramientas y métodos necesarios para facilitar la integración de soluciones multidisciplinares a requisitos concretos de IAm en un mismo sistema, haciéndolas, para ello, accesibles a través de un método común. Más concretamente, se propondrá una arquitectura software, denominada arquitectura HI3, inspirada en los principios de las Arquitecturas Orientadas a Servicios (SOA) [Arsanjani et al., 2009] y adaptada a las necesidades de los entornos de Computación Ubicua.

En los siguientes subapartados de este capítulo se presentará el método seguido para la articulación de la arquitectura HI3 como herramienta vertebradora de una visión holística para el desarrollo de sistemas de IAm. Se describirá el análisis previo junto con el modelo conceptual y la metodología, que en su conjunto establecerán las bases de diseño de la arquitectura HI3. Asimismo, a lo largo del capítulo se introducirá una visión global de los componentes principales de la arquitectura.

## 4.2. Modelo conceptual de referencia

Esta tesis defiende la premisa de que una buena aproximación para afrontar al desarrollo de sistemas complejos es la construcción de modelos conceptuales y middleware que proporcionen a los desarrolladores una serie de patrones de diseño y herramientas con las que resolver problemas que son comunes a un determinado tipo de entorno. En oposición, otros estudios y trabajos en el campo de IAm se focalizan únicamente en objetivos funcionales que son abordados con sistemas ad-hoc. Esta última aproximación a menudo conduce a la construcción de sistemas aislados y la imposibilidad de reutilización de resultados, pero se justifica claramente ante la ausencia de un middleware de referencia en el área. Por otro lado, en lo referente a la primera aproximación, durante la realización de este trabajo también se han identificado dos problemas muy relevantes relacionados con la proliferación

de middleware en el área de la IAm y en áreas relacionadas: i) la **heterogeneidad de middleware** que da como resultado infraestructuras no interoperables; ii) una **excesiva complejidad y rigidez del middleware** que en ocasiones hace que no sean evidentes sus ventajas frente al desarrollo de sistemas ad-hoc.

Como hemos visto, algunas de las infraestructuras middleware existentes lidian razonablemente bien con aspectos relacionados con la distribución de la funcionalidad, con la gestión de componentes o con la configuración dinámica del sistema. No obstante, estas infraestructuras lidian más pobremente con la heterogeneidad del entorno, a menudo delegando en el uso de soluciones propietarias o protocolos concretos. De forma similar, estas infraestructuras tienen problemas relacionados con las capacidades de interoperabilidad de sus componentes funcionales con componentes externos. Esta problemática se traduce, en la práctica, en la construcción de sistemas aislados incapaces de colaborar.

Por otro lado, durante la realización de este trabajo, también se ha detectado que la utilización de una infraestructura middleware existente para el desarrollo de aplicaciones de IAm puede desviar en exceso los esfuerzos de los desarrolladores de la implementación de funcionalidades. Este escenario se presenta normalmente al utilizar una infraestructura middleware muy compleja, excesivamente rígida y restrictiva o difícilmente extensible a diferentes entornos de ejecución.

Ambos problemas se hacen muy notables en el campo de la IAm, en parte por sus ambiciosos objetivos y en parte por el estado de inmadurez en el desarrollo del área en general y en el desarrollo de infraestructuras middleware en particular. Es por ello que, como punto de partida para el desarrollo de la arquitectura software correspondiente a este trabajo se ha establecido un núcleo de premisas y prioridades, con el objetivo de paliar la problemática anteriormente descrita:

- **Identificación de un subconjunto de características** de entre las propias de los sistemas de IAm que constituyen los retos que inevitablemente han de ser abordados en la implementación de nuevas funcionalidades. Esto implica poner el foco en un grupo acotado de características con el fin de mantener las soluciones simples, homogéneas y bien enfocadas.
- Definición de una **arquitectura conceptual**, en consonancia con las características identificadas en el punto anterior, que guiará la construcción de sistemas de IAm.
- La **simplicidad** debe ser uno de los principios clave que guíen el diseño. El grado de complejidad a la hora de implementar un nuevo componente funcional sobre la infraestructura middleware deberá depender en mayor medida de los requisitos de la propia funcionalidad que de los requisitos de implementación que imponga la infraestructura.
- La **flexibilidad** como otro de los principios fundamentales de diseño. La in-

fraestructura debe ser lo suficientemente flexible como para permitir al desarrollador elegir cuáles de sus características utiliza en la construcción de un componente funcional. Asimismo, la infraestructura debe estar abierta a la ejecución de componentes del sistema en entornos computacionales muy diferentes o a la utilización de diferentes lenguajes de programación y protocolos de comunicación. Es decir, debe adaptarse a los escenarios incipientes de la Computación Ubicua, la Computación Móvil o el IoT.

- Fomentar la **interoperabilidad** real entre sistemas junto con el descubrimiento espontáneo de nuevos servicios presentes en el entorno.
- Mantener el foco en facilitar el desarrollo de nuevas **funcionalidades**. Facilitando la adaptación de su comportamiento al contexto en base al descubrimiento de servicios en el entorno.
- Mantener al **usuario** como un elemento central del sistema. Esto significa que, al igual que los componentes funcionales deben adaptarse al entorno, también deben adaptarse a las características y necesidades de los usuarios.

Partiendo de las anteriores premisas, y como paso previo a la definición del modelo conceptual propuesto en este trabajo, se ha realizado un estudio de los aspectos clave que, de forma general, deben ser abordados para la construcción de sistemas de IAm. En la Figura 4.1 se esquematizan los requisitos centrales que se han establecido para el diseño de la arquitectura HI3. Como puede observarse en el diagrama, en el centro de toda solución de IAm debe situarse a los usuarios y los entornos físicos en los que las personas realizan sus actividades. Situarlos tanto en el centro del problema y como en el centro de la solución significa establecer los requisitos en torno a ellos y diseñar las soluciones de modo que integren sus características y respondan a sus necesidades.

Alrededor de los usuarios y el entorno físico se han establecido cuatro requisitos clave. La reducción a un núcleo compacto de requisitos principales se ha hecho con el fin de conducir el diseño hacia objetivos bien definidos para así evitar el problema de la excesiva complejidad del middleware. A continuación se realiza una breve descripción de estos requisitos.

**Requisito 1. Adaptación al entorno físico.** Para que las aplicaciones de IAm puedan adaptar inteligentemente su comportamiento a las necesidades de las personas y a los cambios en el entorno es necesario poblar los entornos físicos de dispositivos capaces de monitorizar su estado y actuar sobre él. Desafortunadamente, la fragmentación tecnológica existente en el mercado para este tipo de dispositivos y tecnologías es enorme. Por tanto, aunque es una noticia positiva que en los últimos años se haya producido un crecimiento exponencial en la cantidad y capacidad de dispositivos digitales integrables en diferentes entornos, también es un reto importante abordar la heterogeneidad tecnológica inherente a los mismos.

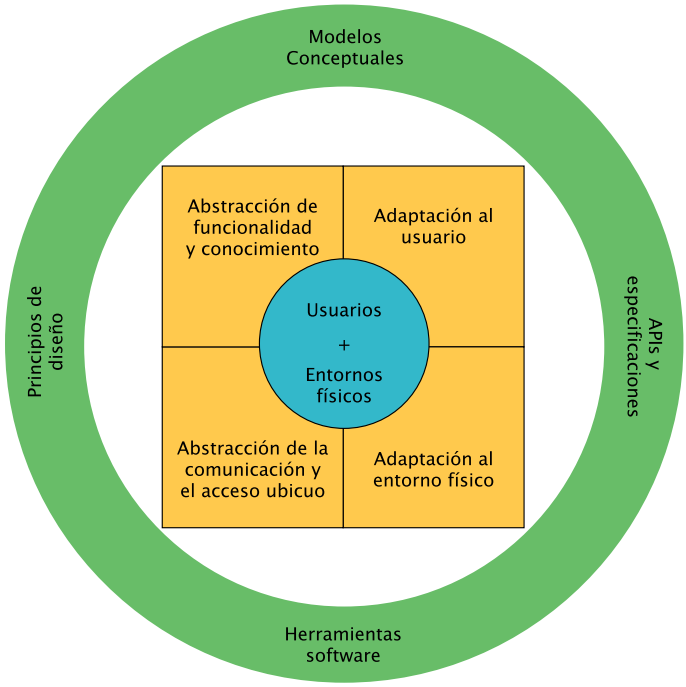



Figura 4.1: Requisitos centrales de la arquitectura HI3.

**Requisito 2. Adaptación al usuario.** Para que los sistemas de IAm sean aceptados por los usuarios deben adaptarse a sus características, preferencias y necesidades. No obstante, prever de antemano todos los posibles escenarios de ejecución en aquellos aspectos relacionados con los usuarios es enormemente complicado en entornos tan amplios y abiertos como son los de IAm. Así, para evitar la necesidad de desarrollar aplicaciones ad-hoc para cada escenario, las funcionalidades deberían poder ser desarrolladas con independencia de usuarios concretos, proporcionando mecanismos que permitan su adaptación en tiempo de ejecución a los usuarios y su contexto. En especial, las aplicaciones deben ser capaces de proporcionar este tipo de adaptación en lo referente a los modos de interacción con el usuario.

**Requisito 3. Abstracción de la funcionalidad y el conocimiento.** Es necesario abstraer el acceso a las funcionalidades para construir aplicaciones capaces de adaptar dinámicamente su comportamiento a través de la utilización espontánea de funcionalidades e información proporcionados por otros sistemas del entorno. Es de esa colaboración no prefijada entre sistemas de dónde pueden surgir muchas de las funcionalidades más atractivas para los usuarios, que de forma natural transitan por distintos entornos y contextos.

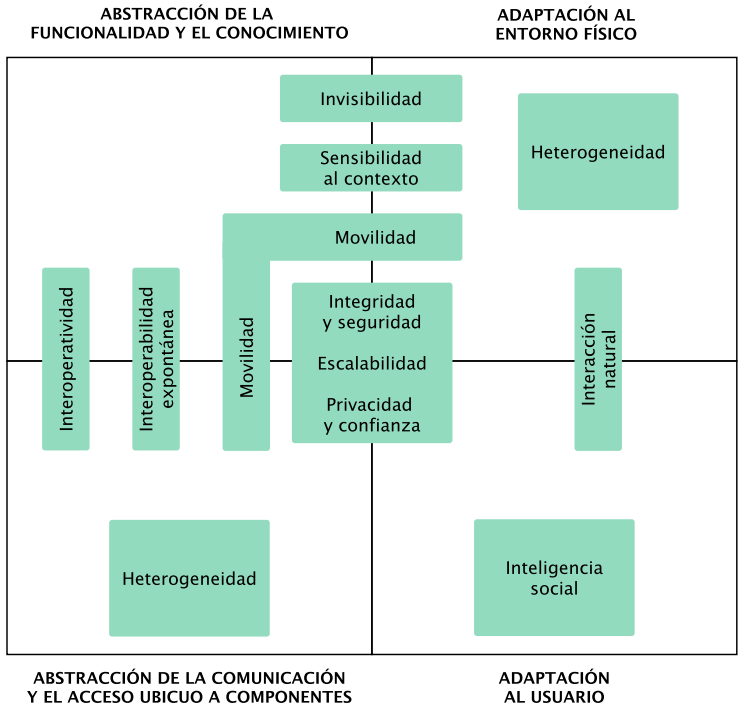



Figura 4.2: Requisitos de los sistemas de IAm.

**Requisito 4. Abstracción de la comunicación y acceso ubicuo.** Este requisito está íntimamente relacionado con el anterior. De nada sirve que un sistema sea potencialmente capaz de entender la funcionalidad y el conocimiento de otro si no es capaz de comunicarse con él y acceder a su funcionalidad. Es necesario por tanto abstraer los aspectos relacionados con la comunicación y el acceso a funcionalidades de terceros. Buscando la simplicidad en la construcción de sistemas, sería asimismo deseable que los sistemas exportasen las funcionalidades susceptibles de ser utilizadas por otros a través de un único componente conceptual.

Este grupo de requisitos se reflejarán claramente en el diseño de la la arquitectura HI3 que se introducirá más adelante. No obstante, es necesario tener presente que este núcleo de requisitos se relaciona, en mayor o menor medida, con la lista más detallada de requisitos para sistemas de IAm que se estableció en el capítulo anterior. En la Figura 4.2 se ilustra dicha relación.

Como se esquematiza en la Figura 4.1, en torno al núcleo de requisitos clave se construirán una serie de modelos conceptuales comunes, se establecerán unos principios de diseño, se definirán APIs y especificaciones software, y se desarrollarán una serie de herramientas software, con el fin de construir un entorno homogé-

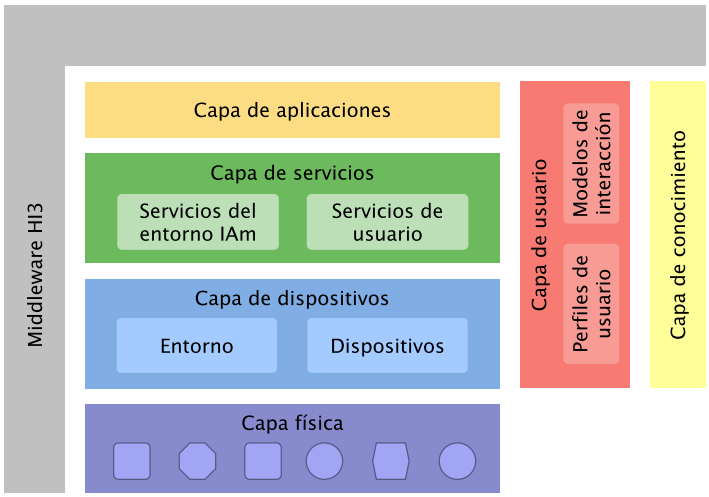



Figura 4.3: Capas del modelo conceptual HI3.

neo de desarrollo de sistemas de IAm. Todos estos elementos se presentan bajo el paraguas de la arquitectura software HI3 descrita en este trabajo.

De acuerdo con las características generales de los sistemas de IAm estudiadas en capítulos previos y con las características clave que se han identificado para la arquitectura HI3, se propone un modelo conceptual para describir cualquier sistema de IAm basado en siete capas (ver Figura 4.3). Un diseño basado en capas permite organizar conceptualmente los componentes del sistema, favoreciendo la abstracción y el reparto de funcionalidades. A continuación se describe cada una de estas capas conceptuales de la arquitectura HI3.

**Capa Física.** Esta capa está compuesta por todos aquellos dispositivos (sensores, actuadores, dispositivos multimedia, etc.) con los que se puebla un entorno de IAm. A través de ellos los sistemas pueden monitorizar y controlar el entorno e interaccionar con los usuarios. De forma general estos dispositivos estarán construidos en base a tecnologías heterogéneas y utilizarán diferentes redes de comunicación.

**Capa de Dispositivos.** Con el fin de abordar la heterogeneidad tecnológica inherente a estos entornos es necesario el desarrollo de middleware que independice a las aplicaciones de variaciones en el entorno con el que se relacionan. Para ello, esta capa es la responsable de establecer un modelo universal de representación del entorno y de los dispositivos, extensible para incorporar otras tecnologías y exportado a través de un lenguaje común que permita la interoperabilidad de sistemas.

**Capa de Servicios.** Esta capa, junto con la Capa de Aplicaciones son las encargadas de resolver las funcionalidades concretas que proveerá un sistema de IAm, de

acuerdo con los requisitos 3 y 4 planteados. Los servicios proporcionarán funcionalidades que puedan ser accedidas y reutilizadas por otros componentes. Por tanto estos deben fomentar la interoperabilidad, tanto a nivel sintáctico como semántico, para favorecer la colaboración en la resolución de tareas. Además, deben posibilitar que dicha colaboración sea espontánea entre servicios a priori desconocidos.

**Capa de Aplicaciones.** Es la capa de más alto nivel y la que alberga elementos que representan aplicaciones que resuelven funcionalidades concretas que algún usuario espera del sistema. Típicamente, estas aplicaciones harán uso de servicios presentes en el entorno para realizar su tarea. Para ello utilizarán los mismos mecanismos de interoperabilidad que los servicios. La posibilidad de desarrollar aplicaciones que resuelvan su funcionalidad apoyándose en los servicios más adecuados que en un momento dado descubran en su entorno es una de las claves para construir sistemas capaces de adaptar su comportamiento al contexto.

**Capa de Conocimiento.** Esta capa representa, a nivel conceptual, el conocimiento común del dominio que deben tener los componentes funcionales de la arquitectura. Este conocimiento se representará en esquemas conceptuales y lenguajes comunes (ontologías) con la finalidad de facilitar la comunicación y el intercambio de información entre diferentes componentes funcionales.

**Capa de Usuario.** Se trata de una capa transversal que relaciona al usuario con la funcionalidad y las características del entorno. Su objetivo principal es mantener al usuario en el centro de los sistemas de IAm. Para ello gestionará perfiles de usuario que permitan adaptar el comportamiento de aplicaciones y servicios a las particularidades de estos. Asimismo, facilitará la interacción natural adaptada a las características de cada usuario.

**Capa de Middleware.** Se trata de la capa en base a la que se construyen los componentes de las demás capas de la arquitectura. Su objetivo es proporcionar una serie de frameworks y herramientas software que faciliten a los desarrolladores la construcción de sistemas de IAm que cumplan los requisitos planteados y estructurados según el modelo conceptual propuesto por la arquitectura HI3.

El modelo conceptual de referencia presentado hasta ahora proporciona únicamente una descripción lógica de la arquitectura, identificando los conceptos principales y el vocabulario con independencia de tecnologías, implementaciones y detalles más concretos de diseño. Dicho modelo podría, por ejemplo, ser realizado de forma que la implementación de sus elementos se encuentre totalmente distribuida. Este tipo de consideraciones corresponden al diseño de la arquitectura software HI3, cuyo detalle se introducirá en los apartados siguientes.

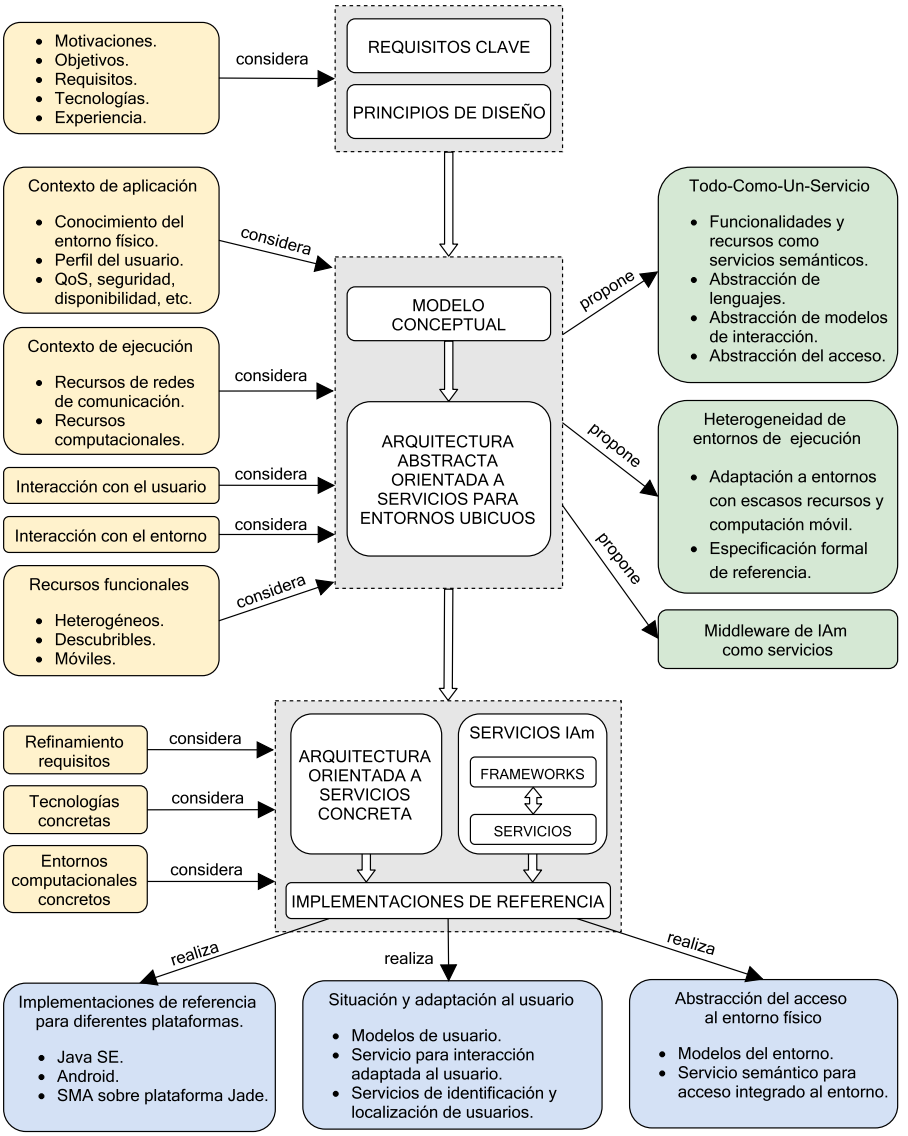



Figura 4.4: Visión global del proceso de construcción de la arquitectura HI3.

## 4.3. Método

El análisis presentado en el apartado anterior constituye una referencia conceptual para la arquitectura HI3 al margen de realizaciones concretas de la misma y de la inevitable evolución en tecnologías concretas. Este marco conceptual es la base para la especificación de una *arquitectura abstracta* independiente de un

dominio de problema, pero en la que se toman decisiones de diseño relacionadas con el paradigma de desarrollo software o la organización y estructura básica de los componentes principales de la arquitectura. En una fase posterior podrá definirse una *arquitectura concreta* en base a la selección de tecnologías particulares y a la especificación detallada de soluciones más concretas. Finalmente, podrán realizarse *implementaciones de referencia* de la arquitectura concreta para entornos computacionales determinados. De forma simplificada, éste es el método de trabajo propuesto en esta tesis para guiar la construcción de la arquitectura HI3, y queda esquematizado en la Figura 4.4.

Por tanto, tras las fases ya descritas de análisis y conceptualización, guiadas por los objetivos, motivación y requisitos que definen el problema a resolver, se vuelve a considerar con mayor detalle el contexto del problema con el fin de diseñar la arquitectura abstracta. En la Figura 4.4 se enumeran los principales aspectos considerados para definir la arquitectura abstracta HI3 junto con las principales propuestas en que se basa su diseño. Comenzando por los aspectos a considerar, cabe destacar la necesidad de incorporar al sistema el mayor conocimiento posible referente al contexto en el que realiza su tarea una aplicación de IAm (contexto de aplicación). Éste puede incluir información referente al estado del entorno físico, información del perfil de los usuarios o información relacionada con aspectos no-funcionales que puede determinar la calidad de servicio, seguridad, etc. de una aplicación. Asimismo, es importante considerar los posibles contextos de ejecución de una aplicación de IAm. En este caso, idealmente es deseable que la ejecución de aplicaciones se extienda a todas las plataformas computacionales y redes de comunicación existentes de forma transparente, a través de las abstracciones de tecnologías necesarias. Además, como ya se ha resaltado previamente, la arquitectura debe considerar formas de abstraer la descripción y acceso a los recursos funcionales fomentando la interoperabilidad dinámica entre sistemas.

En base a las consideraciones previas y al modelo conceptual de referencia, se diseña una arquitectura abstracta HI3 que se asienta en tres pilares fundamentales: i) un núcleo de arquitectura SOA construida en torno a un Middleware Orientado a Servicios (SOM) enfocado a las necesidades de la Computación Ubicua; ii) abstracción del propio middleware de IAm a través de servicios semánticos y ubicuos; iii) una arquitectura de referencia adaptable a multitud de entornos de ejecución.

**Paradigma de Computación Orientada a Servicios adaptado a entornos ubicuos.** Como se analizó en la revisión bibliográfica, la evolución de Internet y de la Computación Móvil presentan importantes desafíos para el paradigma SOC que afectan a todas sus capas arquitecturales [Baresi et al., 2012]. Así, la arquitectura HI3 asienta sus bases en los principios de las arquitecturas SOA, buscando una mejor adecuación del paradigma a los requisitos de la Computación Ubicua. De modo especial, nos centraremos en el problema de la enorme heterogeneidad de

modelos, lenguajes, protocolos y tecnologías existentes, que demanda un profundo conocimiento de las tecnologías middleware por parte de los desarrolladores, convirtiendo el desarrollo de aplicaciones ubicuas en una tarea enormemente costosa en tiempo.

**Middleware de IAm como servicios semánticos.** En una primera fase de estudio de este trabajo, se consideró una aproximación para el diseño de la arquitectura HI3 en la que cada una de las distintas capas del modelo conceptual de referencia (ver Figura 4.3) se realizase a través de elementos constructivos claramente diferenciados. Sin embargo, los primeros experimentos y análisis realizados revelaron que un exceso de complejidad y heterogeneidad de modelos en la arquitectura podría complicar el desarrollo de nuevos sistemas a la vez que aumentaría la rigidez de la arquitectura resultante. Diferentes modelos y entidades en cada capa, diferentes APIs, diferentes formas de representar la información o la necesidad de adaptarla de una capa a otra, pueden en ocasiones ayudar a estructurar un problema complejo pero también pueden complicar su resolución. Esta problemática se hace especialmente patente cuando para la construcción de sistemas simples es necesario realizar laboriosos desarrollos con el fin de encajarlos dentro de una arquitectura. De hecho, consideramos que este es uno de los principales inconvenientes de buena parte de las arquitecturas de IAm planteadas hasta ahora. Con el fin de evitar esta problemática, se decide abstraer como servicios no sólo las nuevas funcionalidades sino también el propio middleware de IAm. De modo que lo que realmente convierte la arquitectura SOA del punto anterior en una arquitectura de IAm es que la propia arquitectura provee un conjunto de servicios semánticos que proporcionan acceso uniforme y ubicuo al middleware con el que se resuelven las problemáticas comunes de los sistemas de IAm. Este nuevo nivel de abstracción, facilitará además la convivencia, integración o reemplazo de diferentes soluciones middleware para un mismo requisito, ayudando a combatir el problema de la heterogeneidad del middleware.

**Soporte a la heterogeneidad de entornos de ejecución.** La situación actual y de la última década de desarrollo de tecnologías en el ámbito SOC se caracteriza porque la mayor parte de las soluciones existentes en el área están orientadas a las tecnologías y necesidades de la Web. Por tanto, la arquitectura HI3 también se centrará en proponer soluciones en un nivel superior de abstracción, tratando de integrar tecnologías y modelos de operación propios de cualquier entorno computarizado en el que las personas realicen su actividad (Web, dispositivos de Computación Móvil, Ciudades Inteligentes, Hogar Digital, etc.). En particular, se perseguirá que la arquitectura permita llevar la ejecución de los componentes funcionales a múltiples plataformas computacionales, permitiendo crear una red dinámica de servicios y aplicaciones de IAm en los diferentes entornos del usuario.

Estos tres pilares de la arquitectura abstracta HI3 deben ayudar a que los desarrolladores puedan centrarse más en construir nuevas funcionalidades y menos en lidiar con las particularidades del middleware y con un conglomerado de tecnologías heterogéneas.

Si miramos de nuevo a la Figura 4.4, el siguiente paso en el proceso de construcción de la arquitectura HI3 es la definición de la arquitectura concreta con un mayor detalle de diseño y en base a una selección más concreta de tecnologías. Así, se propone un diseño enfocado a dar respuesta a los requisitos prioritarios identificados en la primara fase del proceso, manteniendo el objetivo de que la arquitectura pueda incorporar nuevos requisitos y ampliar sus capacidades en el futuro. Por un lado un diseño de un middleware SOA basado en servicios semánticos proporcionará las principales herramientas para resolver los requisitos *abstracción de la funcionalidad y el conocimiento* y *abstracción de la comunicación y acceso ubicuo*. Por otro lado, se dotará a la arquitectura HI3 de un conjunto de soluciones para los requisitos *adaptación al entorno físico* y *adaptación al usuario*, que en último término serán abstraídas como servicios semánticos gracias a las capacidades del middleware SOA.

Finalmente, también se proponen una serie de implementaciones de referencia de la arquitectura basadas en plataformas software concretas. Incluyendo plataformas basadas en la Máquina Virtual Java y en la plataforma móvil Android.

A continuación, en la sección 4.4 del capítulo, se describen en mayor detalle los conceptos principales de la arquitectura abstracta HI3. En las secciones posteriores, 4.5 y 4.6, se da una visión global del diseño de la arquitectura concreta HI3, tanto desde la perspectiva de un middleware genérico de servicios semánticos e interoperables con capacidad de adaptarse a los requisitos de entornos ubicuos, como desde la perspectiva de una arquitectura que proporciona soluciones más concretas para la construcción de entornos de IAm.

## 4.4. Arquitectura abstracta

Hemos visto que, en el nivel superior de abstracción de la arquitectura HI3, se propone un tipo de componente principal (servicio) a través del que resolver y exportar las funcionalidades y capacidades de la arquitectura. La construcción de nuevas funcionalidades queda focalizada en el desarrollo de servicios reutilizables, interoperables, colaborativos y ubicuos, con independencia de las tecnologías concretas con las que internamente cada desarrollador implemente su comportamiento. En consecuencia, las capas de la arquitectura que resuelven requisitos típicos de IAm pasarán a tener también su representación en la Capa de Servicios, haciendo accesible la funcionalidad de la capa de forma ubicua y homogénea.

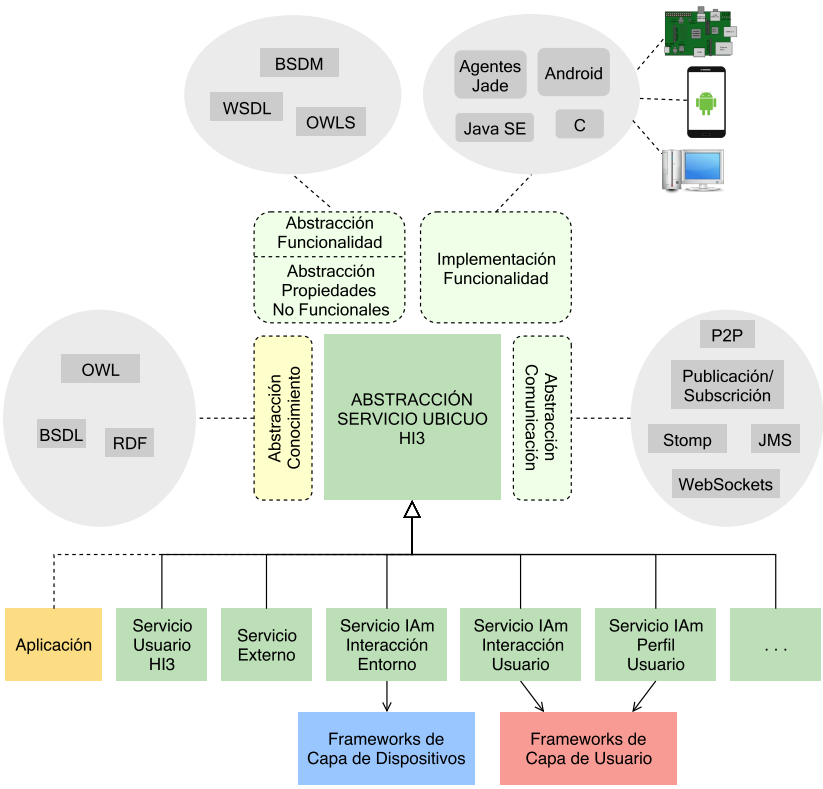



Figura 4.5: Principales elementos de la arquitectura abstracta HI3.

La Figura 4.5 ilustra esta visión abstracta de la arquitectura HI3 en torno a una perspectiva de tipo *todo-como-un-servicio*. De esta forma, la funcionalidad del sistema se implementa a través de aplicaciones y servicios distribuidos en los diferentes entornos en los que los usuarios realizan sus actividades. Estos componentes serán capaces de encontrarse y colaborar dinámicamente en base a sus necesidades funcionales. Para ello, los servicios se construirán con la combinación de cuatro elementos principales: descripción semántica de sus propiedades funcionales/no-funcionales, implementación de la funcionalidad, conocimiento y comunicación.

- **Descripción semántica.** Todo servicio debe proporcionar un descriptor semántico de sus capacidades funcionales, con el fin de que servicios de terceros puedan descubrirlas e interaccionar con él. Asimismo, este descriptor debe tener la capacidad de incluir propiedades no-funcionales (relacionadas con calidad de servicio, seguridad, privacidad, disponibilidad, etc.) que pueden ser relevantes para la selección de uno u otro servicio del entorno. La arquitectura HI3 propondrá un modelo propio de descripción de servicios adaptado a las características de los entornos ubicuos. Este modelo permiti-

rá, a su vez, la integración de otras tecnologías de este tipo, facilitando así la interoperabilidad entre servicios descritos con diferentes modelos.

- **Implementación de la funcionalidad.** Se trata de la implementación concreta que resuelve la funcionalidad que ofrece un servicio, y que éste hace visible a través de su descripción semántica.
- **Conocimiento.** Los servicios deberán exportar aquel conocimiento del dominio que sea necesario para interaccionar con ellos a través de un lenguaje de ontologías neutral (las ontologías proporcionan herramientas adecuadas para describir de forma extensiva la semántica de los conceptos y las relaciones entre ellos [Antoniou and Van Harmelen, 2004] [Euzenat and Shvaiko, 2013]). De nuevo, persiguiendo la interoperabilidad entre sistemas, la arquitectura HI3 tampoco está restringida a la utilización de un único lenguaje para la descripción del conocimiento.
- **Comunicación.** Todo servicio debe proporcionar algún mecanismo a través del que pueda comunicarse remotamente con otros componentes. El middleware de la arquitectura HI3 es responsable de abstraer esta comunicación, de modo que componentes desplegados en distintas plataformas software/hardware e incluso programados en distintos lenguajes, puedan interaccionar entre sí. Con el fin de proporcionar un paradigma de comunicación más rico que el modelo habitual en el ámbito de los servicios semánticos (petición/respuesta), la arquitectura HI3 proporcionará un modelo abstracto con soporte para distintos patrones de interacción (como petición/respuesta o publicación/-subscripción). Este modelo será implementable sobre distintos protocolos de comunicación existentes, pudiendo cohabitar varios de forma transparente.

En el esquema correspondiente a la arquitectura abstracta de la Figura 4.5 también podemos observar como diferentes componentes de la arquitectura conceptual son realizados a través del componente Servicio. Un caso especial es el de las Aplicaciones, que, siendo un componente diferente a los servicios, comparte parte de las características de éstos. Así, las aplicaciones, para adaptar su comportamiento, han de ser capaces de descubrir servicios en el entorno que ofrezcan una funcionalidad que necesitan y deben ser capaces de interaccionar con ellos compartiendo conocimiento del dominio descrito en algún lenguaje neutral.

Además, como ya se ha mencionado, lo que convertirá a esta arquitectura SOA para entornos ubicuos en una arquitectura de IAm es que proporcione un conjunto de servicios que resuelvan requisitos propios de estos entornos. En consecuencia, como parte de la arquitectura se propone un conjunto de frameworks y servicios que resuelven la problemática correspondiente a las capas identificadas en el modelo conceptual como Capa de Usuario y a la Capa de Dispositivos.

Por último, y aunque no aparece reflejado en el diagrama de la Figura 4.5, un componente importante en toda arquitectura SOA es el Registro de Servicios. En el

caso de la arquitectura HI3 se trata de un componente en el que los servicios podrán registrar sus descriptores semánticos, de modo que otros servicios o aplicaciones puedan encontrar en cada entorno las funcionalidades que necesitan. Pensando en la escalabilidad de la solución, la arquitectura propone un sistema de registros distribuidos con capacidad para federarse entre ellos. Asimismo, para fomentar la interoperabilidad, los registros deben ser capaces de operar simultáneamente con descriptores correspondientes a diferentes modelos de servicios.

Resumiendo, para dar soporte a este conjunto de características, la arquitectura HI3 proporcionará un middleware estructurado en dos grandes bloques:

- **Middleware orientado a servicios para entornos ubicuos.** Encargado de soportar las bases de la arquitectura SOA y la abstracción del componente Servicio con las características descritas previamente. Proporciona capacidades para las siguientes capas conceptuales: Capa de Servicios, Capa de Aplicaciones y Capa de Conocimiento. Se organizará en dos frameworks principales:
  - *Framework Broccoli.* Es el responsable de proporcionar un modelo "universal" y los lenguajes necesarios para la descripción de servicios semánticos interoperables adaptados a entornos de Computación Ubicua. Asimismo, es utilizado por las aplicaciones para descubrir servicios e interaccionar con ellos.
  - *Framework Vineyard.* Su función principal es proporcionar contenedores para la ejecución y gestión del ciclo de vida de aplicaciones y servicios HI3 en una determinada plataforma computacional.
- **Middleware de IAm.** Proporciona soporte para requisitos de más alto nivel comunes a la mayoría de sistemas de IAm. Dentro del alcance de este trabajo proponemos varios frameworks que aportan soluciones correspondientes a la Capa de Usuario y a la Capa de Dispositivos. No obstante, la arquitectura queda abierta a su extensión con nuevas capacidades, con el objetivo de crear entornos cada vez más cercanos a lo visionado en los principios de la IAm.

En las secciones siguientes (4.5 y 4.6) se hace una introducción a la realización concreta de la arquitectura HI3 a través de estos dos bloques de middleware.

## 4.5. Arquitectura concreta orientada a servicios para entornos ubicuos

Esta sección proporciona una visión de alto nivel del diseño correspondiente al middleware orientado a servicios que da soporte a la arquitectura HI3, resaltando aquellos elementos que nos permitirán abordar algunos de los desafíos que los entornos de Computación Ubicua e IAm imponen al paradigma SOC.

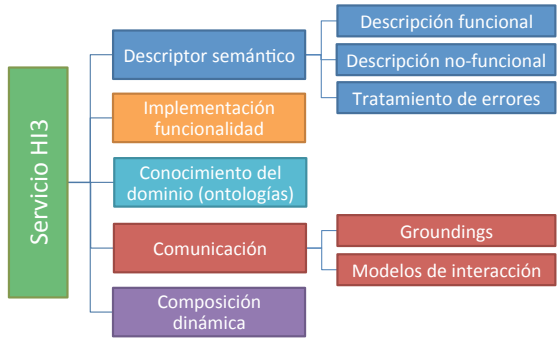



Figura 4.6: Principales elementos del modelo de descripción de servicios.

En los ámbitos de la Computación Ubicua y la IAm, la estandarización es una de las claves principales. Los objetivos más ambiciosos de la IAm únicamente son posibles cuando los dispositivos y los componentes funcionales son capaces de hablar entre ellos, y actualmente no existe una forma realista de que eso ocurra plenamente. Por ello, además de proponer modelos generales es necesario integrar bajo un mismo paraguas otras tecnologías relevantes existentes. Éste es uno de los principios de la visión holística propuesta en este trabajo y a través de las capacidades propuestas para el middleware orientado a servicios de la arquitectura HI3 trataremos de avanzar en este sentido.

Recordemos que, en la arquitectura HI3, los servicios y aplicaciones constituyen los principales componentes funcionales en base a los que se construyen los sistemas de IAm y que estos comparten buena parte de sus características. En la Figura 4.6 se representa con mayor detalle la estructura de un servicio HI3 construido según el modelo proporcionado por el framework Broccoli y que hemos denominado como BSDM (del inglés, Broccoli Service Description Model). Principalmente está compuesto por los siguientes elementos, que describiremos a continuación: descriptor semántico, implementación, conocimiento, comunicación y estrategias de composición.

**Descriptor semántico.** Como se especificó en la definición de la arquitectura abstracta, es imprescindible que componentes funcionales heterogéneos sean capaces de hablar entre ellos, sin la existencia de un conocimiento mutuo de sus capacidades y características predefinido. En consecuencia, es necesario que existan unos conceptos comunes con los que representar todas las propiedades de un servicio. Además de la descripción de las capacidades funcionales, el modelo BSDM pondrá énfasis en otras características más raramente soportadas en modelos de servicios existentes, como la propiedades no-funcionales y la descripción de condiciones de error. Estas últimas pueden facilitar la construcción de mecanismos automáticos de recuperación ante errores.

**Implementación de la funcionalidad.** Implementación concreta de la funcionalidad descrita en el modelo BSDM. En este modelo se contempla que un servicio pueda ofrecer más de una funcionalidad.

**Conocimiento del dominio.** Otro de los puntos clave para que dos servicios puedan hablar entre ellos espontáneamente es que compartan algún conocimiento del dominio descrito en un lenguaje neutral. Por tanto, de forma general, el descriptor semántico de un servicio contendrá en su parte funcional referencias a ontologías del dominio. Asimismo, las propiedades no-funcionales también deben estar definidas en una ontología, con el fin de que otros servicios puedan entender su significado. El framework Broccoli proporciona el lenguaje propio BSDL (del inglés, Broccoli Service Description Language) para la descripción de servicios y ontologías. Para fomentar la interoperabilidad, este lenguaje soporta referencias a conceptos descritos en otros lenguajes de ontologías existentes.

**Comunicación.** Como tercer elemento imprescindible para que un servicio pueda hablar con otro tenemos los canales de comunicación y los modelos de interacción que ambos han de emplear. Al respecto, en el modelo de servicios BSDM se distingue entre dos aspectos fundamentales:

- **Modelo de interacción.** El modelo BSDM soporta tres modalidades diferentes de interacción con las funcionalidades de un servicio. En primer lugar, una modalidad de tipo petición/respuesta síncrona o asíncrona que permite la invocación de una funcionalidad parametrizada con instancias de conceptos del dominio y la recepción de una respuesta puntual. En segundo lugar, una modalidad petición/respuesta asíncrona durable en el tiempo, con un número indeterminado de resultados. En tercer lugar, una modalidad de tipo publicación/subscripción, con soporte para comunicaciones uno-a-muchos, en la que se proporcionan resultados en forma de eventos asíncronos para todos los subscriptores de la funcionalidad. La riqueza de este modelo constituye una clara distinción frente a las soluciones clásicas basadas en tecnologías de Servicios Web que utilizan un modelo único de tipo petición/respuesta.
- **Groundings.** Se refieren a la forma de resolver la comunicación a nivel de protocolos de comunicación y tecnologías de red concretas. El modelo de servicios BSDM abstrae el grounding del servicio y permite diferentes realizaciones concretas del mismo. El framework Broccoli propone una realización inspirada en los patrones de mensajería asíncrona y capaz de soportar todas las modalidades de interacción descritas.

**Composición dinámica.** Un servicio BSDM puede definir funcionalidades compuestas por medio de una descripción semántica de una serie de requisitos funcionales y no-funcionales que necesitará cumplir a través de la interacción espontánea con otros servicios disponibles en el entorno en el momento de la ejecución. Esta

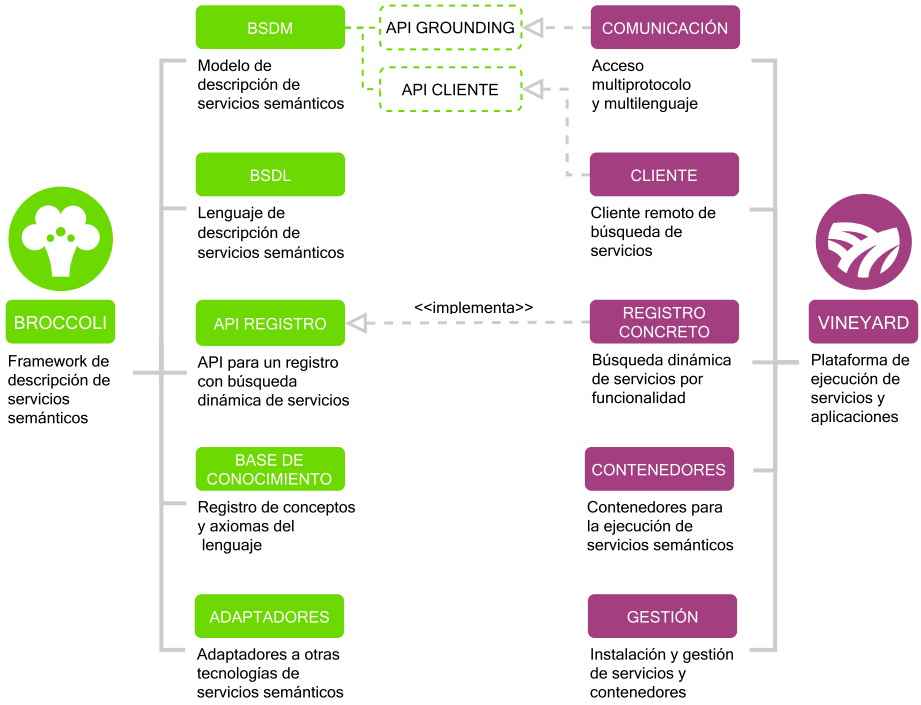



Figura 4.7: Relación entre los frameworks Broccoli y Vineyard.

descripción será realizada utilizando el lenguaje de descripción de servicios BSDL.

Además del modelo BSDM y el lenguaje BSDL, el framework Broccoli proporciona el API que debe cumplir cualquier implementación de un registro de servicios semánticos. Incluye, a su vez, una implementación de una base de conocimiento que permite la carga optimizada en memoria de conceptos y axiomas de diferentes lenguajes de descripción de ontologías. Por último, incluye adaptadores de otros modelos de servicios existentes al modelo BSDM. Por tanto, el framework Broccoli puede ser visto como un framework general de servicios semánticos para entornos ubicuos, con vocación de integrar múltiples tecnologías bajo un modelo de servicios común.

El otro framework a través del que se realiza el middleware orientado a servicios de la arquitectura HI3 es el framework Vineyard. Éste cumple dos funciones principales: i) proporcionar implementaciones de contenedores para la ejecución y gestión del ciclo de vida de servicios y aplicaciones de IAm en diferentes entornos computacionales; ii) proporcionar implementaciones en tecnologías concretas para aquellas APIs y componentes abstractos del framework Broccoli que lo requieran, como son las APIs del registro de servicios y el grounding abstracto basado en ca-

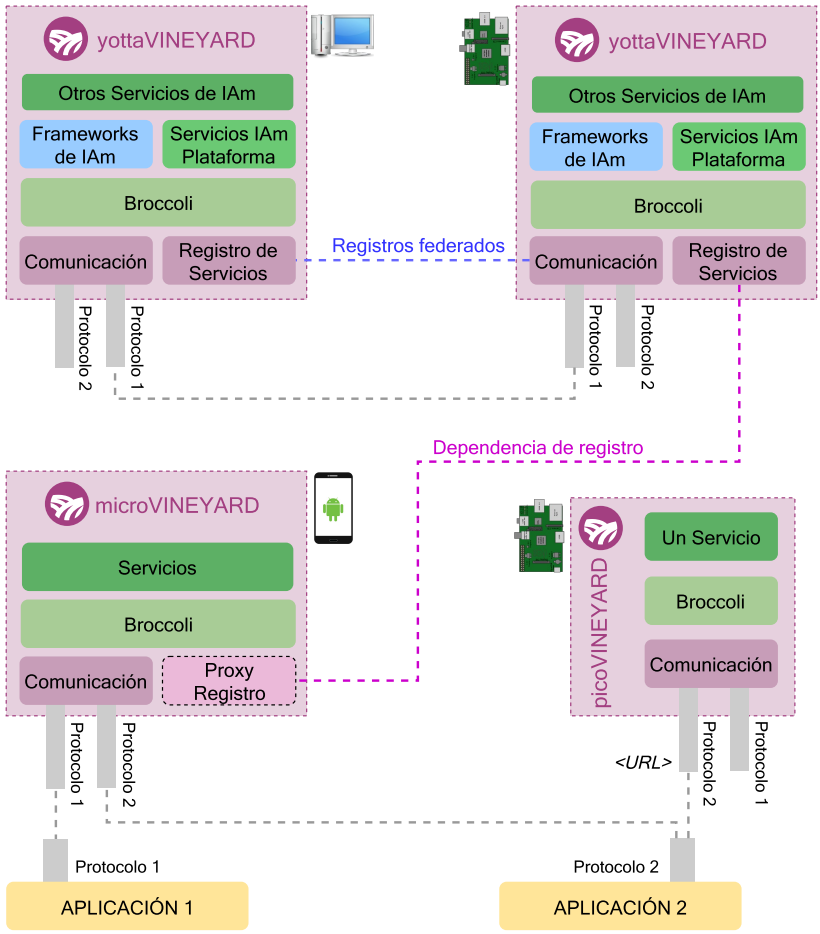



Figura 4.8: Varios tipos de contenedores Vineyard desplegados en un entorno.

nales de mensajería asíncrona. La Figura 4.7 proporciona una visión conceptual de la relación entre el framework Broccoli y el framework Vineyard.

En una primera aproximación se han considerado los siguientes tipos de contenedores de servicios y aplicaciones soportados por el framework Vineyard:

- **Contenedor yottaVineyard.** Se trata de un contenedor completo destinado a contener todo el middleware proporcionado por la arquitectura HI3 y proporcionar las capacidades máximas de un contenedor Vineyard.
- **Contenedor microVineyard.** La principal característica de este contenedor es que no incluye un registro de servicios y necesita federarse con algún contenedor yottaVineyard accesible en la red para tener acceso a un registro. Los servicios que se ejecutan en un microVineyard se registran de forma transparente en el registro remoto y usan dicho registro para buscar a otros ser-

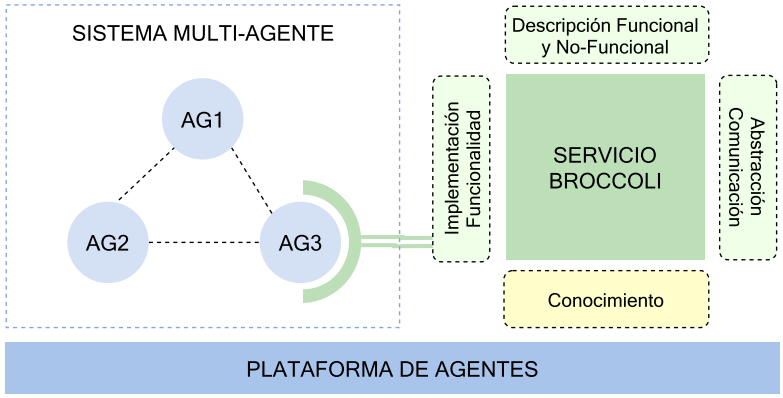



Figura 4.9: Integración de un Sistema Multi-Agente en la arquitectura HI3.

vicios. Se trata de un componente más ligero y con menos dependencias de tecnologías que el yottaVineyard, por lo tanto más fácilmente realizable en plataformas como smartphones o sistemas embebidos.

- **Contenedor picoVineyard.** Se trata de un contenedor mínimo destinado a la ejecución de un único servicio, con una o múltiples implementaciones de groundings. No dispone de registro ni tiene capacidad de federarse en un registro remoto. Por tanto, otros servicios y aplicaciones necesitan conocer de antemano el descriptor de este servicio para poder interaccionar con él. Se trata de una realización con mínimas requisitos para facilitar una implementación sencilla en prácticamente cualquier plataforma.

En la Figura 4.8 se ilustran las capacidades principales de los tres tipos de contenedores de ejecución de servicios propuestos a través de un ejemplo de despliegue de varios de estos contenedores en un mismo entorno. Así, se presentan dos contenedores yottaVineyard federados, de forma que cualquier componente que realice una búsqueda de servicios a través del registro de uno de estos contenedores podrá también encontrar, de forma trasparente, servicios que se ejecutan en el otro. Vemos también como los contenedores podrán proporcionar realizaciones de la comunicación a través de diferentes protocolos, de modo que los servicios ejecutándose en dichos contenedores podrán proporcionar a través de ellos más de un grounding simultáneamente. También se ilustra la dependencia con un registro remoto que tiene un contenedor microVineyard. Esta dependencia queda oculta para servicios y aplicaciones al poder utilizar un *proxy* del registro remoto como si fuese un registro local. Por último, vemos un contenedor picoVineyard que actúa a modo de envoltorio para la ejecución de un único servicio de forma aislada.

Finalmente, como una extensión de las capacidades de la arquitectura HI3 hemos explorado las posibilidades de utilización del paradigma de Sistemas Multi-

Agente para la implementación de funcionalidades en entornos de IAm. Para ello, se propone una aproximación para la integración del paradigma de Sistemas Multi-Agente con el paradigma SOC, de modo que dentro de la arquitectura HI3 sea posible realizar implementaciones concretas de funcionalidades como sistemas multi-agente que posteriormente exporten su funcionalidad pública como servicios semánticos a través de uno de los agentes del Sistema Multi-Agente. La Figura 4.9 proporciona una visión de alto nivel de esta aproximación, mientras que en el Capítulo 6 se abordan los detalles de la misma y se describe una implementación concreta basada en la plataforma de agentes Jade [JADE, 2015].

## 4.6. Construyendo un entorno de IAm

Esta sección se corresponde con el paso inmediatamente siguiente a la construcción del núcleo de la arquitectura HI3 en base a un middleware orientado a servicios para entorno ubicuos. Esto es, la siguiente fase es poblar esa arquitectura con middleware y servicios que resuelvan requisitos de más alto nivel característicos de los entornos de IAm. En concreto, como ya se ha introducido a lo largo del capítulo, en este trabajo nos centraremos en los aspectos relacionados con la Capa de Dispositivos y la Capa de Usuario.

Centrándonos en la Capa de Dispositivos, hemos visto como desde trabajos provenientes de diferentes ámbitos, como la Inteligencia Ambiental, el Internet de las Cosas o las Redes Semánticas de Sensores, se han propuesto soluciones para la integración de redes heterogéneas de dispositivos. Ninguna de las soluciones se ha impuesto como un estándar en la industria, y de nuevo nos encontramos ante un problema de proliferación de middleware heterogéneo. La aproximación que propone la arquitectura HI3 consiste en utilizar un modelo semántico común para la representación de dispositivos (que represente tanto la componente de información como la componente de operación de los dispositivos) junto con una serie de APIs y herramientas que permitan la integración de múltiples tecnologías a través de la utilización de elementos de tipo *gateway* o adaptador.

Utilizar modelos comunes de representación de redes de dispositivos permite integrar la heterogeneidad de tecnologías y middleware existentes, pero a su vez puede crear un problema de heterogeneidad de middleware no interoperable. Así, siguiendo la filosofía de la arquitectura HI3, en un último nivel de abstracción, el acceso se realizará a través de servicios interoperables y el conocimiento esencial del entorno físico se representará a través de ontologías comunes. En concreto, como parte de este trabajo se ilustra la integración en la arquitectura de una librería de IoT existente que proporciona una primera capa de abstracción en el acceso a redes de dispositivos heterogéneos, dejando abierta la posibilidad de integración de otras aproximaciones.

Otro de los requisitos clave identificados para la arquitectura HI3 tiene que ver con posicionar al usuario como un aspecto central en la concepción y desarrollo de cualquier sistema de IAm. Para ello es necesario que los sistemas *entiendan* a los usuarios en su contexto y puedan adaptar su comportamiento a sus necesidades. Así, desde este trabajo se considera como un aspecto importante la inclusión en una arquitectura de este tipo de mecanismos que permitan realizar un perfilado de los usuarios a partir de la monitorización de su actividad, de modo que diferentes tipos de aplicaciones puedan utilizar este conocimiento de los usuarios para adaptarse a sus necesidades dinámicamente. Esta labor excede del alcance del trabajo de esta tesis, no obstante, en la arquitectura HI3 se propone un modelo general para facilitar el acceso e integración de información de perfilado de usuarios proveniente de diferentes frameworks y tecnologías, de modo que aplicaciones heterogéneas concretas puedan sacar partido de ese tipo de información.

Asimismo, se propone un diseño para la integración en la arquitectura de una herramienta que facilite el desarrollo de interfaces físicas de usuario distribuidas y con capacidad de adaptarse dinámicamente a las características de cada usuario y al contexto dinámico en el que se realiza la interacción.

En el Capítulo 7 se describirán en detalle todas las propuestas, tanto desde un punto de vista conceptual como en lo referente al middleware y servicios concretos desarrollados e integrados como parte de la arquitectura HI3.

## 4.7. Resumen

A lo largo del capítulo se ha tratado de aportar una visión global del proceso de desarrollo de sistemas de IAm, partiendo de la premisa de que una metodología de diseño, un modelo conceptual común, una arquitectura software y un middleware, bien fundamentados, son herramientas útiles para la construcción de sistemas complejos como éstos. No obstante, en consecuencia con el estado actual de inmadurez del área de la IAm y otras áreas afines, se decidió comenzar el proceso re-evaluando los requisitos del área y los problemas principales que a nuestro juicio son el motivo del estancamiento en la investigación de soluciones integradas, en favor de la investigación en campos más concretos, como el IoT.

Partiendo de este análisis, se propusieron una serie de modelos conceptuales que deben guiar el proceso de desarrollo de sistemas de IAm. En concreto, se propuso el modelado de sistemas de IAm en base a componentes funcionales representados en forma de servicios interoperables, ubicuos y capaces de colaborar de forma espontánea adaptándose a los recursos y características del entorno. Se propuso asimismo, que el middleware que resuelve problemas comunes de IAm se abstraiga en último término también como servicios y modelos semánticamente descritos.

Por tanto, logrando proporcionar acceso ubicuo y uniforme a frameworks y herramientas heterogéneas.

Vimos, además, como estos principios de diseño y modelos conceptuales se materializan en la definición de la arquitectura HI3, que se fundamentada en los principios de las arquitecturas SOA y extiende sus capacidades para adaptarse a las necesidades de los entornos de Computación Ubicua. Para completar la construcción de la arquitectura HI3, sobre su base de middleware orientado a servicios se proponen soluciones para requisitos de más alto nivel que hemos considerado como calve para la mayoría de sistemas de IAm, como la adaptación dinámica de las funcionalidades a las necesidades y características de los usuarios o el acceso uniforme y ubicuo al entorno físico.

En el Capítulo 5 se describen en detalle los elementos principales con los que se construye el middleware SOA para entornos ubicuos, que constituye la base de la arquitectura HI3. En el Capítulo 6, se describirá en detalle la aproximación propuesta para la integración del paradigma de Sistemas Multi-Agente en la arquitectura HI3, de modo que sea posible utilizar este paradigma para implementar funcionalidades de IAm. Para completar la descripción detallada de las capacidades de la arquitectura HI3, en el Capítulo 7 se describen los frameworks y servicios de IAm propuestos para abordar requisitos de más alto nivel relacionados con la Capa de Usuario y la Capa de Dispositivos.

# Capítulo 5

# Middleware orientado a servicios para entornos ubicuos

## 5.1. Introducción

En el capítulo anterior se aportó una visión general de la arquitectura HI3. Vimos que uno de sus elementos principales es el middleware orientado a servicios (SOM, del inglés Service Oriented Middleware) en base al que se construye una arquitectura orientada a servicios (SOA, del inglés Service Oriented Architecture) adaptada a entornos ubicuos. En este capítulo nos centraremos en describir sus aportaciones y las principales decisiones de diseño adoptadas para su desarrollo.

Vimos, asimismo, en el capítulo 3, como las arquitecturas SOA constituyen un buen punto de partida para dotar a los sistemas de IAm de características como distribución, adaptación, autonomía o interoperabilidad, y también que este paradigma se enfrenta a numerosos desafíos de cara a su adecuación a entornos de Computación Ubicua. En la Tabla 5.1 se hace un resumen de las principales fortalezas y debilidades que, en este sentido, presenta el paradigma. Se puede destacar la falta de interoperabilidad real entre servicios, debida a la heterogeneidad de middleware en todos los niveles de las arquitecturas SOA. Igualmente, es especialmente relevante la focalización del paradigma en el desarrollo de tecnologías y estándares para el ámbito Web, que no siempre se adaptan bien a otros entornos.

En la Figura 4.5 habíamos ilustrado, desde un punto de vista conceptual, como los componentes funcionales y el middleware de IAm de la arquitectura HI3 se rea-

| ASPECTO | FORTALEZAS DE SOC | DEBILIDADES DE SOC |
|---|---|---|
| DESCRIPCIÓN DE LAS CAPACIDADES DEL SERVICIO | - Estándares como WSDL o SAWSDL.<br>- Aportaciones de modelos de descripción semántica como OWL-S y WSMO.<br>- Esfuerzos para definir ontologías de propiedades no-funcionales. | - Heterogeneidad de modelos no interoperables.<br>- Carencia de propiedades no-funcionales en los modelos más extendidos.<br>- Estándares y principales desarrollos focalizados en Servicios Web.<br>- No se suelen considerar mecanismos para el tratamiento automatizado de errores. |
| REPRESENTACIÓN DEL CONOCIMIENTO | - El paradigma favorece la separación entre el conocimiento y la implementación concreta de la funcionalidad.<br>- Lenguajes de ontologías para la descripción semántica del conocimiento.<br>- Gran capacidad descriptica del lenguaje estándar OWL. | - Heterogeneidad de ontologías no interoperables.<br>- El lenguaje estándar OWL dificulta la comprensión y definición del conocimiento por un humano.<br>- Razonamiento y procesamiento computacionalmente costoso de ontologías.<br>- Dificultad para introducir reglas en lenguajes de lógica descriptiva como OWL. |
| COMUNICACIÓN Y ACCESO | - Un protocolo estándar y extendido, SOAP.<br>- Separación de las capacidades funcionales de las capacidades de comunicación.<br>- Algunos modelos de servicios existentes contemplan la heterogeneidad de tecnologías de comunicación para resolver el acceso. | - Heterogeneidad de tecnologías de comunicación en entornos ubicuos mal soportada. En la práctica se tiende a delegar en WSDL y SOAP.<br>- Heterogeneidad y riqueza de modelos de interacción necesaria en entornos ubicuos mal soportada. Predominancia del modelo petición/respuesta típico de Servicios Web. |
| REGISTRO Y BÚSQUEDA | - SOA define un registro lógicamente centralizado para la búsqueda de servicios.<br>- Aportaciones de mecanismos para gestionar redes de registros descentralizados.<br>- Métodos para identificación universal de recursos y servicios.<br>- Aportaciones de algoritmos para matching entre capacidades demandadas-ofrecidas. | - Heterogeneidad de modelos de servicios.<br>- Heterogeneidad de ontologías.<br>- Escasa difusión de registros semánticos.<br>- Matching semántico computacionalmente costoso.<br>- Iniciativas de registros globales han perdido empuje en la actualidad. |

Tabla 5.1: Fortalezas/debilidades del paradigma SOC en Computación Ubicua.

lizan a través de servicios dinámicos, adaptables e interoperables soportados por el SOM. En la Figura 5.1 se ilustra, desde un punto de vista más próximo al diseño final, como se organizan los elementos conceptuales descritos en el anterior capítulo, para conformar la estructura de componentes software con la que se construye la arquitectura SOA HI3. Los elementos resaltados en color verde son proporcionados por el framework Broccoli, que se encarga principalmente de proveer los modelos, herramientas y APIs necesarios para construcción de servicios semánticos interoperables. Los elementos resaltados en color vino son proporcionados por el framework Vineyard, que se encarga principalmente de proveer plataformas contenedor para la ejecución de servicios e implementaciones concretas de algunos elementos abstractos del framework Broccoli, como el Registro de Servicios o Groundings de comunicación basados en protocolos concretos.

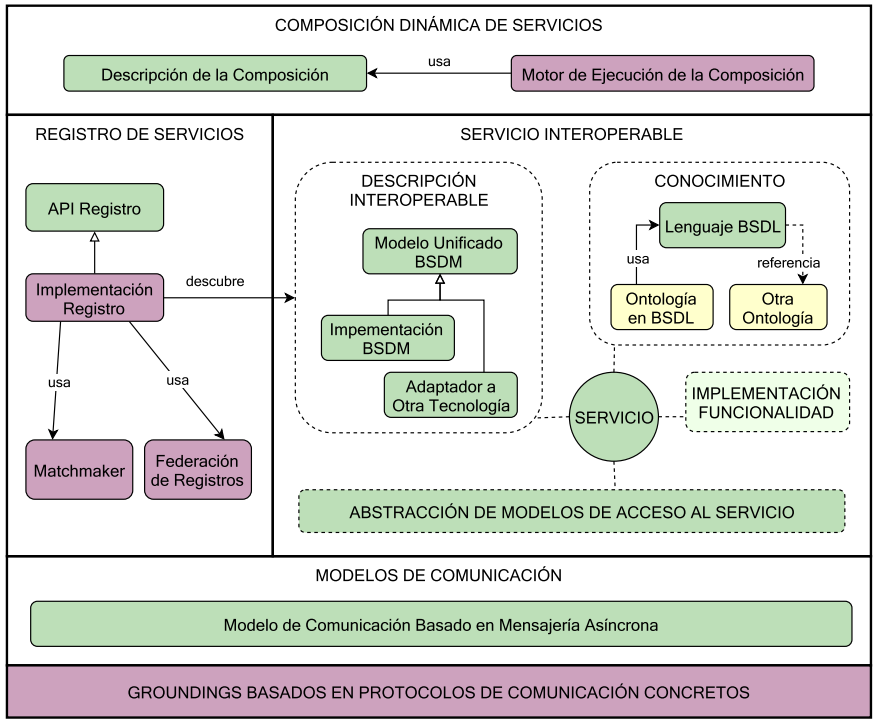



Figura 5.1: Principales elementos constructivos de la arquitectura SOA HI3.

Como ya se estableció en el Capítulo 4, y de nuevo queda patente en la Figura 5.1, un Servicio en la arquitectura HI3 se construye en torno a cuatro elementos fundamentales: una descripción semántica de las capacidades funcionales y no-funcionales del servicio, una representación común del conocimiento del dominio, uno o más métodos de comunicación con el servicio y una implementación concreta de la funcionalidad del servicio. La arquitectura HI3 trata de abundar en las capacidades de interoperabilidad entre servicios relacionadas con los tres primeros elementos proponiendo nuevos modelos y a su vez integrando modelos existentes.

Podemos observar como en los aspectos relacionados con la comunicación, tanto con los servicios como con los registros, se propone una abstracción del acceso en base a modelos de comunicación abstractos, que de forma trasparente resuelven el transporte utilizando diversos protocolos existentes. En el caso del acceso a los servicios, se introduce un nivel adicional de abstracción relacionado con el modo de interacción. Así, en el nivel más alto de abstracción, los servicios proporcionan un modelo de interacción enriquecido capaz de soportar conjuntamente diferentes patrones de interacción, como petición/respuesta y publicación/subscripción.

Por otro lado, también vemos como los componentes relacionados con la composición de servicios se asientan sobre las capas que resuelven tanto la construcción

del elemento Servicio como del elemento Registro. Esto es así porque en la arquitectura HI3 se proponen modelos dinámicos de orquestación de servicios que facilitan la adaptación de los componentes funcionales al entorno. Para soportar este dinamismo es necesario que, además de las funcionalidades propias de un servicio concreto, intervengan en la composición funcionalidades de otros servicios que han de resolverse en tiempo de ejecución a través de la búsqueda en el registro.

A lo largo de este capítulo se ilustrará de forma progresiva como se construye el middleware orientado a servicios para entornos ubicuos que constituye la base de la arquitectura HI3:

- En primer lugar, en las Secciones 5.2 y 5.3 se presentan, respectivamente, el lenguaje propuesto para la descripción de ontologías y servicios (BSDL), y el modelo uniforme para la descripción semántica servicios (BSDM). En el caso del modelo BSDM, además de sus características descriptivas se ilustra como puede abstraer a otros modelos de servicios existentes.
- A continuación, en la Sección 5.4 se detallan las características que tiene un registro de servicios de la arquitectura HI3, capaz de gestionar descriptores de servicios correspondientes a modelos de servicios heterogéneos.
- Utilizando todos los elementos anteriores, en la Sección 5.5 se presentan los mecanismos de composición dinámica de servicios de la arquitectura HI3.
- Por último, en la Sección 5.6 se explica como se construyen los diferentes tipos de contenedores de ejecución de servicios.

## 5.2. BSDL: un lenguaje para la especificación de servicios y ontologías

Un lenguaje neutral para la especificación de servicios permite que los proveedores de servicios publiquen las capacidades y propiedades de sus servicios en un formato interoperable. Por otro lado, posibilita el descubrimiento de servicios por medio del *matching* entre las especificaciones de servicios y los objetivos de búsqueda de funcionalidad enviadas por los clientes. De forma similar, si queremos posibilitar la interoperabilidad semántica entre servicios, también es necesario un lenguaje común para la descripción del conocimiento del dominio en base a ontologías, de forma que pueda ser compartido entre servicios de distintos proveedores.

Considerando los lenguajes ya existentes en el área de la Web Semántica, podemos identificar dos características principales a tener en cuenta:

- La expresividad del lenguaje. Se podría pensar que una mayor expresividad siempre es positiva, no obstante hay que tener en cuenta que a partir de cier-

tos niveles de expresividad el lenguaje deja de ser *decidible*, y por tanto imposibilita el razonamiento sobre él.

- Un enfoque basado en Lógicas Descriptivas frente a un enfoque basado en Programación Lógica [Studer et al., 2007]. En el primer caso, los lenguajes se basan en un elemento fundamental: los axiomas. Esta aproximación permite definir conceptos en base a la incorporación progresiva de axiomas que añaden descripciones de propiedades de dichos conceptos. Por tanto, para entender la naturaleza de un concepto es necesario recabar todos los axiomas que se refieren a dicho concepto. Por contra, en los lenguajes del área de la Programación Lógica (y más en particular Frame Logic), los conceptos son tratados como elementos de primer orden, de modo que la definición de las propiedades de un concepto se encuentra agrupada en torno a éste.

El lenguaje estándar elegido por el W3C para la descripción de ontologías es OWL [W3C, 2015c]. En sus variantes más utilizadas sigue una aproximación basada en Lógica Descriptiva. Se trata de un lenguaje que tiene sus bases en RDF [RDF, 2015] pero que proporciona una mayor capacidad descriptiva que éste. Sin embargo, por su propia naturaleza, las ontologías definidas en OWL son difíciles de mantener a medida que crecen por la dificultad de extraer el conocimiento representado en las mismas por parte de un ser humano. Por otro lado, no se han realizado avances definitivos a la hora de incorporar reglas a OWL, que pueden ser útiles por ejemplo para expresar restricciones sobre las descripciones de entradas o salidas de un servicio. En este sentido, el avance más consistente se refiere al uso de SWRL [SWRL, 2004] en conjunción con OWL, pero en este caso es necesario limitar la expresividad para mantener la decidibilidad. Además, la integración a nivel sintáctico es poco ortodoxa y las herramientas existentes son escasas e incompletas.

Partiendo de este contexto, el enfoque de este trabajo en lo referente al lenguaje de descripción de servicios y ontologías se inspira en los trabajos realizados en el framework de descripción de servicios WSMF [Roman et al., 2006], en el que se plantea un enfoque híbrido partiendo de las bases de la Lógica Descriptiva y utilizando una organización más "lógica" de la información en torno a conceptos, que posibilita un modelado conceptual próximo a la orientación a objetos, propia de los lenguajes del campo de la Programación Lógica como F-Logic. Sin embargo, los desarrollos relacionados con el framework WSMF nunca llegaron a completarse y las implementaciones parciales existentes están desmantenidas.

Así, a semejanza de WSML (leguaje del framework WSMF), en este trabajo se considera interesante disponer de un lenguaje para la descripción de servicios con una sintaxis próxima al estilo F-Logic, por la mayor cercanía de ésta (al utilizar conceptos y axiomas como elementos de primer orden del lenguaje) con los conceptos manejados con más frecuencia en el ámbito de la programación. Además, se ha valorado la enorme dificultad de integrar reglas en lenguajes como OWL, mientras

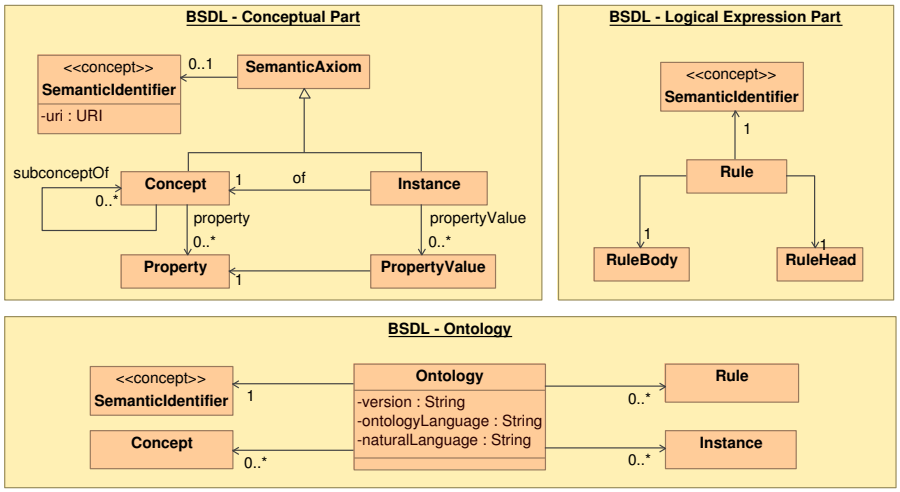



Figura 5.2: Elementos de primer orden del lenguaje BSDL.

que ésto se ha alcanzado con más éxito en el área de la Programación Lógica.

Por otro lado, en consecuencia con los objetivos planteados para la arquitectura HI3, se ha considerado el problema de la heterogeneidad de lenguajes de descripción de servicios y lenguajes de ontologías no interoperables. En este sentido, desde este trabajo se aportan soluciones basadas en la definición de un meta-lenguaje y un meta-modelo para la descripción de servicios y una serie de adaptadores a otras tecnologías existentes. Así, como parte de la arquitectura HI3 se define el lenguaje BSDL (del inglés Broccoli Service Definition Language), con una sintaxis próxima al formalismo de orientación a objetos y con capacidades para integrar referencias a conceptos definidos en ontologías de otros lenguajes, como OWL.

En la Figura 5.2 se representa una visión simplificada de los elementos de primer orden del lenguaje BSDL. En su parte conceptual, incluye tanto Conceptos como Instancias como elementos de primer orden. Un Concepto se compone de un grupo de Propiedades (que serán otros conceptos) y se admiten las relaciones de herencia, indicando que un concepto es subconcepto de otros. Una Instancia es una realización concreta de un Concepto con valores particulares de sus propiedades. Asimismo, tanto conceptos como instancias pueden tener un identificador (SemanticIdentifier) que los identifica de forma unívoca.

El lenguaje también contempla la especificación de expresiones lógicas en base a la definición de Reglas. Estas reglas tendrían la forma de *triples* en los que la Cabecera y el Cuerpo podrían ser conceptos, instancias o reglas. El desarrollo del sistema de reglas ha quedado fuera del alcance de este trabajo. No obstante, la propuesta del mismo es utilizar reglas para definir restricciones sobre conceptos e

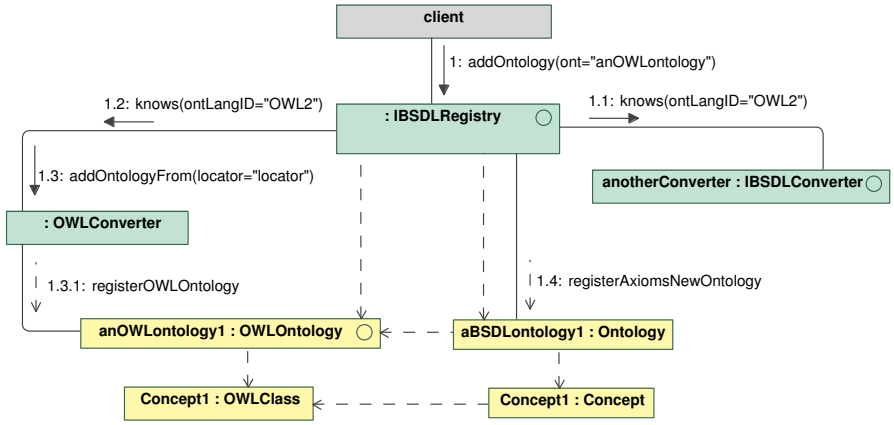



Figura 5.3: Gestión de ontologías heterogéneas en el registro de axiomas BSDL.

instancias, manteniendo la monotonicidad del lenguaje.

Por último, conceptos, instancias y relaciones entre los mismos se organizan en ontologías BSDL que describen el conocimiento de un dominio concreto. Estas ontologías deben tener un identificador único en la red de tipo URI (del inglés Uniform Resource Identifier). Además, en BSDL se propone asociar con las ontologías otra meta-información relacionada con el versionado, la autoría, el lenguaje natural en que están descritos los conceptos y el lenguaje de ontologías en el que fueron definidos originalmente (en el caso de que la descripción en BSDL sea una traducción de una definición previa en otro lenguaje, por ejemplo OWL).

El Apéndice A incluye la especificación formal del lenguaje BSDL, de modo que cualquiera pueda implementar un cliente interoperable con este lenguaje.

El lenguaje BSDL será utilizado en la arquitectura HI3 de dos formas: i) como un lenguaje unificado para la especificación de ontologías, dado que en la definición de una ontología soporta la referencia a conceptos especificados en otros lenguajes; ii) como un lenguaje para la descripción de servicios semánticos. En la sección siguiente se detallará el modelo para la descripción de servicios BSDM propuesto, incluyendo la ontología BSDL que lo define formalmente.

Para dar soporte, al menos en parte, a la heterogeneidad de lenguajes de ontologías existente, se propone un modelo para la gestión de axiomas y ontologías especificados en distintos lenguajes (ver Figura 5.3). Un registro de axiomas del lenguaje BSDL (IBSDLRegistry) se configura con una serie de *plugins* que permiten cargar otras ontologías y crear una representación común en lenguaje BSDL con referencias a los conceptos de la ontología original. Así, cualquier cliente trabajará con representaciones en lenguaje BSDL al margen de la heterogeneidad de lengua-

Listado 5.1: Fragmento de una ontología OWL.

```
<rdf:RDF xml:base="http://hi3project.com/owl/examples/tasksModel#" >

<owl:Class rdf:ID="NamedThing">
   <rdfs:subClassOf rdf:resource="#Thing"/>
</owl:Class>

<owl:Class rdf:ID="Project">
   <rdfs:subClassOf rdf:resource="#NamedThing" />
</owl:Class>

<owl:Class rdf:ID="Task">
   <rdfs:subClassOf rdf:resource="#NamedThing" />
</owl:Class>

<owl:DatatypeProperty rdf:about=#nameOF">
   <rdfs:domain rdf:resource=#NamedThing"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:about="#taskFromProject">
   <rdfs:range rdf:resource="#Project"/>
   <rdfs:domain rdf:resource="#Task"/>
</owl:ObjectProperty>

(. . .)
```

jes original. En concreto, en la arquitectura HI3 se ha incluido un *plugin* para el lenguaje OWL, de modo que BSDL y OWL pueden convivir de forma transparente.

En los Listados 5.1 y 5.2 se ilustra como desde una ontología descrita en lenguaje BSDL se importa una ontología en lenguaje OWL y se referencian sus conceptos.

## 5.3. BSDM: un modelo unificado para la descripción semántica de servicios ubicuos

Como vimos en el Capítulo 3, en el área de la Web Semántica existen varias propuestas de modelos y lenguajes para la descripción de servicios (e.j. OWL-S, SAWSDL, WSMO). Desafortunadamente, todas las propuestas presentan varias carencias y su implantación en sistemas reales es muy baja, manteniéndose la predominancia de los Servicios Web clásicos basados en WSDL y SOAP. Además, la única propuesta con un alto grado de madurez es la menos ambiciosa, SAWSDL, que se limita a incluir anotaciones semánticas en descriptores WSDL. No obstante, para la definición del modelo de descripción de servicios BSDM (Broccoli Service Description Model) propuesto en este trabajo, se han tenido en cuenta numerosas ideas y aportaciones de modelos precedentes como los mencionados. En concre-

Listado 5.2: Fragmento de un servicio descrito en BSDL con referencias a OWL.

```
<import prefix="bsdl" to="http://hi3project.com/broccoli/bsdl"/>
<import prefix="bsdm" to="http://hi3project.com/broccoli/bsdm"/>
<import prefix="bsdmProfile" to="http://hi3project.com/broccoli/bsdm/profile"/>
<import prefix="taskService"
     to="http://hi3project.com/broccoli/examples/taskService"/>

<import prefix="model"
 to="http://hi3project.com/owl/examples/tasksModel" dialect="owl:2"/>

<instance of="bsdm#serviceDescription" URI="taskService#serviceDescription">
   <with property="profile">
      <instance of="bsdm#serviceProfile" URI="taskService#serviceProfile">

         <with property="name" value="taskService"/>

         <with property="advertisedFunctionality">
            <instance of="bsdmProfile#advertisedFunctionality"
                     URI="taskService/profile#buildProjectFrom">

               <with property="name" value="buildProjectFrom"/>

               <with property="input">
                  <with property="name" value="Tarea"/>
                  <with property="type"valueURI="model#Task"/>
               </with>

               <with property="output">
                  <with property="name" value="Proyecto"/>
                  <with property="type"valueURI="model#Project"/>
               </with>
            </instance>
         </with>
   (. . .)
```

to, son especialmente interesantes los objetivos ambiciosos de WSMO respecto a la incorporación de propiedades no-funcionales, la inclusión de múltiples funcionalidades en un mismo servicio o la distinción entre las capacidades de un servicio y los objetivos de un cliente que requiere un servicio.

En el caso de la arquitectura HI3 se intenta, además, abordar la interoperabilidad entre servicios sin estar obligatoriamente ligados a una tecnología concreta. Para este objetivo, se propone un modelo de descripción de servicios que amplia las capacidades descriptivas de los modelos que han alcanzado una mayor difusión (SAWSDL y OWL-S) y a la vez sirve como un meta-modelo capaz de abstraer e integrar otros modelos existentes. Se posibilita así, el descubrimiento, comparación y composición de funcionalidades descritas en distintas tecnologías. En este sentido, la aproximación de nuestro modelo BSDM se inspira en parte en algunos de los trabajos derivados de la arquitectura de IAm Amigo conducentes al desarrollo del middleware AmIi, que promueve la interoperabilidad entre servicios semánti-

cos [Georgantas et al., 2010]. La aproximación de AmIi también propone un modelo genérico al que son traducidos distintos lenguajes de descripción de servicios. No obstante, en lo referente a la abstracción de la comunicación entre servicios, el middleware AmIi únicamente se centra en la interoperabilidad entre protocolos de tipo RPC (SOAP, RMI, CORBA). Ya hemos mencionado que para la arquitectura HI3 se considera imprescindible contemplar también modelos de interacción basados en eventos, como el modelo publicación/subscripción, necesarios para resolver muchas de las interacciones que se dan típicamente en entornos de IAm. Además, protocolos como RMI y CORBA pueden adaptarse razonablemente bien a entornos locales controlados, pero no son adecuados para su utilización en redes de Computación Ubicua e IoT donde es necesario integrar dispositivos heterogéneos, incluyendo dispositivos móviles, y redes de comunicación global.

También han servido de inspiración para la conceptualización del modelo BSDM otras aproximaciones de tipo MDA (del inglés Model Driven Architecture) que plantean la definición en fase de diseño de un meta-modelo estático de descripción del servicio (descrito en UML por ejemplo) y la posterior traducción automática a los distintos modelos concretos existentes [López-Sanz et al., 2008] [Autili et al., 2014]. El carácter estático de este tipo de aproximaciones no las hace válidas para soportar la interoperabilidad de servicios heterogéneos descubribles en tiempo de ejecución. Aún así, plantean una propuesta interesante que podría incorporarse en la fase de diseño de servicios BSDM, permitiendo el modelado de éstos en lenguaje UML y la generación automática del descriptor BSDL correspondiente.

En la Figura 5.4 se presenta una visión de alto nivel de los componentes descriptivos propuestos para el modelo BSDM.

- **Servicio.** Un servicio semánticamente descrito con el modelo BSDM.
  - *Perfil.* Todo servicio BSDM debe tener un perfil a través del que se especifican tanto sus capacidades funcionales como las propiedades no-funcionales que lo caracterizan.
  - *Grounding.* Un grounding define un modo de acceso y un modelo de interacción con el servicio. Permite la abstracción de los diferentes aspectos de la comunicación y la realización de la misma a través de diferentes tecnologías y protocolos. El modelo BSDM soporta la coexistencia de varios groundings concretos heterogéneos en un mismo servicio.
  - *Implementación.* Se refiere a la realización concreta de la funcionalidad del servicio en un lenguaje, un paradigma y unas tecnologías determinadas. El modelo BSDM deja esta decisión en manos de los desarrolladores. No obstante, el modelo propuesto sí anima a la incorporación al mismo de plugins que automaticen en la medida de lo posible el desarrollo de las funcionalidades usando lenguajes y tecnologías concretas. Veremos más adelante un ejemplo de ello.

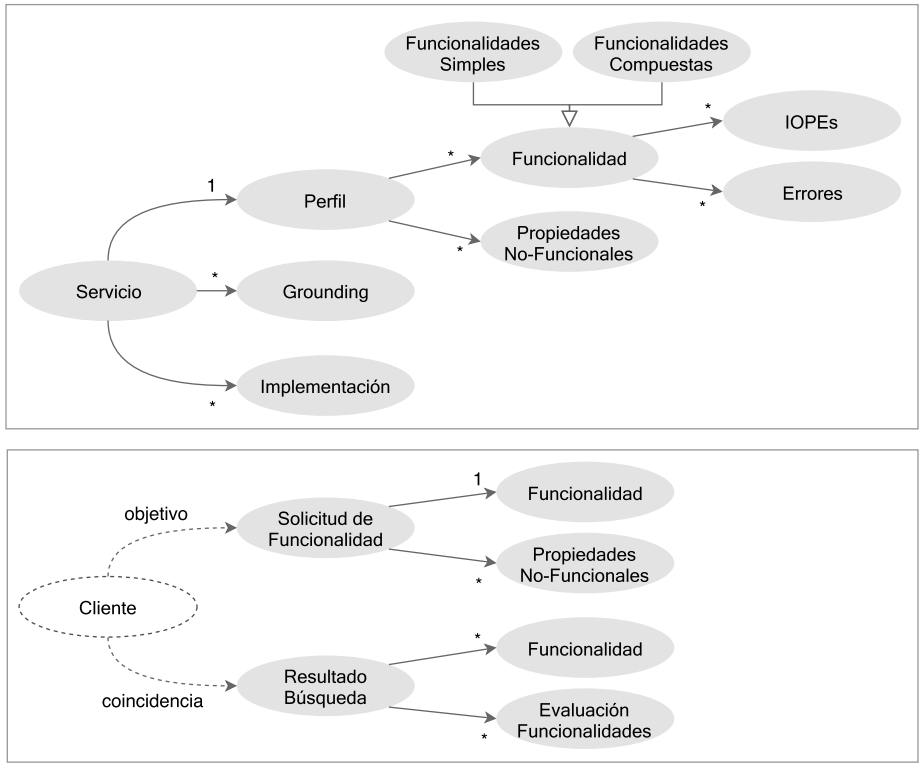



Figura 5.4: Visión general del modelo de servicios BSDM.

- **Cliente.** Se propone una separación explícita entre las capacidades publicadas por un servicio y las capacidades demandadas por un cliente.
  - *Búsqueda de funcionalidad.* Un cliente expresa su objetivo en base a la descripción de las capacidades funcionales y no-funcionales que requiere para realizar su tarea. Con el fin de enriquecer la semántica de la búsqueda de funcionalidades, el modelo soporta la distinción entre las características demandadas que son obligatorias y las que son opcionales. Además, también permite expresar la preferencia sobre una característica concreta dentro de una característica más general.
  - *Resultado búsqueda.* Si la búsqueda correspondiente a un objetivo expresado por un cliente se realiza a través de un Registro de Servicios BSDM se obtendrá un resultado que incluye aquellas funcionalidades de servicios que tienen una coincidencia con el objetivo. Además, se obtendrá una evaluación (o *ranking*) que indica el grado de similitud entre los resultados de la búsqueda y el objetivo especificado.

La especificación completa del modelo BSDM se realiza a través de una ontología descrita en lenguaje BSDL (publicada y disponible en https://github.com/GII/

broccoli/specification/bsdmOntology.xml). En los apartados siguientes se aborda una descripción más detallada de cada uno de los elementos que componen la descripción semántica de un servicio HI3 según el modelo BSDM. Además, en el Apéndice A se incluye la especificación formal en notación ABNF del modelo.

### 5.3.1. Descripción del perfil funcional y no-funcional de un servicio

En el modelo BSDM el perfil de un servicio es descrito como un conjunto no vacío de funcionalidades y un conjunto de propiedades no-funcionales (ver Figura 5.5). Al contrario que otros modelos, como OWL-S, en BSDM se considera la existencia de varias funcionalidades en un mismo servicio. Una funcionalidad se describe con: su nombre; un posible conjunto de propiedades no-funcionales; una posible categoría, que permite clasificar las funcionalidades respecto a alguna taxonomía semántica general o de un dominio de aplicación; y su interfaz. La interfaz de una funcionalidad se describe en base a conjuntos de entradas, salidas, condiciones y efectos. Las entradas y salidas se describen con sus nombres y con la referencia a un concepto existente en una ontología. Condiciones y efectos están pensados para permitir expresar mediante reglas restricciones sobre las entradas y cambios en el entorno ante determinadas condiciones de las salidas. Dado que el sistema de reglas de BSDL está pendiente de desarrollar tampoco están actualmente soportadas las condiciones y efectos en la descripción de servicios.

La descripción de una funcionalidad así definida se asemeja a las propuestas de OWL-S y AmIi, con excepción de la ausencia del calificador semántico para la *Categoría Funcional* de la funcionalidad y la ausencia de soporte para propiedades no-funcionales en OWL-S. No obstante, en BSDM también se introduce otra característica que le da mayor riqueza a la especificación: la posibilidad de calificar las entradas de una funcionalidad como opcionales. La idea detrás de este concepto es que una funcionalidad podría realizar su tarea con una serie de información de entrada obligatoria, pero otras entradas podrían simplemente permitir "mejorar la calidad" en la resolución de la tarea pero no imposibilitarla. Con este tipo de consideración pueden especificarse funcionalidades validas para un mayor rango de clientes potenciales, favoreciendo la flexibilidad y la adaptación.

En la Figura 5.5 vemos como existe una jerarquía de tipos de funcionalidad en el perfil de un servicio BSDM. Esta característica supone una distinción notable con los principales modelos existentes, y en concreto se proponen los siguientes tipos de funcionalidades:

- Funcionalidad correspondiente a una ejecución puntual de un proceso que puede recibir un conjunto de entradas y producir un conjunto de salidas (AdvertisedFunctionality).

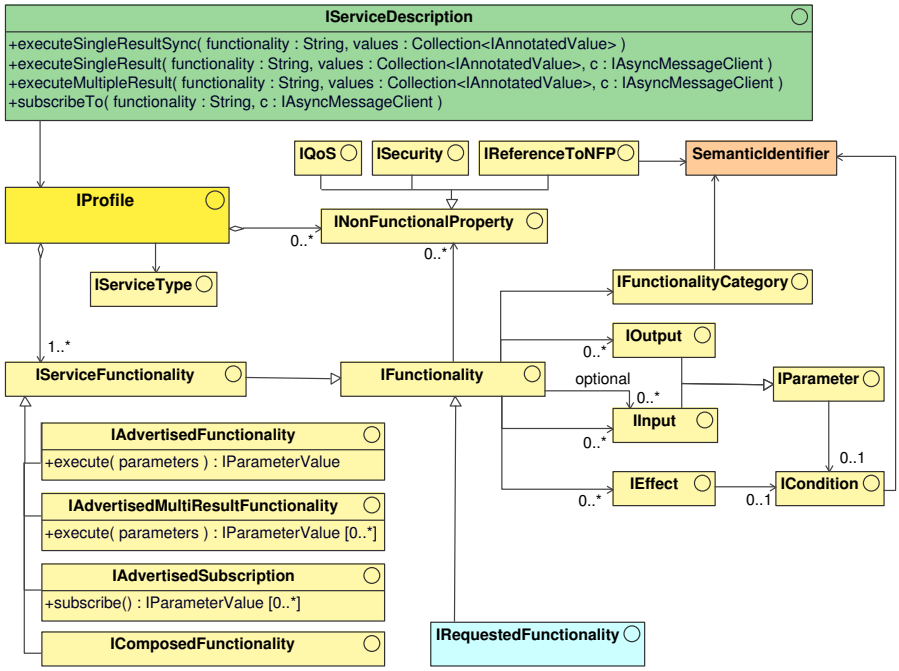



Figura 5.5: Profile del modelo de descripción de servicios BSDM.

- Funcionalidad durable en el tiempo, que puede recibir un conjunto de entradas y producir varios resultados espaciados en el tiempo (AdvertisedMultiResultFunctionality).
- Funcionalidad asimilable al concepto de publicación, que ejecuta el servicio por iniciativa propia en vez de a petición de un cliente (AdvertisedSubscription). Puede proporcionar salidas a modo de eventos que son comunes para todos los clientes interesados en la funcionalidad.
- Funcionalidades compuestas, que describen orquestaciones dinámicas formadas tanto por funcionalidades del propio servicio como por funcionalidades de otros servicios descubiertas en tiempo de ejecución (ComposedFunctionality). Los mecanismos de composición serán explicados en el apartado 5.5.

Ya se ha mencionado que en la aproximación propuesta por la arquitectura HI3 se considera como un punto clave la especificación de propiedades no-funcionales de los componentes. Aspectos relacionados con la seguridad, la privacidad o la calidad de servicio pueden ser tan determinantes como la funcionalidad a la hora de seleccionar y utilizar un servicio. Es más, en ocasiones una propiedad no-funcional puede ser un extra deseable, pero en otras ocasiones puede constituir una característica irrenunciable. Por ejemplo, un componente que actúe como cliente potencial de un servicio podría considerar inasumible proporcionar como entradas del servi-

Listado 5.3: Fragmento de la descripción en BSDL de un perfil de un servicio.

```
<ontology URI="http://hi3project.com/broccoli/examples#taskServiceOntology"
        ontologyLanguage="http://hi3project.com/broccoli/bsdl#ontology"
        versionNumber="0.9.0">

<import prefix="bsdl" to="http://hi3project.com/broccoli/bsdl"/>
<import prefix="bsdm" to="http://hi3project.com/broccoli/bsdm"/>
<import prefix="bsdmProfile" to="http://hi3project.com/broccoli/bsdm/profile"/>
<import prefix="taskService"
             to="http://hi3project.com/broccoli/examples/taskService"/>
<import prefix="model"
             to="http://hi3project.com/owl/examples/tasksModel" dialect="owl:2"/>

<instance of="bsdm#serviceDescription" URI="taskService#serviceDescription">
   <with property="profile">
   <instance of="bsdm#serviceProfile" URI="taskService#serviceProfile">

      <with property="name" value="taskService"/>

      <with property="advertisedFunctionality">
         <instance of="bsdmProfile#advertisedFunctionality"
                  URI="taskService/profile#buildProjectFrom">

            <with property="name" value="buildProjectFrom"/>

            <with property="input">
               <with property="name" value="Tarea"/>
               <with property="type" valueURI="model#Task"/>
            </with>
            <with property="output">
               <with property="name" value="Proyecto"/>
               <with property="type" valueURI="model#Project"/>
            </with>
         </instance>
      </with>

      <with property="nonFunctionalProperty">
         <instance of="bsdmProfile/nonFunctionalProperties/security/AES">
            <with property="keyLength" value="128"/>
         </instance>
      </with>
   </instance>

(. . .)
```

cio datos sensibles (información médica por ejemplo) de una persona si el servicio no proporciona mecanismos de privacidad adecuados. Aunque se trata de un tipo de información que en la practica se tiene poco en cuenta en la descripción de servicios, existe un consenso referente a la necesidad de definir taxonomías comunes de propiedades no-funcionales [OASIS, 2015] [Becha and Amyot, 2012] [Chung and Leite, 2009]. El modelo BSDM soporta dos enfoques para la inclusión de propiedades no-funcionales en la descripción de un servicio: i) una jerarquía propia de propiedades no-funcionales proporcionada por la arquitectura HI3; ii) la posibili-

dad de incluir referencias a taxonomías descritas en ontologías externas. Por último, mencionar que el modelo soporta la especificación de propiedades no-funcionales en dos ámbitos distintos: propiedades que afectan globalmente a todo el servicio y propiedades asociadas a una funcionalidad concreta.

Finalmente, el perfil de un servicio BSDM también incluye la posibilidad de especificar un *tipo* para el servicio que haga referencia a alguna taxonomía general o a una taxonomía particular de un dominio de aplicación.

En el Listado 5.3 se incluye un fragmento correspondiente a la descripción del perfil de un servicio semántico BSDM que tiene una entrada, una salida y una propiedad no-funcional relacionada con la seguridad. En este caso, el concepto relacionado con la propiedad no-funcional proviene de una ontología BSDL, mientras que los conceptos utilizados en la entrada y la salida provienen de una ontología OWL. Constituye por tanto un ejemplo del soporte que proporciona la arquitectura HI3 para la integración de ontologías heterogéneas.

### 5.3.2. Modelos de interacción y grounding

En este ámbito, un modelo de interacción define la forma de relacionarse un servicio con un cliente del mismo. El modelo BSDM delega principalmente en el *grounding* del servicio para resolver el acceso remoto al mismo. Así, en el diagrama de la Figura 5.1 vimos como la comunicación con un servicio HI3 se abstrae en varios niveles: i) la abstracción del modelo de acceso a la funcionalidad del servicio; ii) la abstracción de los modelos de comunicación a través de los que se resuelve la interacción; iii) realizaciones concretas basadas en protocolos de comunicación existentes. Como ya se ha mencionado, en este aspecto encontramos otra de las diferencias de enfoque más relevantes entre el modelo de servicios BSDM y los principales modelos de servicios semánticos existentes. El modelo BSDM escapa de la asunción, tan común en el ámbito de la Web Semántica, de que el modelo de interacción con un servicio ha de estar basado en un esquema petición/respuesta. Consideramos que un modelo único de tipo petición/respuesta no captura muchas de las interacciones espontáneas y desacopladas que demandan los entornos de Computación Ubicua e IAm. Así, este tipo de interacciones a menudo encajan mejor en un modelo de interacción basado en eventos.

Por otro lado, además de tomar en consideración la importancia de soportar más de un modo de interacción entre servicios también se considera importante que el modelo BSDM abstraiga las particularidades de la interacción, con independencia del modo utilizado, proporcionando una descripción del grounding simple y de alto nivel. Así, como vimos en la definición del perfil de un servicio, en el nivel más alto de abstracción se consideraron tres tipos de funcionalidades, que capturan conceptualmente los modos de interacción propuestos para el acceso a la funcio-

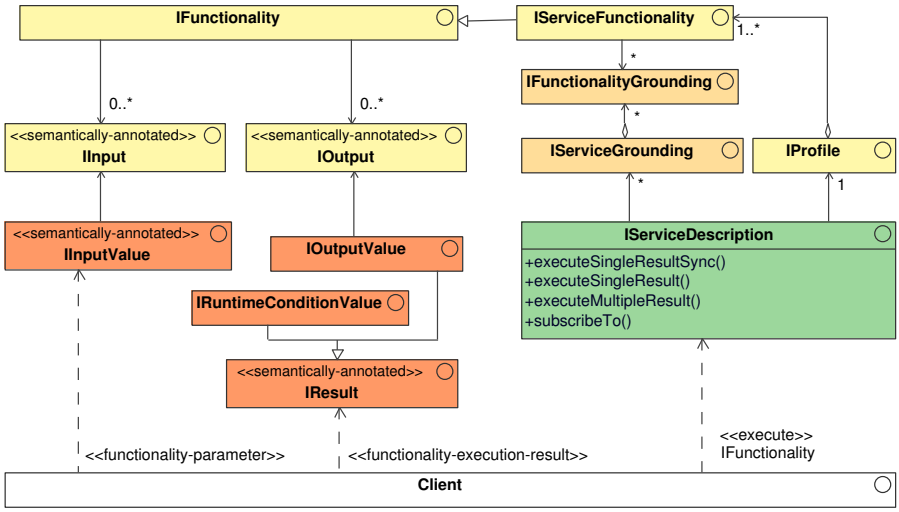



Figura 5.6: Grounding BSDM, que posibilita la ejecución de una funcionalidad.

nalidad de un servicio. Así, tratando de englobar tanto el patrón de comunicación petición/respuesta como el patrón publicación/subscripción basado en eventos, el modelo BSDM soporta los siguientes tipos de interacción:

1. Acceso asíncrono a una funcionalidad a la que el cliente puede enviarle un conjunto de entradas semánticamente anotadas y tras la ejecución de la funcionalidad ésta podrá enviar una única respuesta que con un conjunto de salidas semánticamente anotadas o un error semánticamente anotado.

2. Acceso síncrono y puntual a una funcionalidad según el mismo esquema de la anterior. El componente cliente inicia activamente la interacción y no continua su tarea hasta que recibe un resultado.

3. Interacción durable entre el componente cliente y una funcionalidad del servicio en la que ni el número de respuestas ni la duración de la interacción puede predecirse de antemano. Por tanto, se trata de un acceso asíncrono a una funcionalidad cuya ejecución no puede considerarse puntual. Al igual que en los casos anteriores, tanto las entradas que espera recibir la funcionalidad como los resultados que produce estarán semánticamente anotados.

4. Interacción asíncrona basada en el modelo publicación/subscripción. La funcionalidad genera eventos a lo largo del tiempo relacionados con su tarea y se los notifica a cualquier cliente interesado.

Como puede observarse en la Figura 5.6, en BSDM todos los modos de interacción propuestos se realizan a través del grounding del servicio y están accesibles a

través de los métodos de ejecución que proporciona la descripción del servicio (interfaz IServiceDescription). Un mismo servicio BSDM puede contener la descripción de varios groundings, implementados utilizando protocolos de comunicación distintos. Sin embargo, para cualquier componente interesado en una funcionalidad del servicio esta complejidad quedará oculta.

Observando más en detalle la Figura 5.6 podemos ver los principales elementos del modelo BSDM involucrados en resolver el acceso a una funcionalidad de un servicio por parte de otro servicio o cualquier componente que actúe como cliente. El proceso parte de que el cliente disponga del descriptor del servicio, bien porque lo ha obtenido a través de una búsqueda en un registro de servicios o bien porque es un servicio que conocía previamente. Así, para interactuar con la funcionalidad deseada el cliente necesita proporcionar los valores para los parámetros de entrada (Input) demandados por la funcionalidad en su especificación. Para ello utilizará instancias BSDL del concepto InputValue proporcionado por la ontología BSDM (bsdmOntology). Tras solicitar la ejecución de la funcionalidad proporcionando los valores de entrada deseados, el cliente podrá recibir resultados generados por dicha funcionalidad durante su ejecución, que serán instancias BSDL de alguno de los subconceptos del concepto Result de la ontología BSDM (en concreto una salida OutputValue o una condición de error RuntimeCondition). Tanto los valores de entrada como los valores de resultado estarán semánticamente anotados con referencias a conceptos de una o varias ontologías del dominio, recordando que la solución soporta la coexistencia de ontologías en distintos lenguajes. Las particularidades de como se resuelve la comunicación y la interacción entre el cliente y el servicio quedan así ocultas para ambos participantes de la interacción.

Ya se ha explicado que BSDM actúa también como un meta-modelo de servicios que permite integrar otros modelos existentes (veremos más sobre esto en la sección 5.3.5), por tanto el proceso de ejecución de funcionalidades será también transparente entre servicios descritos en distintos modelos que estén soportados por la arquitectura HI3.

Para dar soporte internamente al modelo abstracto de interacción que acabamos de presentar, se propone una realización del mismo en base a un modelo concreto basado en canales de mensajería asíncrona. Recordemos, sin embargo, que este modelo concreto de interacción seguirá siendo agnóstico de los protocolos de comunicación sobre los que se resolverá en el momento de la ejecución, y que serán proporcionados por los groundings concretos, posibilitando la utilización de un protocolo u otro sin que se vea afectada la interacción entre servicios. Así, obviando en primer lugar la lógica de gestión de los canales de comunicación, se definen los mensajes estándar necesarios para resolver cualquier interacción con independencia de canales y protocolos utilizados. En la Figura 5.7 se ilustra la sencilla estructura de mensajes necesaria para resolver la interacción, compuesta de un mensaje que encapsula una petición de ejecución de una funcionalidad BSDM y un mensaje

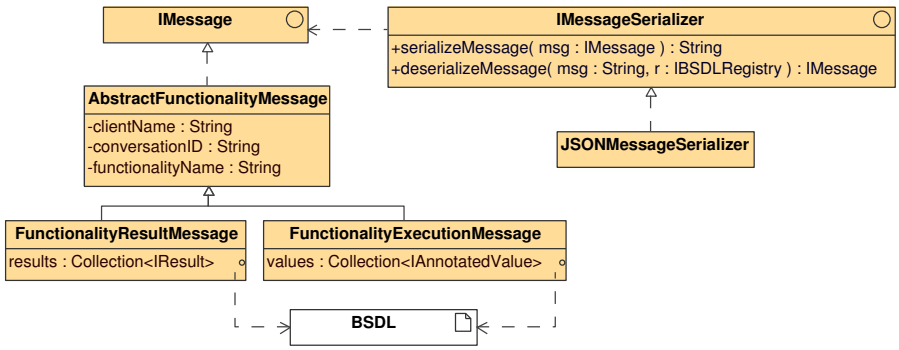



Figura 5.7: Mensajes del modelo de interacción basado en mensajería asíncrona.

que encapsula los resultados de dicha ejecución que se enviarán como respuesta. Además, es necesario un componente *serializador* capaz de transformar el contenido de los mensajes (objetos BSDL) a un formato multi-plataforma óptimo para el intercambio de mensajes y que pueda ser enviado por la mayoría de protocolos de comunicación existentes. En concreto la arquitectura HI3 propone una realización concreta de un serializador a formato JSON [JSON, 2015].

La especificación exacta de estos mensajes y el modelo de interacción se incluye en el Apéndice A. Junto con la especificación del lenguaje BSDL y el modelo BSDM, también incluidas en el mismo apéndice, constituyen los únicos elementos que un desarrollador necesita estrictamente conocer para construir un componente en cualquier lenguaje y plataforma que sea compatible con un servicio HI3. No obstante, recordemos que para construir un servicio interoperable con un servicio HI3 ni siquiera es necesario ninguna de estas especificaciones puesto que la arquitectura ofrece soporte para otros modelos (como OWL-S) y para otros lenguajes de ontologías (como OWL).

El elemento que falta para la definición de un grounding basado en mensajería asíncrona que soporte los cuatro modos de interacción propuestos en el modelo BSDM es la gestión de la lógica y la operación de los canales de mensajería. En la Figura 5.8 se proporciona una visión de los elementos principales en torno a los que se abstrae el grounding. En primer lugar se definen dos tipos de canales lógicos: ChannelConsumer para la recepción asíncrona de mensajes y ChannelProducer para el envío de mensajes. Estos elementos serán los que tendrán que extender las implementaciones de groundings concretos que vayan a utilizar un protocolo de comunicación existente en particular. Por otro lado, vemos que tanto el grounding de un servicio como el grounding concreto de una funcionalidad manejan canales de este tipo. Así, un grounding de un servicio proporciona siempre un canal en el que recibir los mensajes de petición de ejecución de funcionalidades por parte de los clientes. Para enviar los resultados de la ejecución de una funcionalidad, generados

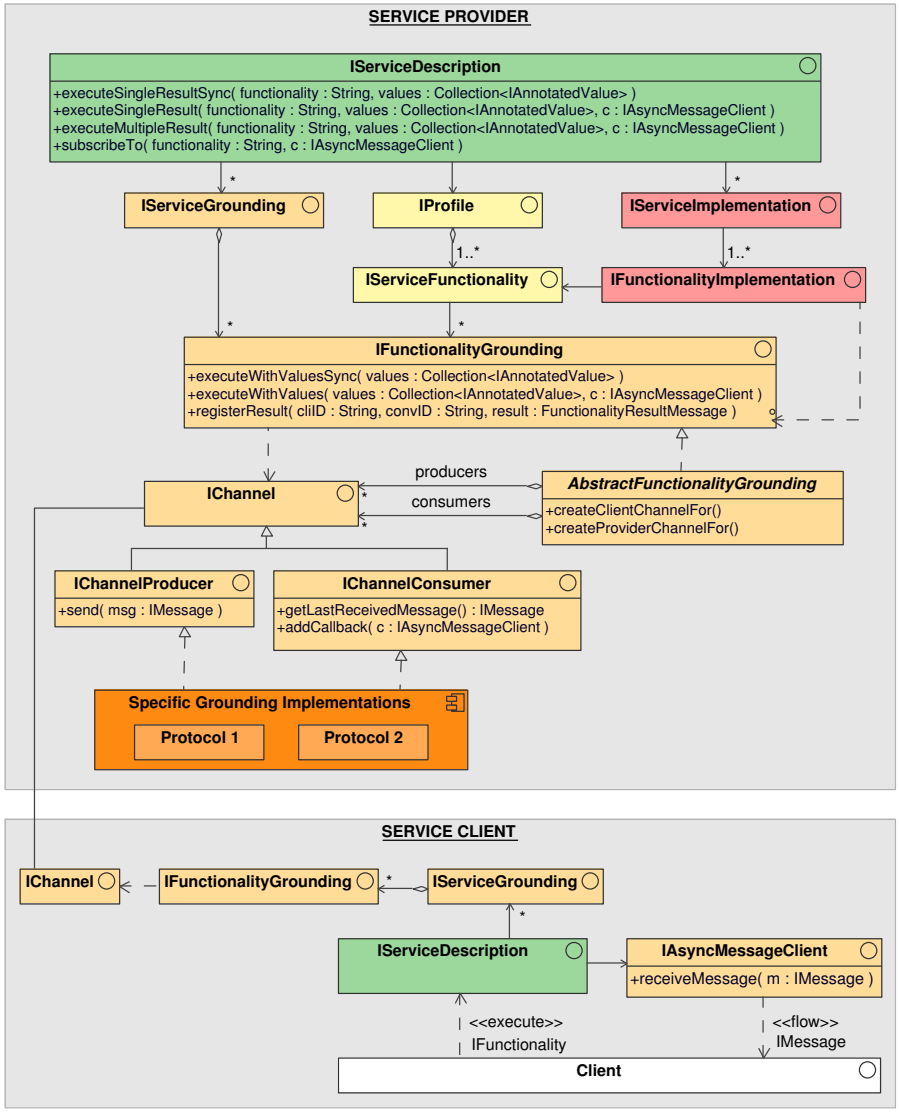



Figura 5.8: Modelo abstracto de interacción basado en mensajería asíncrona.

desde su implementación, se crea dinámicamente un canal lógico asociado con la funcionalidad concreta y el cliente, que se mantendrá mientras dure la interacción entre ambos. En el caso concreto de interacciones correspondientes a funcionalidades de subscripción, se compartirá el canal lógico de repuesta entre los clientes (un publicador-múltiples suscriptores). Desde el lado del cliente del servicio, éste solicitará la ejecución de una funcionalidad y recibirá un objeto *callback* en el que irá recibiendo los resultados.

Listado 5.4: Ejemplo de definición en BSDL de un grounding de un servicio.

```
<ontology URI="http://hi3project.com/broccoli/examples#taskServiceOntology">

<import prefix="bsdl" to="http://hi3project.com/broccoli/bsdl"/>
<import prefix="bsdm" to="http://hi3project.com/broccoli/bsdm"/>
<import prefix="bsdmProfile" to="http://hi3project.com/broccoli/bsdm/profile"/>
<import prefix="bsdmGrounding" to="http://hi3project.com/broccoli/bsdm/grounding"/>
<import prefix="taskService"
            to="http://hi3project.com/broccoli/examples/taskService"/>

<instance of="bsdm#serviceDescription" URI="taskService#serviceDescription">
   <with property="profile">
      (. . .)
   </with>

   <with property="grounding">
   <instance of="bsdm#asyncMessageServiceGrounding"
            URI="taskService#serviceGrounding">

      <with property="url" value="tcp://localhost:61703"/>

      <with property="functionalityGrounding">
         <instance of="bsdmGrounding#asyncMessageFunctionalityGrounding"
                  URI="taskService/grounding#buildProjectFromGrounding">

            <with property="advertisedFunctionality"
                     ofURI="taskService/profile#buildProjectFrom"/>
         </instance>
      </with>
   </instance>
   </with>
</instance>
```

En el descriptor de un servicio BSDM la única información obligatoria que será necesario especificar para un grounding concreto será la URL de acceso al mismo y las funcionalidades a las que está asociado (ver Listado 5.4).

### 5.3.3. Tratamiento y recuperación de excepciones

En el ámbito de los sistemas informáticos el término *error* se usa principalmente para designar una situación en la cual una acción, método o función de un servicio ha producido un resultado de distinta naturaleza del esperado debido a condiciones que no están en el control inmediato de la implementación. Por otra parte, existe un consenso amplio en usar el término *excepción* para errores que se producen en tiempo de ejecución y que son capturados en alguno de los niveles del código donde se producen. Por tanto, en este apartado nos referiremos únicamente a excepciones, que son los errores que han podido ser capturados y posibilitan su tratamiento.

En los estándares actuales de servicios provenientes de la Web Semántica, con

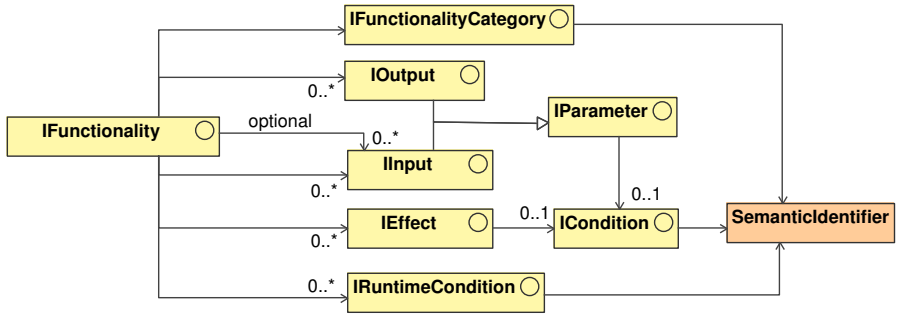



Figura 5.9: Representación estática de los posibles resultados de una funcionalidad correspondientes a una excepción.

carácter general, se deja al cliente la responsabilidad de lidiar con el tratamiento de las excepciones producidas durante la invocación de un servicio. En este trabajo, se considera que la incorporación de estrategias para el tratamiento automatizado de excepciones es un aspecto importante que puede favorecer la flexibilidad y autonomía en la ejecución de servicios que colaboran para resolver tareas en entornos dinámicos. Estrategias que pueden, por ejemplo, posibilitar la recuperación automática ante una excepción buscando una alternativa en otro servicio para continuar la ejecución, o probando la ejecución de la misma funcionalidad con un conjunto alternativo de valores de entrada.

En el modelo BSDM se contemplan las estrategias de tratamiento y recuperación de excepciones en dos niveles. En un primer nivel, se realiza una extensión de la definición del perfil de un servicio para incluir la definición de las posibles excepciones que puede producir como resultado la ejecución de una funcionalidad concreta (ver Figura 5.9). Utilizando esta capacidad, es posible proporcionar una ontología que describa los casos de excepción de un determinado dominio de aplicación y añadir a la descripción del servicio las posibles excepciones de cada funcionalidad semánticamente anotadas con referencias a dicha ontología.

El verdadero potencial de la inclusión de mecanismos para la gestión de excepciones está en el desarrollo de estrategias dinámicas y automáticas de recuperación ante excepciones. En este sentido, existen algunos lenguajes de programación como Common Lisp y Smalltalk, donde se implementan mecanismos de continuación ante excepciones en los que se desacopla dónde se señala la excepción (condición), dónde se captura (manejador) y dónde se trata (reinicio) [Seibel, 2005]. A partir de ello se posibilita la implementación de distintos mecanismos para tratar una excepción, pudiendo escoger un mecanismo correctivo y continuar la ejecución en el punto en el que se produjo, o incluso dejar la excepción a un mecanismo del sistema. En este tipo de aproximación se separa la definición de un conjunto de acciones de

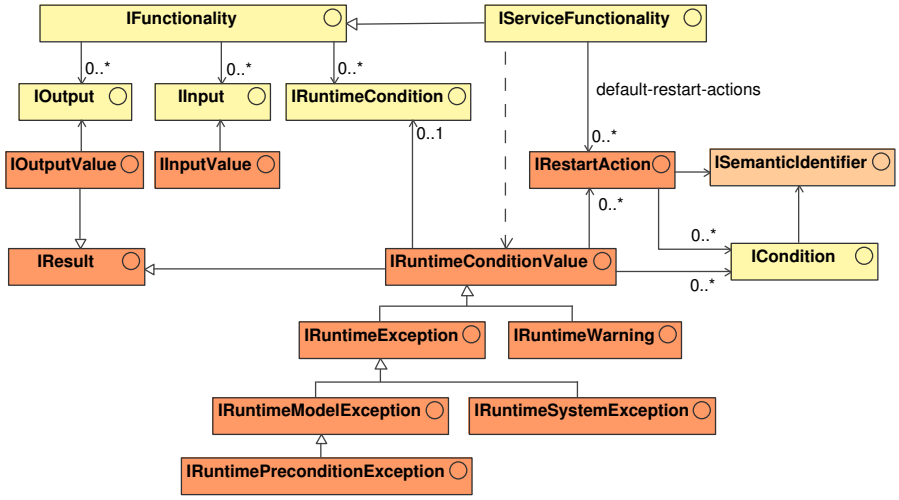



Figura 5.10: Tratamiento de los resultados de error de una funcionalidad.

reinicio con las que se puede continuar la ejecución después de una excepción, del mecanismo que tiene la responsabilidad de elegir una acción de reinicio concreta en tiempo de ejecución.

Creemos que este tipo de aproximación es muy adecuada para entornos dinámicos de Computación Ubicua e IAm, en los que los componentes tienen que adaptarse a los recursos funcionales existentes en el entorno. Esta adaptación se facilitaría si ante una excepción que se produce durante la interacción entre dos servicios, existe la posibilidad de que se reinicie la interacción tratando de cumplir las precondiciones previamente no satisfechas que causaron la excepción. Inspirándonos en este tipo de aproximaciones, en el diagrama de la Figura 5.10 se proponen las bases para implementar mecanismos automáticos de continuación ante la ocurrencia de excepciones en la invocación de funcionalidades de servicios BSDM. Partimos de la base de que una funcionalidad puede producir en tiempo de ejecución unos resultados correspondientes a excepciones (RuntimeConditionValue) e incluso que éstos pueden corresponderse con una especificación estática semánticamente anotada (RuntimeCondition) incluida en el descriptor del servicio. Así, durante la ejecución de la funcionalidad del servicio puede surgir una excepción del modelo, una excepción de sistema, etc. Estas excepciones concretas (RuntimeConditionValue) podrían además ser anotadas por el servicio para indicar información de las condiciones en que se produjeron, y serán capturadas por el cliente del servicio. Entonces, el cliente puede escoger entre las siguientes opciones:

- Lanzar alguna de las *acciones de reinicio* identificadas en la funcionalidad.
- Continuar la ejecución, si el tipo de señal lo permite (en el caso de una ex-

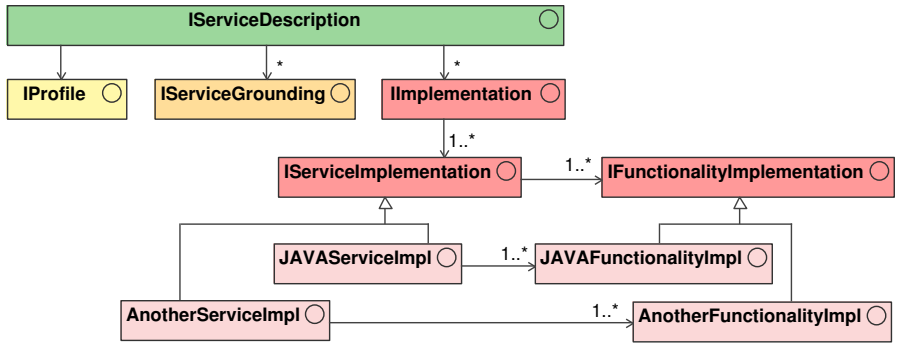



Figura 5.11: Descripción de la implementación en el modelo BSDM.

cepción no sería posible pero sí ante una señal de aviso, por ejemplo).

- Buscar otro servicio que cumpla los requisitos funcionales y probar su ejecución. Si esta solución funciona, puede añadirse a las *acciones de reinicio* de la funcionalidad que causó la excepción, relacionándola con la *condición* señalada.

En la sección 5.5 veremos como se utiliza este tipo de aproximación en la definición de un mecanismo de composición dinámica de servicios.

### 5.3.4. Implementación de la funcionalidad

Como ya se ha mencionado previamente, la arquitectura HI3 no impone restricciones para que los desarrolladores implementen internamente los componentes funcionales utilizando el paradigma y las tecnologías más adecuadas al problema.

Gracias al modelo BSDM, el desarrollador que implemente la funcionalidad de un servicio sabe que cuando a su servicio llegue una petición de ejecución de una funcionalidad a través del grounding tendrá disponible un mecanismo común para recibir la información de dicha petición en lenguaje BSDL y de acuerdo con la ontología del modelo BSDM, con independencia de la tecnología en la que se implemente la funcionalidad. Es responsabilidad del desarrollador vincular la información recibida que describe la petición de ejecución de una funcionalidad con el código concreto que implementa dicha funcionalidad en el servicio. Típicamente, el desarrollador tendrá, además, que mapear los valores de entrada (InputValue) para la ejecución de la funcionalidad a estructuras del modelo de datos interno de su implementación. Igualmente es posible que, para enviar los resultados de la ejecución de la funcionalidad, tenga que convertir estructuras de datos internas a instancias de la ontología correspondiente y así poder construir los OutputValue necesarios.

Listado 5.5: Ejemplo de anotaciones en una clase Java para el mapeo automático entre instancias BSDL y objetos Java.

```
@Namespace("http://hi3project.com/bsdl/examples/tasksModel#")
public class Task
{
   private String name;

   @nameOF
   public String setName(String name)
   {
      this.name = name;
   }

   (. . .)
}
```

Listado 5.6: Ejemplo de definición en BSDL de una implementación de servicio.

```
<ontology URI="http://hi3project.com/broccoli/examples#taskServiceOntology">

<import prefix="bsdl" to="http://hi3project.com/broccoli/bsdl"/>
<import prefix="bsdm" to="http://hi3project.com/broccoli/bsdm"/>
<import prefix="bsdmProfile" to="http://hi3project.com/broccoli/bsdm/profile"/>
<import prefix="bsdmImpl" to="http://hi3project.com/broccoli/bsdm/implementation"/>
<import prefix="taskService"
            to="http://hi3project.com/broccoli/examples/taskService"/>

<instance of="bsdm#serviceDescription" URI="taskService#serviceDescription">
   (. . .)

   <with property="implementation">
   <instance of="bsdm#serviceImplementation" URI="taskService#serviceImpl">

      <with property="implementationType" value="javaJenaBeansBSDL"/>

      <with property="functionalityImplementation">
      <instance of="bsdmImpl#functionalityImplementationJena"
               URI=taskService/implementation#buildProjectFrom">

         <with property="class"
                  value="com.hi3project.broccoli.examples.tasks.TaskService"/>

         <with property="method" value="buildProjectFrom"/>

         <with property="advertisedFunctionality"
                  ofURI="taskService/profile#buildProjectFrom"/>
      </instance>
      </with>
   </instance>
   </with>
</instance>
```

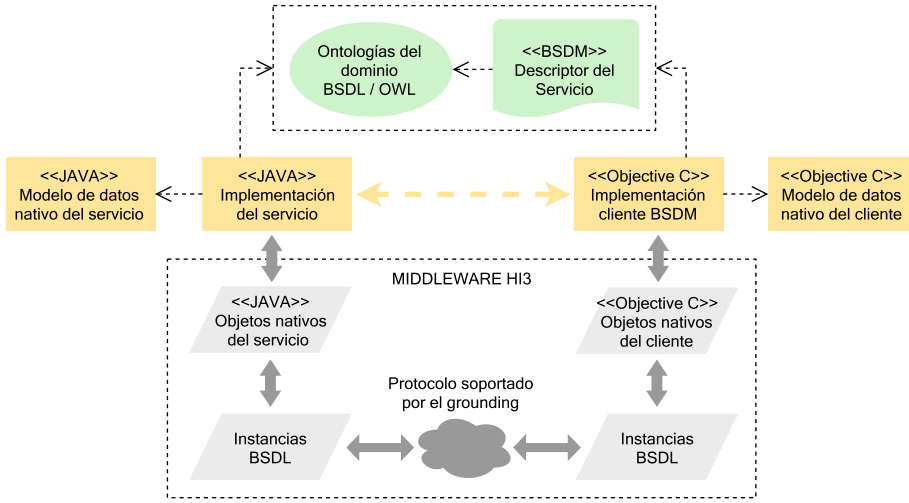



Figura 5.12: Interacción cliente-servicio independiente de las tecnologías concretas de implementación.

El proceso anterior supone un trabajo de desarrollo repetitivo cuando se implementan distintos servicios en un mismo lenguaje y plataforma. Por ello, aunque conceptualmente la implementación de un servicio no necesariamente requiere verse reflejada en la descripción semántica del mismo, creemos que es deseable soportar la posibilidad de incluir estrategias que faciliten al desarrollador dicha implementación utilizando tecnologías concretas. Así, en la Figura 5.11 vemos el detalle de la parte del modelo BSDM dedicada a la descripción de la implementación de un servicio. El modelo proporciona elementos abstractos para representar la información de descripción de la implementación de un servicio y sus funcionalidades. Además, describe el tipo de implementación realizada. Esta estructura funciona como un sistema de *plugins*, que permite extender el modelo con descripciones para implementaciones realizadas en lenguajes y plataformas concretos.

En particular, se ha desarrollado un *plugin* de implementación para el lenguaje de programación Java, que facilita el proceso de ejecución de una funcionalidad implementada en este lenguaje, automatizando el siguiente proceso:

- Tras la recepción de una petición de ejecución de una funcionalidad a través del grounding del servicio, se convierten automáticamente las instancias de valores de entrada (instancias de una ontología BSDL o una ontolgoía OWL) recibidas para la ejecución de la funcionalidad a objetos de un modelo Java proporcionado por la implementación. Para ello, utiliza un mecanismo basado en anotaciones que indica la correspondencia entre los conceptos de una ontología y clases Java. Utilizando el mecanismo de reflexión de Java se crean automáticamente las instancias necesarias. En el Listado 5.5 se muestra un

ejemplo de una clase Java anotada con referencias a una ontología.

- Se invoca automáticamente al método de la clase Java que ejecuta la implementación de la funcionalidad proporcionándole como parámetros las instancias de objetos Java que representan las entradas y que fueron creadas en el paso anterior. Para saber cuál es el método y la clase que representan la implementación de la funcionalidad, éste se define en el modelo BSDM en la descripción de una implementación Java como se muestra en el Listado 5.6.
- Los resultados de la ejecución de la funcionalidad que hay que enviar al cliente se pueden traducir automáticamente desde objetos Java de un modelo de la implementación del servicio a las instancias de la ontología BSDL adecuada. Para ello se utiliza el mismo método basado en anotaciones utilizado para convertir los parámetros de entrada.

De forma similar, es posible añadir al modelo BSDM otros *plugins* con estrategias similares para diferentes lenguajes de programación. Así, en la Figura 5.12 se ilustra el proceso de interacción cliente-servicio entre dos componentes implementados en lenguajes distintos. Vemos como, gracias a las capacidades del modelo BSDM para la descripción de la implementación, se consigue automatizar en mayor medida la resolución de la interacción con un servicio, evitando que el desarrollador "contamine" su código con lógica de control relacionada con la comunicación.

### 5.3.5. Adaptadores a otros modelos de descripción de servicios

Entre los requisitos de la arquitectura HI3 se estableció la necesidad general de abstraer e integrar otras tecnologías bajo modelos comunes que las hagan interoperables, con el fin de luchar contra el problema de la heterogeneidad del middleware. Así, el modelo de servicios BSDM también tiene entre sus objetivos integrar otros modelos existentes, actuando como un meta-modelo. Para ilustrar esto, veremos como, en concreto, se ha incorporado al framework de servicios semánticos de la arquitectura HI3 un adaptador del modelo de servicios OWL-S [W3C, 2004] al modelo BSDM.

Según la especificación de OWL-S, el Profile es el encargado de especificar los IOPEs (entradas, salidas, pre-condiciones y efectos) de un servicio, así como también sus propiedades no-funcionales. En concreto, los IOPEs abarcan lo siguiente:

- Inputs equivalentes a los IInput de una funcionalidad en BSDM.
- Outputs equivalentes a los IOuput de una funcionalidad en BSDM.
- Results que sirven para enlazar condiciones con entradas, salidas y efectos.
- Precondition equivalente a ICondition asociada a un IEffect en BSDM.

En cuanto a las propiedades no-funcionales, OWL-S incluye el concepto ServiceParameter que se refiere a una lista de propiedades con nombre y valor que califican a un servicio. Por tanto se pueden asimilar al concepto NonFunctionalProperty de una funcionalidad en BSDM.

En el caso de OWL-S, un servicio define una única funcionalidad, mientras que en BSDM cada Profile agrupa varias funcionalidades, cada una con sus IOPEs y sus propiedades no-funcionales. Por tanto, lo que en la especificiación de OWL-S es un servicio, en BSDM tiene su equivalente en una funcionalidad.

Por tanto, con ciertos matices, se puede decir que el modelo BSDM abarca un super-conjunto de las características proporcionadas por el modelo OWL-S. Así, un servicio OWL-S es traducido al modelo BSDM como un servicio que prescinde de algunas de las posibilidades expresivas de BSDM y contiene una única funcionalidad en su Profile. Esta funcionalidad será de tipo AdvertisedFunctionality ya que OWL-S únicamente contempla un modo de ejecución puntual de tipo petición/respuesta. En la Figura 5.13 podemos ver como se adaptan estos elementos principales de la descripción de un servicio OWL-S al modelo BSDM, extendiendo los elementos correspondientes de este último. Un objeto OWLSAtomicService representa un servicio OWL-S completo, que conoce su grounding y sabe cómo ejecutarse. Para completar la adaptación al modelo BSDM, un objeto OWLSServiceDescription se encarga de aportar la representación de un servicio BSDM completo construido a partir de un servicio OWL-S cuya única funcionalidad está descrita en un objeto OWLSAtomicService.

Adaptando de esta forma un servicio OWL-S al modelo BSDM, se posibilita que ambos tipos de servicios puedan convivir en una arquitectura HI3 de forma transparente para otros servicios o aplicaciones que los usen. Así, un servicio OWL-S envuelto como un servicio BSDM podrá incorporarse a un registro de servicios y ser buscado, descubierto y ejecutado transparentemente. Una búsqueda de servicios, por ejemplo, podrá proporcionar como resultado un conjunto de descripciones de servicio en el que se incluyan servicios BSDM y servicios OWL-S sin ninguna distinción, más allá de sus capacidades particulares.

En la Figura 5.13 también podemos ver la correspondencia entre conceptos de OWL-S y BSDM que proporciona el adaptador de OWL-S para permitir la ejecución de la funcionalidad de un servicio OWL-S a través de la interfaz de un servicio BSDM. Un cliente que desee ejecutar un servicio deberá proporcionar unos valores adecuados para las entradas declaradas por la funcionalidad. Y esta podrá responder con unos valores de salida. OWL-S utiliza instancias de ontologías OWL como valores de las entradas/salidas de sus servicios. Gracias a las capacidades del lenguaje BSDL, los servicios BSDM pueden soportar referencias a conceptos e instancias en múltiples lenguajes de ontologías, por tanto en este caso únicamente es necesario envolver los objetos que representan entradas y salidas en OWL-S

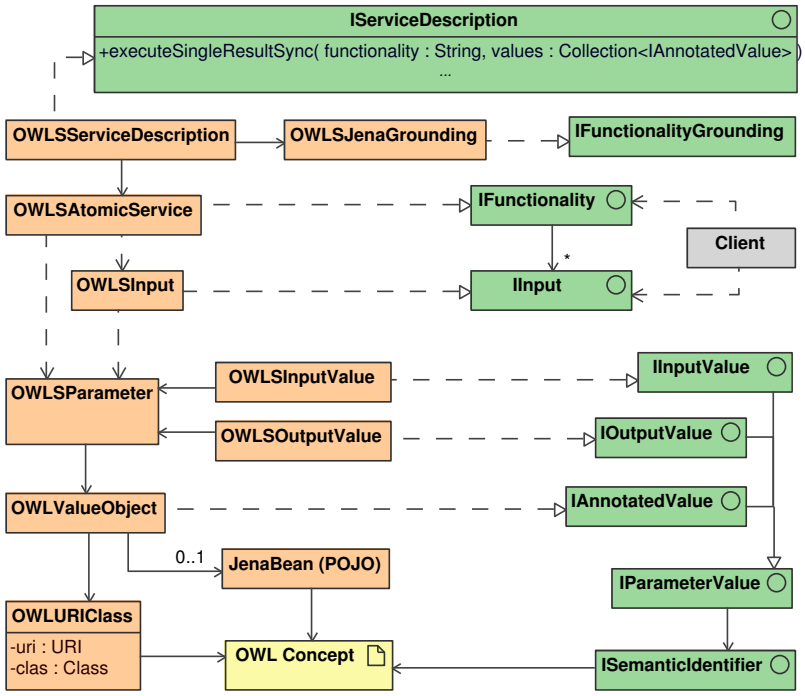



Figura 5.13: Adaptación de conceptos principales de OWL-S al modelo BSDM.

como entradas y salidas de BSDM con referencia al concepto de la ontología OWL correspondiente.

## 5.4. Registro y búsqueda de servicios

Uno de los componentes principales de toda arquitectura SOA es el registro de servicios. Su objetivo es proporcionar un elemento lógicamente centralizado a través del que buscar servicios disponibles en el entorno. En consecuencia, la arquitectura HI3, siguiendo el paradigma SOC propone un registro de servicios con capacidad de búsqueda de servicios semánticos en base a sus capacidades funcionales y no-funcionales. Para ello, un registro de servicios de la arquitectura HI3 podrá ser configurado con uno o más *Matchmakers*, que permitirán comparar las capacidades buscadas por un cliente con las capacidades ofrecidas por los servicios registrados, y uno o más *Rankers*, que permitirán puntuar la similitud entre las capacidades buscadas y las capacidades ofrecidas.

A diferencia de modelos como OWL-S, el modelo de servicios BSDM proporciona un elemento distinguido (RequestedFunctionality) para especificar los objeti-

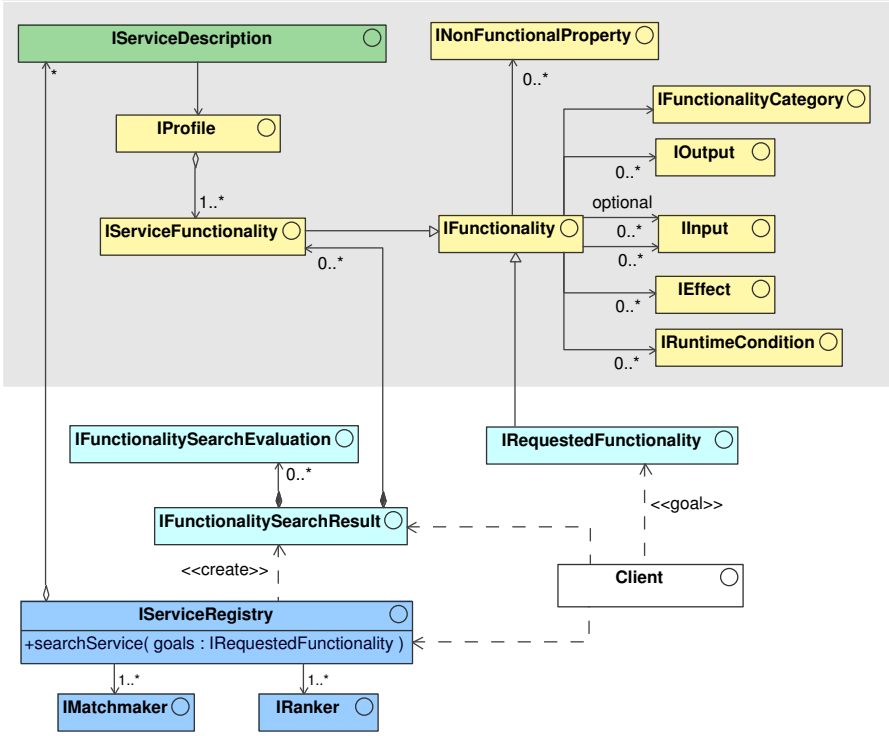



Figura 5.14: Especificación de objetivos de búsqueda en el modelo BSDM.

vos de un componente cliente que desea encontrar en el entorno otro componente (servicio) que ofrezca unas determinadas capacidades funcionales y opcionalmente unas determinadas propiedades no-funcionales. Como se ve en la Figura 5.14, este elemento permite utilizar cualquier subconjunto de las capacidades descriptivas de una funcionalidad para que un cliente especifique un objetivo. A partir de este objetivo, el registro de servicios realiza la búsqueda y devuelve una lista de descriptores de funcionalidades de servicios que encajan con la búsqueda. Este resultado incluye la información de evaluación que permite ordenar las funcionalidades encontradas de acuerdo con su grado de similitud con el objetivo. Cualquier implementación concreta de un registro de servicios para la arquitectura HI3 debe cumplir con estas especificaciones, que son proporcionadas por un API del framework Broccoli.

En la Figura 5.15 se muestra una visión general de la estructura de un registro de servicios concreto propuesto para la arquitectura HI3. Una de las características adicionales que se buscan es que un registro de la arquitectura HI3 debe fomentar la interoperabilidad entre servicios que utilizan distintas tecnologías. Gracias a que el modelo de servicios BSDM se ha diseñado como un meta-modelo con capacidad de integrar a otros modelos de servicios, el registro de servicios puede trabajar

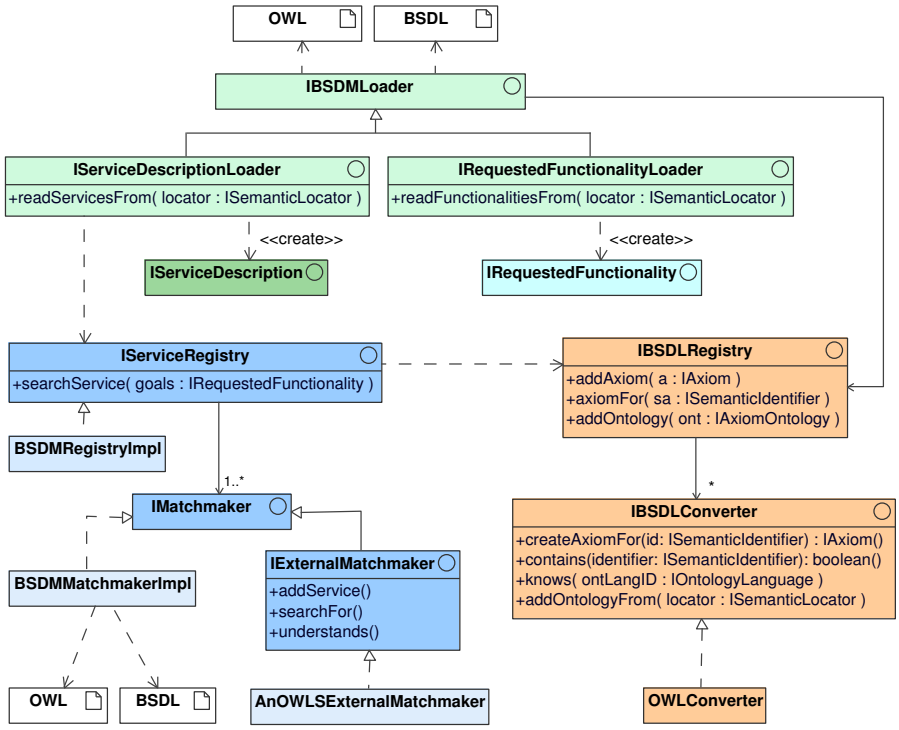



Figura 5.15: Estructura del registro de servicios.

transparentemente con servicios originalmente descritos en diferentes modelos y que una vez cargados en la arquitectura HI3 se abstraen bajo el modelo BSDM. En concreto, en el estado actual de desarrollo de la arquitectura, el registro de servicios soporta tanto el modelo propio BSDM como servicios descritos en OWL-S (que recordemos utilizan anotaciones semánticas en el lenguaje de ontologías OWL). Se utiliza un componente para el procesamiento y carga tanto de descriptores de servicios completos (ServiceDescription) como de descriptores de objetivos de búsqueda (RequestedFunctionality). Dicho componente soporta tanto descriptores en lenguaje BSDL como en lenguaje OWL, e incluso están soportados descriptores de servicios en BSDL con referencias a conceptos de ontologías OWL (gracias a las capacidades de BSDL para la integración de otros lenguajes de ontologías).

En cuanto al proceso de *matchmaking*, el registro utiliza un algoritmo propio que permite realizar la comparación tanto entre descriptores que tienen referencias a conceptos de ontologías BSDL como de ontologías OWL. Se ofrece también un mecanismo para la inclusión de otros *matchmakers* externos existentes, por ejemplo desarrollados explícitamente para servicios OWL-S que incluyan algoritmos con capacidades de razonamiento más complejas [OWLS-MX, 2008] [OWLS-SLR, 2008].

Listado 5.7: Ejemplo de un objetivo de búsqueda descrito en BSDL.

```
<import prefix="example" to=http://hi3project.com/broccoli/examples/example01"/>

<instance of="http://hi3project.com/broccoli/bsdm/profile#requestedFunctionality"
        URI="example#requestWithPreferred">

   <with property="output">
      <with property="name" value=Property1"/>
      <with property="type" valueURI="example/ontology#GeneralConcept"/>
   </with>

   <with property="preferred">
      <with property="output">
         <with property="name" value="Property1"/>
         <with property="type" valueURI="example/ontology#AMoreSpecificConcept"/>
      </with>
   </with>

</instance>
```

El *Matchmaker* y *Ranker* propuestos para la arquitectura HI3 explotan especialmente las capacidades descriptivas del lenguaje BSDL y del modelo de servicios BSDM. En concreto, las características más destacadas soportadas para las búsquedas de servicios BSDM son las siguentes:

- Tratamiento de jerarquías de conceptos en todos los elementos de una funcionalidad semánticamente anotados (entradas, salidas, categoría y propiedades no-funcionales). En la evaluación se da una puntuación menor cuanto más lejos está el concepto buscado del ofrecido dentro de una jerarquía.
- Soporte para parámetros obligatorios y opcionales. Una RequestFunctionality en BSDM puede especificar que algunos de los elementos (aplicable a entradas, salidas y propiedades no-funcionales) deseados para una funcionalidad son opcionales, es decir, si no están presentes no se invalida la funcionalidad para la búsqueda. Una funcionalidad que ofrezca elementos opcionales incrementará su puntuación en la evaluación.
- Soporte para la especificación de elementos preferidos. Esta opción se combina con las jerarquías de conceptos, de modo que es posible especificar como requisito obligatorio un concepto más general y al mismo tiempo indicar que se tiene preferencia por un concepto más específico (aplicable a entradas, salidas y propiedades no-funcionales). Se dará una puntuación mayor a una funcionalidad que proporcione exactamente el concepto preferido, pero no se invalidará una funcionalidad que proporcione uno más general.

En el Listado 5.7 se muestra un sencillo ejemplo de la descripción en BSDL de un objetivo de búsqueda para una funcionalidad en la que se ilustra el uso de la posibilidad de indicar una preferencia en una salida de la funcionalidad. Tanto el

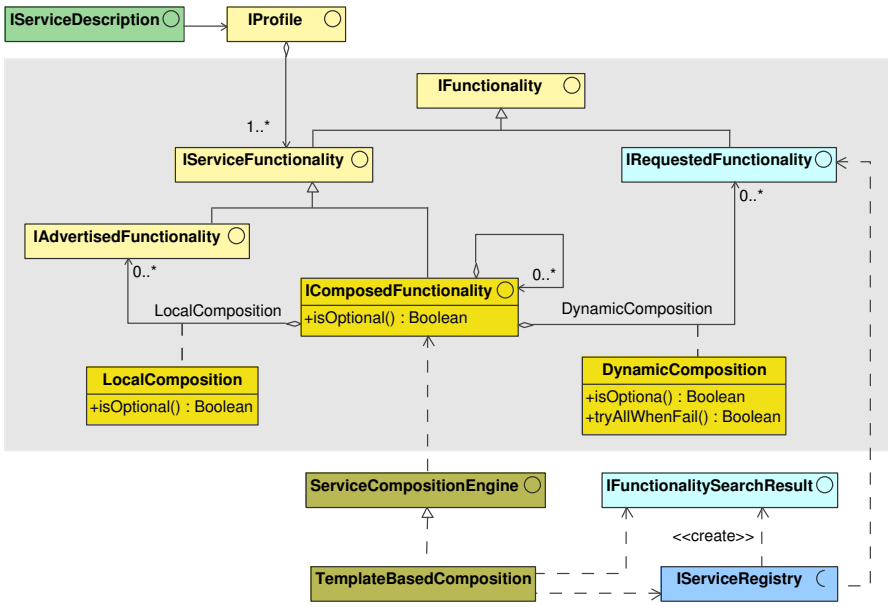



Figura 5.16: Funcionalidades compuestas en el modelo BSDM.

concepto más general como el más específico forman parte de una jerarquía descrita en una ontología del dominio.

## 5.5. Composición dinámica de servicios

Como se ha expuesto repetidamente a lo largo de esta tesis, uno de los objetivos últimos de construir servicios interoperables, accesibles y con sus capacidades semánticamente descritas es que éstos puedan exhibir un comportamiento autónomo, puedan adaptarse dinámicamente a los recursos existentes en el entorno y sean capaces de colaborar para resolver tareas. En definitiva, estamos hablando de componentes funcionales que van a depender de otros componentes que no conocen de antemano para completar sus tareas. La naturaleza espontánea y no prefijada de esta colaboración es la que va a permitir a estos componentes adaptarse al contexto y al entorno. En buena parte, estas capacidades ya se consiguen con los elementos de la arquitectura HI3 descritos hasta el momento, en particular gracias a un modelo de servicios semánticos interoperables y a un registro que facilita la búsqueda dinámica de los mismos. En esta sección veremos como se pueden ampliar estas capacidades proponiendo estrategias para la orquestación de servicios.

Como se describió en el capítulo 3, las tecnologías de servicios semánticos que han alcanzado un mayor grado de madurez han prestado, sin embargo, una escasa

atención a los aspectos relacionados con la composición dinámica de servicios. Además, lenguajes, como BPEL, diseñados para especificar composiciones de servicios han tenido resultados poco fructíferos a la hora de integrarlos en dichos modelos de servicios semánticos. No obstante, también hemos visto como desde otras aproximaciones y otras áreas se sigue trabajando activamente en este aspecto y se han conseguido diversos resultados relevantes.

Consecuentemente con todo lo anterior, en la arquitectura HI3 se pretende, en primer lugar, dar un soporte básico desde el modelo de servicios BSDM para la descripción de funcionalidades compuestas, incluyendo tanto la posibilidad de composición estática como dinámica. Con esto se abre la puerta a que sobre el modelo BSDM se desarrollen e integren mecanismos más avanzados para la orquestación y coreografía de servicios, tanto partiendo de desarrollos propios como de la integración de aproximaciones externas. En segundo lugar, se pretende definir alguna estrategia concreta más avanzada que explote las características de la arquitectura orientada a servicios HI3 para realizar una orquestación dinámica de servicios.

Relacionado con el primer objetivo, en la Figura 5.16 podemos ver como en el perfil de un servicio del modelo BSDM se incluye un tipo especial de funcionalidad que permite describir funcionalidades compuestas (ComposedFunctionality). Esta funcionalidad compuesta se define recursivamente, pudiendo contener además de otras funcionalidades compuestas, funcionalidades simples de dos tipos: funcionalidades locales (AdvertisedFunctionality) del propio servicio dónde está definida la funcionalidad compuesta, y funcionalidades expresadas como un objetivo de búsqueda de funcionalidad (RequestedFunctionality). Este último elemento permite especificar como parte de la composición requisitos funcionales que necesariamente tendrán que ser resueltos dinámicamente en el momento de la ejecución de la funcionalidad compuesta. Además, se permite especificar que alguno de los elementos de la composición sea opcional.

Sobre este modelo de composición así descrito, se pueden desarrollar distintos algoritmos o estrategias para ejecutar y planificar una composición concreta de este tipo. La arquitectura HI3 proporciona una interfaz a través de la que integrar motores de composición compatibles con el modelo BSDM. Además, se propone un motor de composición propio, cuyo funcionamiento se ejemplifica en el esquema de la Figura 5.17. El motor de composición entra en funcionamiento en tiempo de ejecución en el momento en que un servicio recibe una petición de invocación de una funcionalidad compuesta. Por tanto es necesario que previamente se haya definido en el descriptor del servicio una funcionalidad compuesta, que actuará a modo de plantilla de composición. Como ya hemos adelantado, las composiciones así descritas pueden considerarse híbridas, dado que admiten la inclusión tanto de funcionalidades estáticas como de objetivos que habrán de rellenarse en tiempo de ejecución con funcionalidades disponibles en el entorno. La Figura 5.17 representa el proceso dinámico de ejecución de una composición sobre la estructura estática

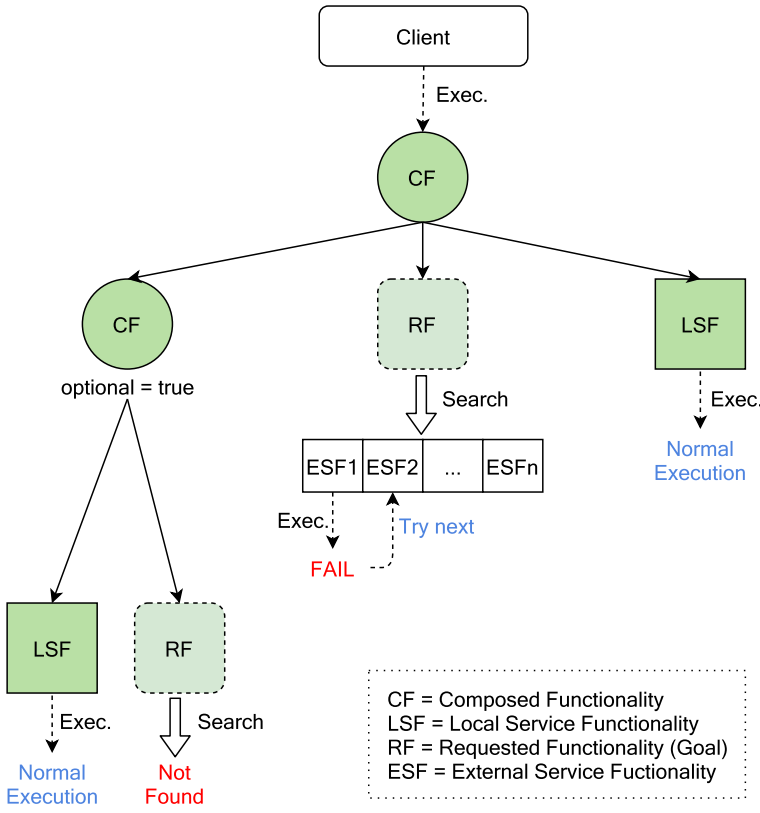



Figura 5.17: Modelo de ejecución de una funcionalidad compuesta.

de una funcionalidad compuesta previamente descrita en un modelo BSDM. En concreto, el motor de composición se guía por las siguientes reglas:

1. Cuando se encuentra con una funcionalidad compuesta, tratará de ejecutar en orden todas las funcionalidades que la componen.

2. Con carácter general, la ejecución de una funcionalidad puede ser exitosa o puede fallar (producir como resultado una excepción, es decir, un resultado RuntimeCondition del modelo BSDM).

3. En el caso de que la ejecución de una funcionalidad de la composición falle se contemplan dos posibilidades: i) la funcionalidad estaba descrita como obligatoria y en consecuencia se aborta la ejecución global de la composición; ii) la funcionalidad estaba descrita como opcional y en consecuencia puede continuarse con la ejecución de la composición.

4. Cuando el motor de composición se encuentra con una funcionalidad de tipo RequestedFunctionality (un objetivo), éste debe ejecutar una búsqueda

correspondiente a dicho objetivo a través del registro de servicios, para intentar encontrar en el entorno funcionalidades que satisfagan dicho objetivo. Si no se encuentra ninguna funcionalidad se aplica la regla número 3. En el caso contrario, es decir, el registro devuelve una lista puntuada y ordenada de funcionalidades que satisfacen el objetivo, se contemplan a su vez varias posibilidades:

*a*) Se ejecuta la funcionalidad mejor puntuada satisfactoriamente y se continua con la ejecución de la composición.

*b*) Se ejecuta la funcionalidad mejor puntuada y produce un resultado de excepción (RuntimeCondition). En este caso se continúa probando el resto de funcionalidades hasta encontrar una que no produzca como resultado una excepción.

*c*) Todas las funcionalidades producen un resultado de excepción. En este caso se aplica la regla número 3.

Como hemos dicho, sobre este mismo modelo de composición se podrían aplicar otro tipo de estrategias y algoritmos. Por ejemplo, algoritmos que aprendan de la interacción entre servicios y traten de extraer automáticamente estructuras de composición que permitan resolver tareas colaborativas.

## 5.6. Contenedores de servicios

A lo largo de este capítulo se han ido describiendo, de forma progresiva, elementos que permiten construir la base de la arquitectura HI3, fundamentándola en principios de las arquitecturas SOA que han sido extendidos para satisfacer requisitos adicionales de los entornos de Computación Ubicua. Así, se han propuesto elementos que permiten construir servicios semánticamente descritos e interoperables, elementos que permiten abstraer el conocimiento en diversos tipos de ontologías, elementos que permiten construir componentes capaces de adaptarse al entorno e incluso elementos que definen como deben construirse los registros de servicios que posibilitan el descubrimiento de servicios presentes en el entorno. En general, todos estos elementos se agrupan formando un framework para la construcción de servicios semánticos que, como introdujimos en el capítulo 4, hemos denominado framework Broccoli. Ahora nos centraremos en un nuevo tipo de elemento que aportará mayor estructura a la arquitectura HI3, integrando y enriqueciendo los elementos anteriores. Nos referimos a los contenedores de servicios, que ya fueron introducidos conceptualmente en el capítulo 4, y que proporcionan plataformas para la ejecución y gestión del ciclo de vida de servicios HI3 en diferentes entornos computacionales. Estos contenedores se agrupan en el framework que hemos

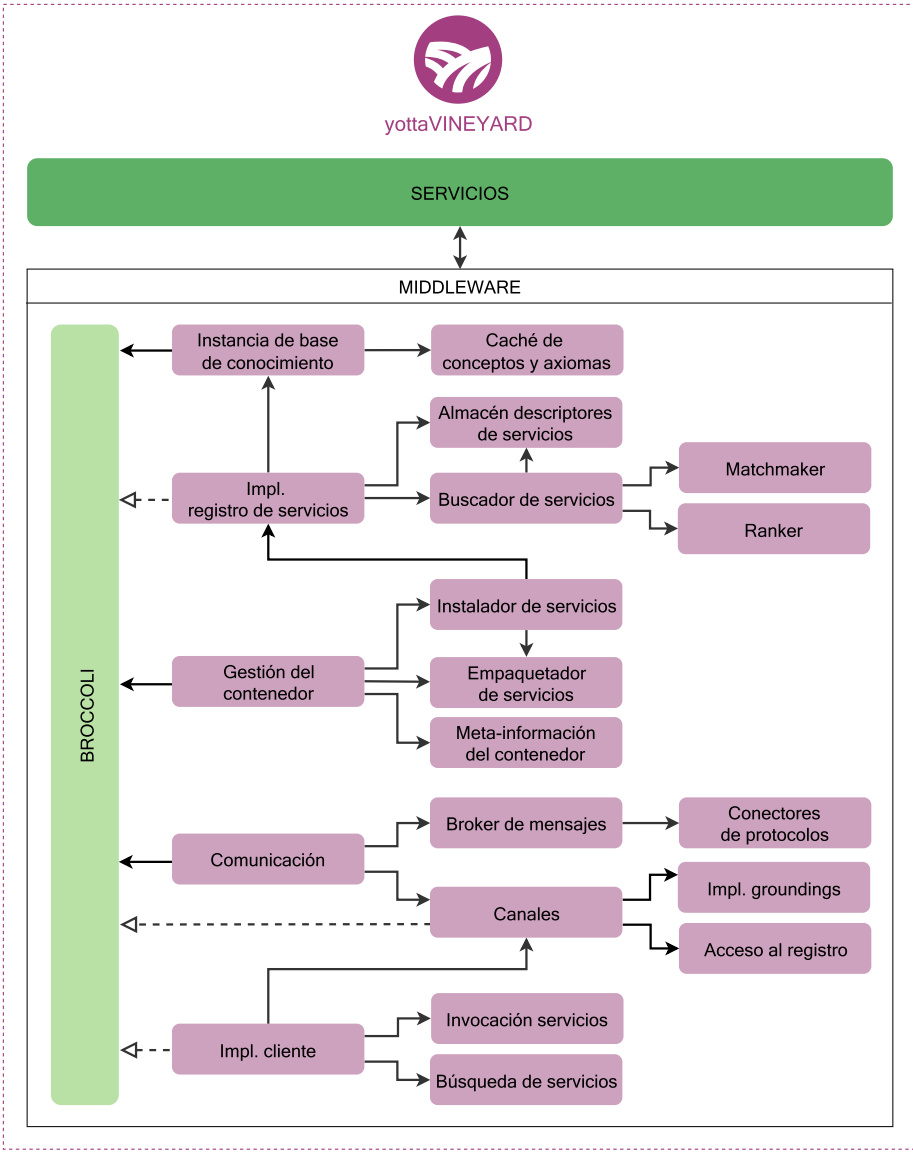



Figura 5.18: Principales componentes software de un contenedor yottaVineyard.

denominado Vineyard, y que además es el responsable de proporcionar implementaciones en tecnologías concretas para aquellas APIs y componentes abstractos del framework Broccoli que lo requieren, como son las APIs del registro de servicios y el grounding abstracto basado en canales de mensajería asíncrona.

Desde un punto de vista lógico, la arquitectura HI3 debe verse como una organización de servicios ubicuos que aparecen y desaparecen, que se adaptan al en-

torno, que se encuentran y colaboran con independencia de su localización física, y que proporcionan soluciones adaptadas a los requisitos de IAm. De esta forma, los desarrolladores de nuevos servicios y aplicaciones tendrán una visión lógica de los sistemas de IAm, que les permitirá centrarse en la construcción de funcionalidades abstrayéndose en parte de la complejidad del entorno. Para conseguirlo, es necesario, completar las capacidades de la arquitectura HI3 descritas hasta ahora de dos formas: i) integrando los elementos constructivos que componen el middleware orientado a servicios de la arquitectura y automatizando en la medida de lo posible todos los procesos relacionados con el despliegue y ejecución de servicios; ii) soportando la ejecución de los elementos funcionales de la arquitectura (servicios y aplicaciones) en la mayor variedad de entornos físicos posible. Para el primer objetivo se desarrolla el concepto de contenedor de servicios Vineyard. Para el segundo objetivo se proponen diferentes tipos de contenedores de servicios que faciliten su ejecución en entornos computacionales con distintas capacidades y que además pueden ser implementados sobre diferentes lenguajes y plataformas. A través de múltiples instancias de estos contenedores puede crearse una red física de contenedores, que, sin embargo, será percibida como un único entorno lógico de ejecución de servicios. Sobre esta meta-arquitectura de contenedores de servicios y aplicaciones los desarrolladores encontrarán las facilidades necesarias para distribuir sus componentes de IAm de forma ubicua.

En la Figura 5.18 se ilustra la estructura de un contenedor de servicios con las máximas capacidades, que hemos denominado como contenedor yottaVineyard. Este tipo de contenedor debe obligatoriamente ofrecer una implementación de un componente Registro de Servicios. Además, debe soportar la asociación dinámica con otros contenedores remotos, de modo que, a través de la federación de sus registros, se consiga crear una distribución dinámica y descentralizada de registros de servicios en un espacio lógico único. Más en concreto, proporciona los siguientes componentes y capacidades:

- Una instancia de la base de conocimiento. Se realiza como una instancia del registro de axiomas del lenguaje BSDL (BSDLRegistry) proporcionado por el framework Broccoli. Facilita la gestión en tiempo de ejecución de una caché de axiomas y ontologías especificados en distintos lenguajes. El contenedor proporciona una instancia común del componente BSDLRegistry para todos los servicios que se están ejecutando en él.

- Implementación de un registro de servicios con realizaciones concretas de *matchmaker* y *ranker* para facilitar el descubrimiento dinámico de servicios. La instalación de un nuevo servicio (un servicio BSDM o un servicio en otra tecnología adaptado a BSDM) en el contenedor supondrá que éste lo incluya automáticamente en el registro de servicios.

- Realización concreta de las capacidades de comunicación requeridas por los

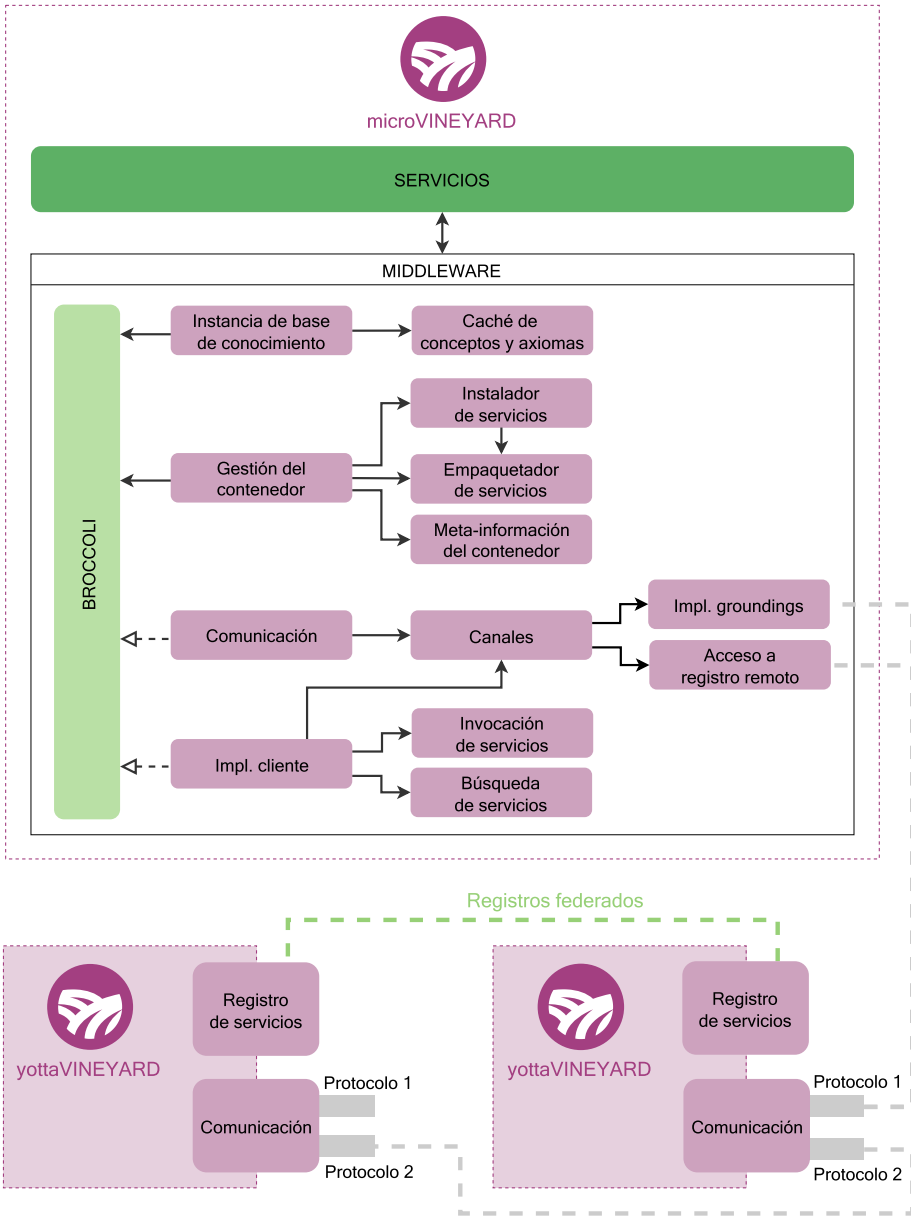



Figura 5.19: Arquitectura de un contenedor microVineyard.

servicios y por el registro. Como ya hemos adelantado previamente, el contenedor proporcionará implementaciones concretas sobre protocolos de comunicación comunes del grounding abstracto definido por el framework Broccoli. De esta forma, los servicios desplegados en un contenedor podrán utilizar directamente cualquiera de estos groundings con tan solo indicarlo en su

descriptor semántico. De forma similar, un contenedor resuelve el modelo estándar de interacción con el registro sobre algún protocolo común concreto. Además, el contenedor debe proporcionar un mecanismo estándar para gestionar la federación dinámica de registros entre contenedores. Dado que ya es necesario que los contenedores proporcionen uno o más protocolos de comunicación concretos que soporten el modelo de mensajería asíncrona definido para los groundings de servicios, se resolverán sobre esos mismos protocolos los aspectos concretos relacionados con la comunicación con los registros y entre registros federados. Para ello, el contenedor proporciona un componente Broker, correspondiente al patrón del mismo nombre en un Middleware Orientado a Mensajes [Curry, 2004] [Hohpe and Woolf, 2003], que puede ser integrado con distintos protocolos de comunicación comunes.

- Acceso al API de cliente para la invocación de funcionalidades de servicios y para la búsqueda en el registro. La invocación de una funcionalidad de un servicio se resuelve directamente a través de un descriptor de servicio BSDM, siendo la única aportación del contenedor los groundings concretos sobre los que se realiza el modelo estándar de interacción basado en mensajería asíncrona. En el caso del acceso al registro, el contenedor es responsable de resolver transparentemente la comunicación con un registro de servicios respetando el modelo estándar de interacción con el registro que se ha definido para la arquitectura HI3.
- Gestión del propio contenedor y del ciclo de vida de los servicios instalados en él. Todo contenedor Vineyard puede proporcionar alguna información básica sobre sí mismo, referente al lenguaje en que está implementado, un número de versión o una descripción textual, junto con información también básica de los servicios que en él hay instalados. Además, todo contenedor debe proporcionar herramientas para lanzar su ejecución en los entornos computacionales para los que fue implementado. En lo referente a los servicios, un contenedor también proporciona herramientas para la instalación de servicios y para la posterior ejecución de los mismos.

En la Figura 5.19 se presenta la organización en componentes del segundo tipo de contenedor propuesto, denominado microVineyard. Se puede observar como se ha prescindido de los componentes que implementan el registro y el broker de mensajes pero se mantiene el cliente que facilita la interacción con un registro disponible en otro contenedor. Los servicios del microVineyard utilizarán de forma transparente un *proxy* para acceder al registro remoto de una plataforma yottaVineyard, como si éste estuviese disponible localmente. Así, cuando se instala un nuevo servicio en este tipo de contenedor, realmente se registra de forma transparente en el registro de servicios del contenedor yottaVineyard del que es dependiente. En el diagrama también se ilustra como, de forma transitiva, gracias al mecanismo de federación entre registros de distintas plataformas, un servicio puede descubrir otros servicios

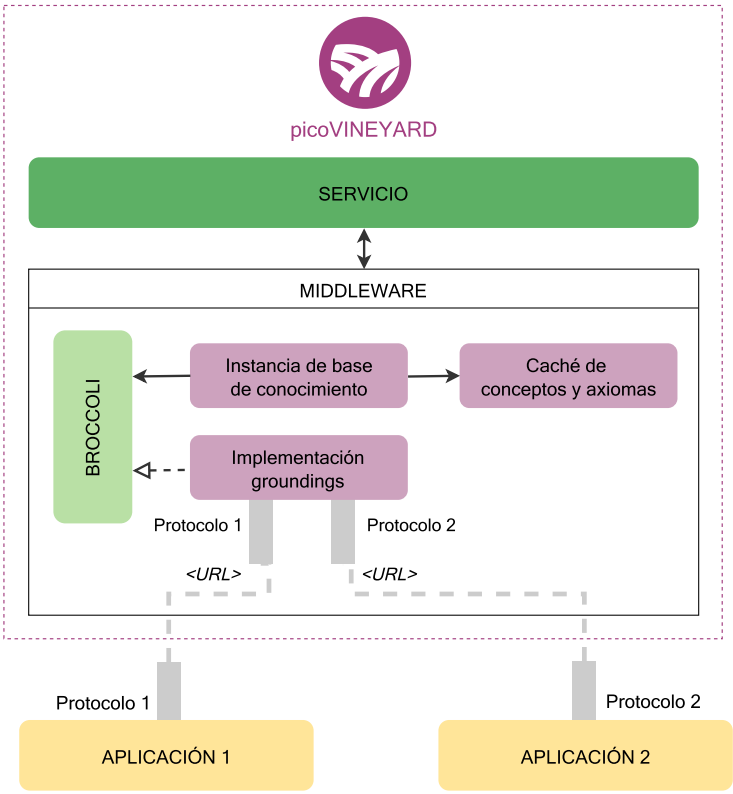



Figura 5.20: Arquitectura de un contenedor picoVineyard.

de terceras plataformas. Se trata por tanto de un contenedor que permite una implementación notablemente más ligera y que no requiere dependencias con una tecnología que proporcione un broker de mensajería. Por lo tanto, será más apto para su implementación en plataformas con pocos recursos y en dispositivos con acceso a un grupo de tecnologías más cerrado, como es el caso de *smartphones*.

Por último, en la Figura 5.20 se esquematiza la estructura de un contenedor picoVineyard. Facilita el despliegue de un único servicio semántico, compatible con las especificaciones HI3 y con mínimas dependencias, en multitud de plataformas. Así, por ejemplo, una implementación en lenguaje Javascript del framework Broccoli y un sencillo contenedor picoVineyard, podría permitir el despliegue de un servicio HI3 en prácticamente cualquier sistema, incluida la posibilidad de incrustar un servicio de este tipo en la Web. Al no disponer ni de un registro de servicios local ni de relación con otro contenedor que lo proporcione, se comporta como un servicio aislado hasta que se haga accesible su descriptor a potenciales clientes para que lo utilicen. No obstante, cualquier componente compatible con las especificaciones HI3 (especificaciones descritas en el Apéndice A) será capaz de interaccionar con

él a partir de únicamente su descriptor semántico especificado en lenguaje BSDL, en el que se indique la URL del grounding o los groundings que está utilizando. En este sentido, se comportaría de forma similar a un Servicio Web estándar accesible a partir del conocimiento de su descriptor WSDL.

Como parte de este trabajo se han realizado implementaciones completas de referencia de los tres tipos de contenedores, incluyendo implementaciones en la plataforma Java de los tres contenedores e implementaciones en la plataforma Android de los contenedores microVineyard y picoVineyard.

En los subapartados siguientes se proporciona un mayor detalle de los componentes de los contenedores Vineyard relacionados con el empaquetado/despliegue de servicios y con la implementación de la comunicación sobre protocolos y tecnologías concretas.

### 5.6.1. Empaquetado y despliegue de servicios

El objetivo primero de empaquetar un servicio es que sea posible contener en un sólo archivo lógico un servicio completo, de forma que se facilite su distribución y que contenga todo lo necesario para que un contenedor de servicios pueda desplegarlo y gestionar su ciclo de vida.

Así, en el caso de la arquitectura HI3 se propone un modelo para empaquetar los servicios BSDM consistente en un fichero comprimido en formato ZIP, con extensión .BSD y que tiene la siguiente estructura interna:

- El descriptor del servicio ubicado en el primer nivel del paquete.
- Un directorio *ontologies* conteniendo las ontologías referenciadas por el servicio, descritas en lenguajes soportados por la arquitectura HI3 (actualmente BSDL y OWL).
- Un directorio *implementation* conteniendo la implementación (código compilado, librerías, etc.) del servicio.

Partiendo de un servicio empaquetado, el proceso de despliegue en un contenedor Vineyard consiste en validar, desempaquetar, y cargar en memoria los contenidos del servicio. Para ello, se utilizará un componente del contenedor (ServiceInstaller) que hará uso de las siguientes herramientas:

- Un componente ServiceDeployer, encargado de todas las operaciones en las que se validan y desempaquetan servicios.
- Un componente ServiceDescriptionLoader, que cargará en memoria las descripciones de servicios a partir de descriptores soportados (actualmente descriptores de servicios BSDM y OWL-S).

- Un componente DocumentLoader, responsable de cargar en memoria los axiomas de otras ontologías referenciadas en los descriptores de servicios.
- Una jerarquía de componentes especializados en cargar las implementaciones concretas de groundings.

Además, para facilitar la instalación y ejecución automatizada de servicios en un contenedor, el componente ServiceInstaller es capaz de instalar y cargar todos los servicios empaquetados situados en un directorio configurado como directorio de despliegue del contenedor.

### 5.6.2. Resolución de los modelos de comunicación a través de tecnologías concretas

Recordemos que el modelo de servicios BSDM proporcionado por el framework Broccoli proporciona un modelo abstracto de interacción basado en el concepto de canales de mensajería asíncrona que puede ser realizado a través de groundings concretos sobre distintos protocolos existentes. De esta forma se abstrae la interacción entre servicios de la utilización de protocolos concretos y se posibilita incluso que un mismo servicio proporcione más de un grounding concreto de forma simultánea, aumentando sus capacidades para la interoperabilidad. Asimismo, se definió un modelo de interacción que da cobertura tanto al modo petición/respuesta como al modo publicación/subscripción, soportando en total cuatro tipos de interacción con una funcionalidad de un servicio.

Teniendo en cuenta los requisitos del modelo de interacción definido para los servicios HI3, se hace una evaluación de las tecnologías y protocolos existentes más adecuados para implementar groundings concretos de servicios BSDM. Además, se establecen algunos requisitos adicionales: i) se considera fundamental dar cobertura al mayor número posible de plataformas de ejecución, incluyendo dispositivos móviles; ii) deben soportarse diferentes ámbitos de comunicación comunes, como redes de datos locales, Internet o redes de datos de telefonía móvil.

Dado que se ha propuesto un modelo basado en patrones de mensajería asíncrona, el primer paso es seleccionar una tecnología que realice la función de *broker*, a través del que se gestionarán los canales para la comunicación entre clientes y servicios. Existen numerosas implementaciones de brokers de mensajería que soportan un abanico amplio de opciones. Por su amplia implantación y madurez de desarrollo se seleccionó el broker ActiveMQ [ActiveMQ, 2015].

En lo referente a protocolos de comunicación de mensajería, existe un amplio abanico de protocolos ampliamente utilizados y con implementaciones en múltiples plataformas: AMQP, MQTT, OpenWire, REST, STOMP, etc. Estos protocolos a su vez se encapsulan a través de algunos protocolos comunes más generales, como TCP,

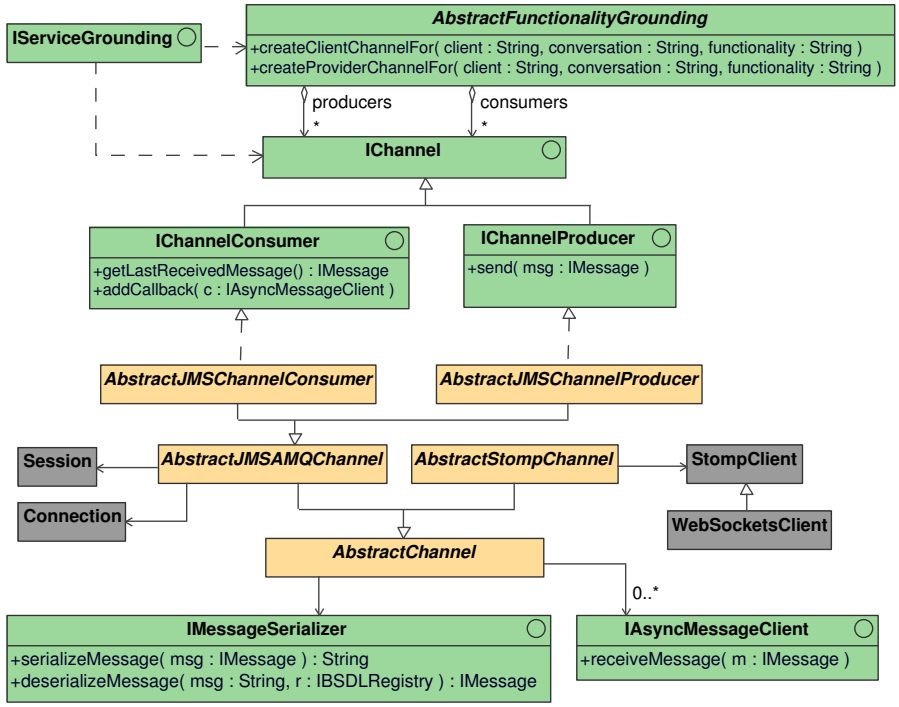



Figura 5.21: Implementaciones de groundings concretos sobre varios protocolos de comunicación existentes.

SSL sobre TCP, HTTP, HTTPS, Websockets o NIO. En concreto se han realizado tres desarrollos completos de grounding basado en mensajería asíncrona soportados por diferentes tecnologías:

- Grounding JMS. JMS es una especificación parte de Java EE y constituye un middleware que permite enviar mensajes de forma asíncrona entre distintos clientes que forman parte de una aplicación distribuida [JMS, 2015].
- Grounding STOMP. STOMP es un protocolo sencillo y ligero de mensajería, basado en texto plano, para una comunicación asíncrona entre clientes interconectados mediante un broker. Existen implementaciones de clientes en la mayoría de lenguajes y plataformas. Bajo una perspectiva de eficiencia en el tráfico de mensajes es una alternativa adecuada para dispositivos móviles con pocos recursos y con dependencia de una batería.
- Grounding STOMP-WebSockets. WebSockets es una tecnología que permite multiplexar varios canales de comunicación sobre un único puerto TCP [WebSockets, 2015]. Su desarrollo forma parte de HTML5 y supone un gran avance para facilitar las comunicaciones bidireccionales en la Web. El grounding propuesto es una particularización del grounding STOMP que se realiza a tra-

vés de una conexión WebSockets. Se trata de una solución de actualidad que además evita problemas con firewalls en comunicaciones a través de Internet.

En el diagrama 5.21 se aporta una visión simplificada de como se implementan los groundings concretos, proporcionando implementaciones para las interfaces que representan los canales de comunicación en el grounding abstracto que proporciona el framework Broccoli (cuya descripción vimos en el apartado 5.3.2 de este capítulo). Cada tipo de grounding utiliza una URL con un prefijo distinto para indicar el acceso al grounding de un servicio en su descriptor (tcp:// para JMS, stomp:// para STOMP y ws:// para WebSockets), de modo que con dicha URL se disponga de toda la información necesaria para saber que tipo de grounding es necesario cargar para un servicio concreto.

Por otro lado, los contenedores Vineyard utilizan el mismo tipo de solución para resolver la comunicación necesaria entre registros de servicios de distintos contenedores. Como ya se explicó, se presentan dos casos: i) federación entre registros completos para crear una red distribuida de registros; ii) relación de dependencia entre un registro proxy y un registro real (la que se da entre un contenedor microVineyard y un contenedor yottaVineyard). Así, sobre el modelo general de comunicación basado en canales de mensajería asíncrona, se definen dos nuevos protocolos lógicos de mensajes (la especificación completa de estos protocolos se encuentra en sendos apartados del Apéndice A):

- Un protocolo para solicitar remotamente que se añada un nuevo descriptor de servicio a un registro. Es utilizado, por ejemplo, por un contenedor microVineyard para añadir los descriptores de sus servicios al contenedor con el que está asociado.
- Un protocolo para el envío de consultas de búsqueda a un registro remoto y la recepción de la repuestas a dicha consulta. El protocolo soporta la recursividad de las búsquedas dentro de una red de registros federados.

Para el establecimiento de la conexión que permite federar a dos contenedores, éstos deben tener conocimiento de sus URI de conexión respectivas y de los protocolos concretos soportados. En las realizaciones concretas de contenedores Vineyard se soporta JMS-OpenWire, STOMP y STOMP sobre WebSockets.

## 5.7. Implementación de referencia

La implementación de referencia del middleware orientado a servicios, que constituye la base de la arquitectura HI3, ha sido desarrollado principalmente utilizando los lenguajes Java y XML. Además, se han realizado implementaciones de todos los tipos de contenedores Vineyard para la plataforma Java y también se han

realizado implementaciones de los contenedores microVineyard y picoVineyard para la plataforma móvil Android.

Como ya se ha mencionado, estas implementaciones se organizan en torno a dos frameworks: Broccoli y Vineyard. Creemos que es importante hacer que estén disponibles los resultados de investigación para que el resto de la comunidad científica pueda validarlos y realizar aportaciones a los mismos. En consecuencia, la implementación de ambos frameworks se encuentra disponible en repositorios públicos de GitHub y publicados bajo una licencia GNU Affero GPL v3. En concreto, el framework Broccoli está accesible en https://github.com/GII/broccoli/ y el framework Vineyard en https://github.com/GII/vineyard/. Además, se ha tenido especial cuidado en que todas las dependencias de tecnologías externas utilizadas en las implementaciones sean accesibles y estén convenientemente mantenidas y actualizadas. Así, tanto la implementaciones binarias de nuestros propios frameworks como las de librerías externas utilizadas se encuentran disponibles a través del repositorio Maven Central (http://search.maven.org/).

Por último, en el Apéndice A se incluyen las especificaciones imprescindibles para realizar nuevas implementaciones HI3-compatibles en diferentes lenguajes y plataformas.

## 5.8. Resumen

A lo largo de este capítulo se han descrito los principales elementos que permiten construir la base orientada a servicios de la arquitectura HI3. El objetivo último es que las capacidades en este nivel de la arquitectura faciliten a los desarrolladores la construcción de nuevos servicios reutilizables, interoperables, adaptables, colaborativos y ubicuos, con independencia de las tecnologías concretas con las que internamente implementen su funcionalidad.

En los aspectos SOC de la arquitectura HI3 se ha tratado de avanzar hacia una visión de este paradigma más próxima a la Computación Ubicua. En particular, se ha puesto el énfasis en los siguientes aspectos: i) paliar el problema de la heterogeneidad a diferentes niveles (lenguajes de representación del conocimiento, modelos de descripción de servicios, modelos y protocolos para el acceso e interacción entre servicios, etc.); ii) proponer una visión ubicua en torno a servicios colaborativos que posibilitan su ejecución en el mayor número posible de plataformas, más allá de la preponderancia actual de las tecnologías de Servicios Web; iii) ampliar las capacidades descriptivas de los modelos clásicos de descripción de servicios.

Como núcleo del middleware orientado a servicios se ha definido un modelo avanzado de servicios semánticos (BSDM) que trata de abstraer capacidades de otros modelos existentes e incorporar otras nuevas. Por un lado, BSDM incorpo-

ra un modelo de interacción avanzado independiente de protocolos de comunicación, así como capacidades para el tratamiento de resultados de error, descripción de múltiples capacidades funcionales y descripción de propiedades no-funcionales aplicables a diferentes niveles. Por otro lado BSDM actúa como un meta-modelo al que se pueden traducir conceptos de otros modelos existentes, abstrayendo al desarrollador de la heterogeneidad de modelos de descripción de servicios semánticos. Para ilustrar este concepto también se ha incluido soporte para el modelo de servicios semánticos OWL-S bajo la abstracción del modelo BSDM.

Junto con el modelo BSDM también se ha definido un lenguaje para la especificación de servicios y ontologías (BSDL). Como ocurre con BSDM, este lenguaje también cumple una doble función dentro del middleware orientado a servicios HI3. En primer lugar, facilita la descripción natural de servicios y ontologías gracias a que proporciona una sintaxis más estructurada y próxima a los conceptos de la programación orientada a objetos, a cambio de sacrificar parte de la expresividad de lenguajes basados en lógicas descriptivas como OWL. En segundo lugar, permite integrar conceptos descritos en otros lenguajes como parte de una misma ontología BSDL. Esto último posibilita, por ejemplo, realizar una descripción semántica de un servicio más legible y estructurada en lenguaje BSDL, dónde los conceptos del dominio manejados por el servicio hacen referencia una ontología OWL.

Asimismo, también se han añadido al modelo BSDM capacidades y mecanismos para describir y ejecutar funcionalidades compuestas que se resuelven en tiempo de ejecución en base al descubrimiento, guiado por objetivos, de otras funcionalidades disponibles en el entorno.

Para facilitar la creación de redes lógicas de servicios ubicuos, capaces de colaborar con independencia de que se ejecuten en entornos físicos con capacidades computacionales muy distintas, se propuso también un conjunto de contenedores de servicios capaces de ejecutar instancias de la arquitectura HI3 con diferentes capacidades, pero manteniendo la compatibilidad entre ellas y abstrayendo a los servicios de las diferencias.

Por último, se proporciona una implementación de referencia del middleware orientado a servicios de la arquitectura HI3 que está publicado y accesible para su uso y extensión por parte del resto de la comunidad científica, junto con las especificaciones imprescindibles para realizar nuevas implementaciones HI3-compatibles en diferentes lenguajes y plataformas.

# Capítulo 6

# Una arquitectura SOA para sistemas multi-agente

## 6.1. Introducción

La evolución del software y los sistemas de comunicaciones posibilita la construcción de sistemas cada vez más complejos, más distribuidos y más flexibles. En este sentido, hay cada vez una mayor demanda para integrar plataformas y compartir recursos entre sistemas. No se trata únicamente de compartir y reutilizar recursos desarrollados en un mismo lenguaje, con un mismo paradigma y sobre una misma plataforma, sino que los desarrolladores deberían poder elegir el paradigma software más adecuado para cada problema y aún así reutilizar recursos existentes implementados en otros paradigmas. Los principios de la arquitectura HI3 están en consonancia con esta visión, y, en consecuencia, no se restringe el paradigma con el que internamente los componentes implementen su funcionalidad. Así, en este apartado se exploran las posibilidades de integración de los paradigmas SOC y Sistemas Multi-Agente y se describe una propuesta y una implementación de un contenedor de servicios HI3 en el que se utilizan sistemas multi-agente para la implementación de funcionalidades.

Como vimos en el capítulo 3, los agentes software tienen una serie de características como autonomía, proactividad, movilidad, comportamiento guiado por objetivos o capacidades sociales que los hacen especialmente adecuados para entornos dinámicos y distribuidos, dónde los agentes deben adaptarse autónomamente a las características cambiantes del entorno. No obstante, esta tecnología también se ha tenido que enfrentar a diversos problemas que han impedido que el uso de la misma se haya extendido a mayor escala [Leyton-brown, 2010]. En este sentido,

la programación de agentes suele ser compleja debido a la falta de herramientas avanzadas que faciliten el desarrollo. Otros problemas tienen que ver con la falta de mecanismos explícitos y uniformes que exploten información semántica en el proceso de comunicación entre agentes, como ocurre en las implementaciones de plataformas que siguen el estándar FIPA (del inglés, Foundation for Intelligent Physical Agents). Por todo ello y por la experiencia concreta en desarrollos con este tipo de tecnología, desde este trabajo se concluye que el paradigma puede no ser el más adecuado para el desarrollo de cualquier sistema de IAm, pero sí debería ser tenido en cuenta como método eficaz de enfrentar el desarrollo de funcionalidades concretas en este tipo de entornos cuyos requisitos encajen con los beneficios potenciales de los sistemas multi-agente.

Existen otras aproximaciones que plantean la integración del paradigma de Computación Orientada a Servicios con el paradigma de Sistemas Multi-Agente. Algunos, como el caso de la arquitectura FUSION@ [Tapia et al., 2009b] o la propuesta de [Ricci et al., 2007] buscan la integración de los protocolos de comunicación de los sistemas multi-agente con los protocolos propios de servicios web (SOAP). En otros casos se busca la integración con tecnologías de web semántica [García-Sánchez et al., 2009] [Shafiq et al., 2006]. La aproximación de este trabajo trata de explotar todas las capacidades del middleware orientado a servicios de la arquitectura HI3 para que un sistema multi-agente pueda exportar sus capacidades funcionales como un componente que es visto como un servicio semántico e interoperable. El resultado es una nueva implementación de referencia de un contenedor de servicios, compatible con las especificaciones de la arquitecura HI3, dónde la implementación interna de las funcionalidades de los servicios se hace como agentes de la plataforma Jade [JADE, 2015], y que denominaremos como *contenedor masVineyard*.

## 6.2. Integración de los paradigmas Computación Orientada a Servicios y Sistemas Multi-Agente

Asumiendo el interés del paradigma de Sistemas Multi-Agente para abordar el desarrollo de soluciones a ciertos problemas en entornos de Computación Ubicua e IAm, desde este trabajo se propone una metodología para la construcción de sistemas multi-agente integrados en una arquitectura SOA. Aunque la metodología propuesta constituye una aproximación independiente de tecnologías concretas, en este trabajo se ilustrará utilizando el middleware orientado a servicios de la arquitectura HI3 y una plataforma que cumple las especificaciones del estándar FIPA como Jade. Así, la metodología de desarrollo que se propone puede resumirse en los siguientes pasos:

1. Identificar y especificar la funcionalidad que se desea que el sistema multi-agente exporte en su interfaz de interacción con otros componentes de un sistema de IAm.

2. Diseñar e implementar el sistema multi-agente que resuelve la tarea concreta deseada a través de una sociedad de agentes colaborativos.

3. Validar el sistema multi-agente construido individualmente.

4. Diseñar e implementar el descriptor semántico de un servicio que proporcione la funcionalidad identificada en el primer paso. En nuestro caso se hará utilizando las capacidades del framework de servicios Broccoli.

5. Seleccionar un agente de la sociedad de agentes que represente al sistema multi-agente y proporcione el envoltorio de servicio semántico correspondiente al descriptor creado en el paso anterior.

6. Incorporar el componente envuelto como un servicio semánticamente descrito a un registro de servicios HI3, de modo que pase a estar disponible para ser descubierto y utilizado por otros servicios accesibles en la red.

En la Figura 6.1 se describe la arquitectura de componentes propuesta para la construcción de un contenedor masVineyard de sistemas multi-agente que cumple con las especificaciones de la arquitectura HI3. Para la realización concreta del contenedor masVineyard que se ha implementado en este trabajo se ha desarrollado una extensión de la plataforma Jade. El principal motivo para extender las capacidades de Jade es que por sí mismo no promueve ningún tipo de separación entre el código que implementa la funcionalidad y el código que lidia con los aspectos de comunicación e interacción entre agentes. Ésto hace de programar agentes una tarea costosa en tiempo, a la vez que limita la reutilización de componentes. Además, al estar únicamente orientado a gestión de agentes individuales, tampoco proporciona herramientas de alto nivel para la gestión de sistemas multi-agente como un todo. Así, el contenedor masVineyard incorpora un conjunto de modelos y herramientas que facilitan la utilización de modelos de comunicación desacoplados; aporta facilidades para la gestión de conversaciones entre varios agentes; y proporciona un modelo de sistemas multi-agente mejorado que permite la construcción declarativa de agentes en base a componentes funcionales reutilizables. Todo ello dentro de un contenedor para la ejecución de sistemas multi-agente que incorpora herramientas para la instalación y monitorización de nuevos componentes.

De nuevo, fijándonos en la Figura 6.1, una vez que el desarrollador ha completado la implementación de una funcionalidad a través de un conjunto de agentes que forman un sistema multi-agente, podrá utilizar las herramientas proporcionadas por el middleware HI3 ya conocidas, como son los frameworks Broccoli y Vineyard, para convertir su sistema multi-agente en un servicio HI3. Asimismo, el contenedor

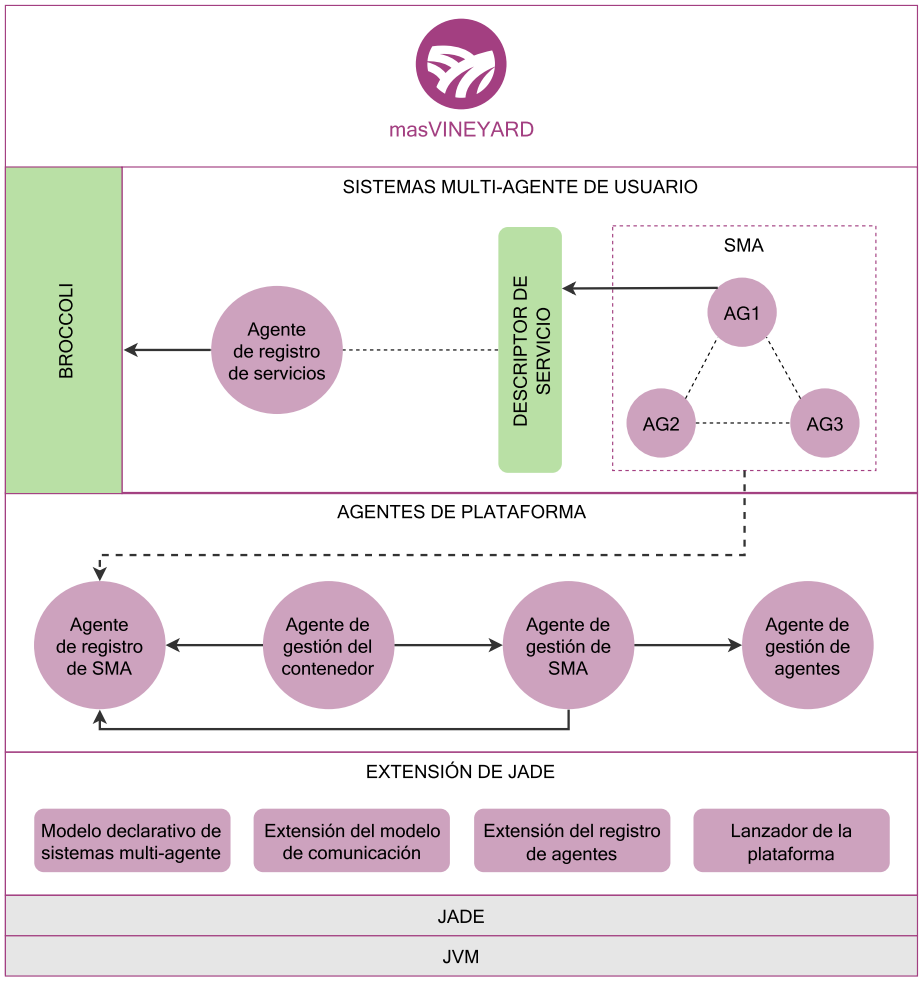



Figura 6.1: Principales componentes software del contenedor masVineyard.

masVineyard le proporciona una implementación de registro, en este caso ejecutándose como un agente, en el que podrá registrar el descriptor semántico de su nuevo servicio. Veremos además, que el proceso de ejecución y registro de un nuevo sistema multi-agente en el contenedor puede realizarse de forma automática, una vez instalado el componente a través de la herramienta proporcionada para ello.

En los siguientes subapartados se describen con más detalle los elementos principales con los que se han extendido las capacidades de la plataforma de agentes Jade con el fin de facilitar a los desarrolladores la implementación de funcionalidades utilizando el paradigma de agentes.

## 6.3. Extensión de las capacidades de comunicación entre agentes

En lo referente a las capacidades de comunicación, se ha diseñado un nuevo modelo de intercambio de mensajes entre agentes desacoplado, una extensión del modelo de conversaciones y un nuevo modelo de interacción basado en el patrón publicación/subscripción.

En términos de intercambio de mensajes entre agentes, Jade proporciona un sistema básico para envío y recepción de mensajes asíncronos que invita a los desarrolladores a programar la comunicación entre agentes mezclada con la lógica de control y de forma ad-hoc. Con el fin de limitar esta problemática se introduce un mecanismo, basado en el concepto *Procesador de Mensajes*. Un Procesador de Mensajes representa la forma más básica de que un agente reciba mensajes en este contenedor. Se implementa como un comportamiento que se puede asociar con una o más Plantillas de Mensajes en las que se especifican características que un mensaje debe cumplir. Así, cuando un agente recibe un mensaje que cumple con las especificaciones de una plantilla, éste se envía automáticamente al Procesador de Mensajes asociado para su procesamiento. Estos Procesadores de Mensajes se encargarán de ejecutar cualquier pre-procesamiento necesario a los mensajes, desacoplando esta tarea de la lógica de control del agente, y posteriormente enviarán el mensaje al comportamiento adecuado del agente.

Por tanto, las Plantillas de Mensajes asociadas con cada Procesador de Mensaje constituyen una especificación de la interfaz de comunicación que proporciona un agente en cada instante de tiempo (Procesadores de Mensajes y Plantillas pueden ser añadidas y eliminadas de un agente dinámicamente). Estos mecanismos combinados con el modelo de construcción declarativa de agentes, que se describirá más adelante, proporcionan las bases para el soporte a las comunicaciones declarativas entre agentes.

Sobre el concepto de Procesador de Mensaje, se introduce un mecanismo de planificación basado en prioridades para la recepción de mensajes. Éste proporciona a los desarrolladores la posibilidad de priorizar algunas capacidades de comunicación de un agente sobre otras, con el fin de focalizar la atención del agente en sus tareas críticas.

En lo referente a conversaciones multi-agente, la plataforma Jade proporciona soporte para algunos modos predefinidos, tanto 1-a-1 como 1-a-N, basados en los protocolos propuestos por el estándar FIPA. No obstante, en contextos complejos con una elevado número de agentes con comportamientos heterogéneos, parece difícil establecer en todos los casos una comunicación compleja entre una sociedad de agentes de una forma ya predefinida en la plataforma. En consecuencia se

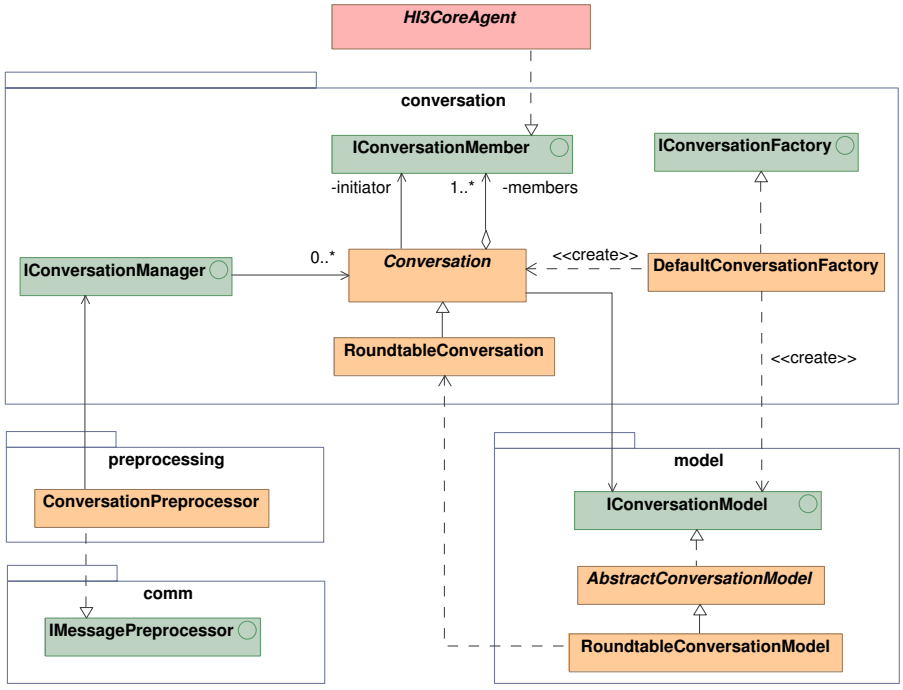



Figura 6.2: Modelo que da soporte a las conversaciones N-a-M entre agentes.

extiende el concepto de conversación como una forma de soportar interacciones N-a-M entre los agentes de una sociedad. En este modelo, cada agente tiene las capacidades inherentes de participar en una conversación existente, crear una nueva conversación, de invitar a otros agentes a una conversación, de enviar/recibir mensajes dentro de una conversación y de ser informado sobre otros agentes que se unen o dejan una conversación.

La Figura 6.2 muestra un diagrama de clases simplificado del modelo de conversaciones diseñado. Muestra como las conversaciones son controladas por un *ConversationManager* que maneja los mensajes de una conversación utilizando una implementación concreta de un *ConversationModel*. Estos modelos de conversación contienen la lógica con la que se rige un tipo concreto de conversación, incluyendo cómo invitar a miembros a la conversación, cómo abandonar una conversación o cómo notificar a los miembros de la llegada de un mensaje.

Como añadido a los modelos de comunicación previos, se ha diseñado una extensión para soportar comunicaciones de tipo publicación/subscripción. En este modelo, los agentes que desean proporcionar alguna información la publican como enventos, mientras que los agentes interesados en recibir un determinado tipo de información se subscriben a esos eventos. Un modelo de este tipo es un mecanis-

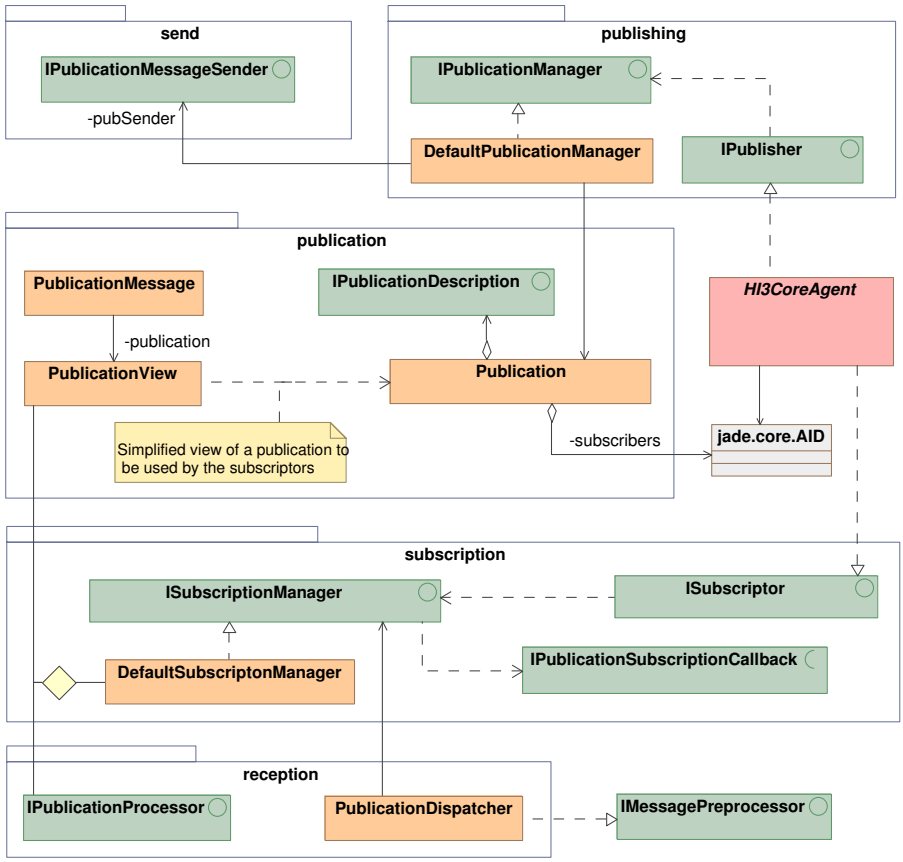



Figura 6.3: Modelo publicación/subscripción para la interacción entre agentes.

mo de comunicación altamente desacoplado, en el que el publicador de un evento no es necesariamente consciente de los receptores interesados en el mismo. Este modelo de comunicación encaja perfectamente con algunas de las interacciones típicamente presentes en entornos de IAm, como notificaciones de cambios de estado en sensores o eventos disparados por la actuación del usuario.

En la Figura 6.3 se muestra un diagrama de clases simplificado del diseño del modelo de comunicación publicación/subscripción. La gestión de publicaciones y subscripciones es desacoplado del código de los agentes a través del uso de un elemento *IConversationManager* que encapsula la lógica y datos requeridos para recibir mensajes publicados y para enviar mensajes a los subscriptores. Además, con la ayuda del modelo declarativo de agentes descrito más adelante, los desarrolladores pueden definir de forma declarativa el conjunto de publicaciones que proporciona un agente concreto. Después, en tiempo de ejecución, otros agentes pueden descubrir publicaciones existentes y dinámicamente subscribirse a ellas.

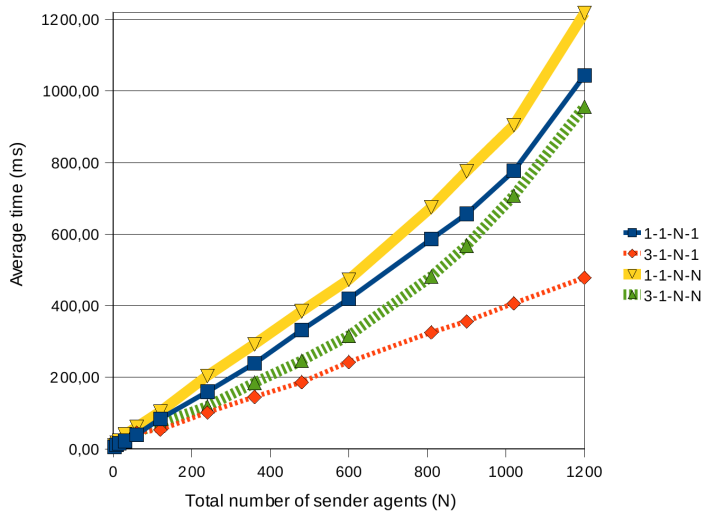



Figura 6.4: Pruebas de rendimiento del sistema extendido de comunicaciones.

Uno de los principales aspectos de rendimiento a tener en cuenta en una plataforma multi-agente es cómo ésta escala en términos de número de mensajes, número de agentes y número de máquinas involucradas en la comunicación. En consecuencia, se realizaron varias pruebas para evaluar el rendimiento de este contenedor utilizando las capacidades de comunicación extendidas que se acaban de describir.

Los test consisten en un grupo de agentes que envían un conjunto de mensajes a otros agentes ejecutándose en diferentes máquinas. Las mediciones de los test se han llevado a cabo en un *cluster*. Los nodos de dicho *cluster* eran biprocesadores, con cuatro núcleos en cada procesador, utilizando Linux de 64 bits como sistema operativo y una red gigabit ethernet. Las mediciones se realizaron en un escenario en el que se enviaban 500 mensajes entre cada par de agentes con el fin de obtener una media del tiempo correspondiente al envío de un mensaje y la recepción de una respuesta al mismo. Los resultados concretos de algunas pruebas se muestran en la Figura 6.4. En particular, se muestra el tiempo medio de comunicación (envío con respuesta) para las siguientes configuraciones:

- **1-1-N-1**. Máquinas: 1 emisor - 1 receptor. Agentes: N emisor - 1 receptor.
- **3-1-N-1**. Máquinas: 3 emisor - 1 receptor. Agentes: N emisor - 1 receptor.
- **1-1-N-N**. Máquinas: 1 emisor - 1 receptor. Agentes: N emisor - N receptor.
- **3-1-N-N**. Máquinas: 3 emisor - 1 receptor. Agentes: N emisor - N receptor.

Se puede observar que el sistema escala bien con el número de mensajes hasta que el número de agentes comienza a ser un factor limitante. El rendimiento em-

pieza a degradarse a partir de 600 agentes ejecutándose de forma simultánea en la misma máquina y tratando de comunicarse con otros agentes remotos. Este efecto se ve claramente en los casos 3-1-N-1 y 3-1-N-N, y es debido principalmente a la penalización introducida por la gestión de un número tan elevado de *threads* por parte de la máquina virtual Java y del sistema operativo.

## 6.4. Un modelo declarativo para la construcción de sistemas multi-agente

Como se mencionó anteriormente, tanto en el ámbito de los sistemas multi-agente como en el ámbito de los sistemas de IAm, se tiende a construir sistemas amplios y complejos en los que habitualmente se podrían compartir y reutilizar muchas funcionalidades entre varias aplicaciones. Una forma de disminuir esta complejidad es promoviendo la división de los sistemas en componentes funcionales altamente desacoplados y reutilizables. Este punto de vista general, llevado al proceso de construcción de un determinado servicio a través de una sistema multi-agente, nos lleva a proponer lo que denominaremos Modelo Declarativo Multi-Agente, que extiende ampliamente el modelo de agentes soportado por la plataforma Jade (Jade ni siquiera proporciona el concepto de sistema multi-agente).

El Modelo Declarativo Multi-Agente proporciona, en primer lugar, un modelo que describe un sistema multi-agente como una sociedad de agentes y sus componentes. En segundo lugar, proporciona un conjunto de herramientas para definir cada uno de esos componentes de forma declarativa. Asimismo, permite construir automáticamente en tiempo de ejecución un sistema multi-agente en base a componentes reutilizables (agentes, comportamientos, mecanismos de comunicación, etc.) definidos en un modelo declarativo.

El modelo está soportado por una serie herramientas que permiten la definición declarativa de los componentes utilizando ficheros de definición en lenguaje XML. El concepto de más alto nivel del modelo son los *Sistemas Multi-Agente (MAS)*. Los MAS son una colección de instancias de diferentes tipos de agentes que trabajan juntos para llevar a cabo una tarea. Por debajo de los MAS se encuentra el concepto de *Tipo de Agente*. Un Tipo de Agente se describe en base a un conjunto de elementos soportados por la plataforma que a su vez también disponen de definiciones declarativas independientes.

Los componentes en base a los cuales se puede construir declarativamente un Tipo de Agente son los siguientes: Comportamientos (correspondientes a Behaviours de Jade), Procesadores de Mensajes, Publicaciones y Funcionalidades. Las funcionalidades de un agente especifican públicamente lo que éste es capaz de hacer (internamente serán resueltas con la ejecución de comportamientos). Cada uno de

Listado 6.1: Ejemplo de definición declarativa de un procesador de mensajes.

```
<hi3–core–message–processing>
  <hi3–core–message–template>

    <feature–list>
      <and>
        <performative> INFORM </performative>
        <or>
          <content–instance–of> package1.Class1.class </content–instance–of>
          <priority> 1 </priority>
        </or>
      </and>
    </feature–list>

    <message–processor–list>
      <class> package1.msgprocessor.MsgProcessor1.class </class>
      <class> package1.msgprocessor.MsgProcessor2.class </class>
    </message–processor–list>

  </hi3–core–message–template>
</hi3–core–message–processing>
```

estos componentes tendrá su propio fichero XML con su definición declarativa. Además, cada descriptor de un componente se acompañará de la implementación en Java del mismo. A modo ilustrativo, en el Listado 6.1 se muestra la estructura del fichero de definición declarativa de un componente Procesador de Mensaje.

Para completar la definición declarativa de un Tipo de Agente, es necesario aclarar que la herencia entre tipos de agentes está soportada. Así, una definición de un Tipo de Agente puede especificar que extiende a otro indicando la URL del Tipo de Agente extendido en su fichero de definición. Al especificar la relación de herencia, las instancias de los agentes del Tipo de Agente hijo heredarán los componentes definidos por el tipo de agente padre.

En el Listado 6.2 se muestra la estructura que debe tener un fichero XML de definición de un Tipo de Agente. A partir de definiciones de Tipos de Agente existentes se pueden definir instancias concretas de los mismos. Dichas instancias se especificarán en ficheros de definición de sistemas multi-agente, de modo que la plataforma automáticamente podrá cargar el descriptor del sistema multi-agente e instanciar y ejecutar cada uno de los agentes que lo forman (ver ejemplo de definición en el Listado 6.3).

Por último, en el Listado 6.4 se muestra la estructura de ficheros y directorios adecuada para empaquetar la definición e implementación de un agente, junto con sus componentes. En el caso de un sistema multi-agente estaría compuesto por su fichero XML de definición junto con la definición e implementación de todos los agentes que lo forman. No obstante, hay que recordar que, para fomentar la reutilización, diferentes agentes pueden construirse en base componentes comunes.

Listado 6.2: Definición declarativa de un Tipo de Agente.

```
<hi3–core–agent–type type="user–agent">
   <name> User Agent 1 </name>
   <extends> hi3–agent–type://package1/agent2/ </extends>

   <argument–list>
      <argument required="true">
         <name> Argument1 </name>
         <data–type> int </data–type>
      </argument>
   </argument–list>

   <behaviour–list>
      <behaviour load–on–start="true">
         hi3–behaviour://package1/agent1/behaviour/behaviour1/
      </behaviour>
   </behaviour–list>

   <functionality–list>
      <functionality>
         hi3–functionality://package1/functionality/functionality1/
      </functionality>
   </functionality–list>
</hi3–core–agent–type>
```

Listado 6.3: Definición declarativa de un Sistema Multi-Agente.

```
<hi3–core–mas>
   <name> MAS 1 </name>

   <agent–instance–list>
      <agent–instance>
         <local–agent–name> Agent1–Instance1 </local–agent–name>
         <type> hi3–agent–type://package1/agent1/ </type>
         <argument–list>
            <argument name="Argument1"> 100 </argument>
            <argument name="Argument2"> Text </argument>
         </argument–list>
      </agent–instance>

      (. . .)
   </agent–instance–list>
</hi3–core–mas>
```

## 6.5. Gestión del contenedor de agentes

A mayores de la construcción en sí del contenedor masVineyard, se han desarrollado una serie de herramientas con el objetivo de facilitar la gestión del contenedor así como del proceso de despliegue y ejecución en el mismo de componentes implementados como sistemas multi-agente.

Listado 6.4: Estructura de ficheros de un agente (descripción e implementación).

```
agent1
|-- Agent1.class
|-- hi3-core-agent-type.xml
|-- hi3-publication-list.xml
|-- hi3-message-processing.xml
|-- behaviour
|   |-- behaviour1
|   |   |-- hi3-core-behaviour.xml
|   |   |-- Behaviour1.class
|   |-- behaviour2
|   |   |-- hi3-core-behaviour.xml
|   |   |-- Behaviour2a.class
|   |   |-- Behaviour2b.class
|-- functionality
|   |-- functionality1
|   |   |-- hi3-core-functionality.xml
```

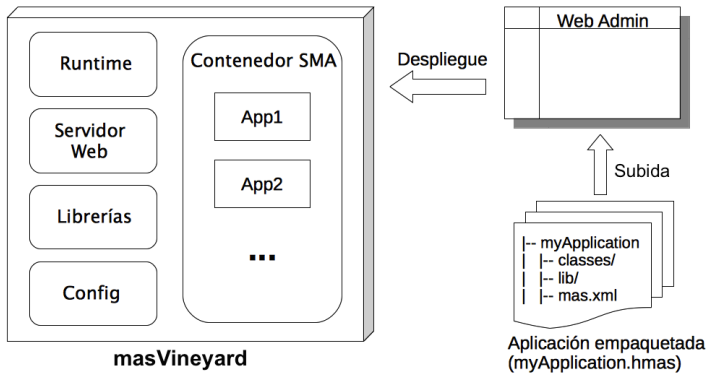



Figura 6.5: Despliegue de un sistema multi-agente en el contenedor masVineyard.

En base al sistema declarativo de definición de sistemas multi-agente, se ha desarrollado un sistema de empaquetamiento de componentes que permite integrar los diferentes elementos de un sistema multi-agente (implementación binaria Java, ficheros XML de definición de componentes, ficheros de configuración, etc.) en un único fichero que puede utilizarse para instalar y desplegar dicho componente directamente en un contenedor masVineyard.

Además, en el contenedor masVineyard se ha integrado un servidor web Jetty [Jetty Project, 2015]. Se trata de un servidor web J2EE ligero que puede ser utilizado por los desarrolladores para implementar interfaces gráficas de usuario asociadas a un agente y basadas en tecnologías Web. Utilizando este servidor Web, junto con una librería desarrollada a tal efecto, se ha implementado e integrado una aplicación web para la administración del contenedor masVineyard, que permite, a través de una cómoda interfaz gráfica, instalar y ejecutar nuevos componentes,

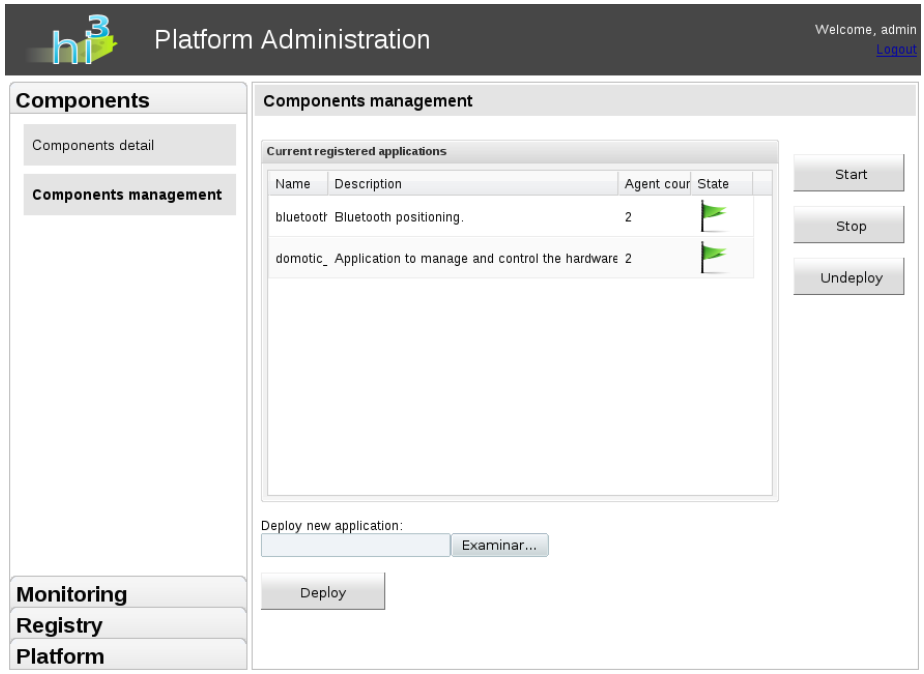



Figura 6.6: Interfaz de usuario para el despliegue de nuevos sistemas multi-agente en el contenedor masVineyard.

gestionar el ciclo de vida de componentes instalados, y monitorizar el estado de los componentes en ejecución (ver Figura 6.5).

En las figuras 6.6, 6.7 y se pueden ver tres capturas de pantalla de la aplicación web desarrollada para la administración del contenedor masVineyard.

## 6.6. Resumen

Aunque la arquitectura HI3 está fundamentada en los principios de las arquitecturas SOA en cuanto a la organización de sus componentes, no restringe el paradigma que los desarrolladores han de utilizar para la implementación interna de nuevas funcionalidades. Así, reconociendo el interés del paradigma de Sistemas Multi-Agente para la construcción de sistemas distribuidos dinámicos y complejos, en este capítulo se propuso una metodología para la integración de este paradigma en la arquitectura HI3. La aproximación propone resolver internamente la funcionalidad con una sociedad de agentes, mientras que un agente representante de dicha sociedad exportará una descripción semántica de la funcionalidad según el modelo BSDM, a la vez que proporciona la interfaz a través de la que se ejecuta ésta.

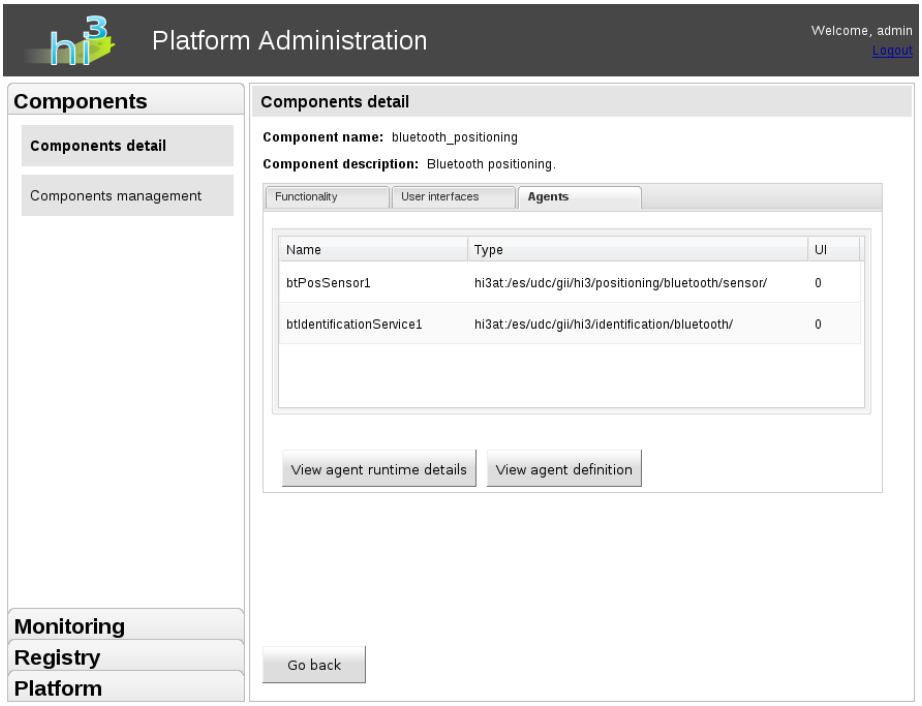



Figura 6.7: Interfaz de usuario para la consulta de información de un sistema multi-agente desplegado en el contenedor masVineyard.

La propuesta teórica para la integración de ambos paradigmas se trasladó a una implementación de un contenedor de la arquitectura HI3, denominado masVineyard, basado en la plataforma de agentes Jade. Dicha implementación ha permitido, asimismo, constatar la escasez de soluciones estándar de alto nivel, para el desarrollo de sistemas multi-agente, existentes en las plataformas de agentes más relevantes. En consecuencia, para la construcción del contenedor masVineyard se desarrollaron una serie de extensiones de la plataforma Jade para dotarla de capacidades de comunicación de más alto nivel o capacidades para reutilización de componentes en base a la definición declarativa de los mismos.

# Capítulo 7

# Construyendo un entorno de Inteligencia Ambiental

## 7.1. Introducción

El objetivo último de la arquitectura HI3 es facilitar a los desarrolladores la tarea de construir y desplegar nuevos sistemas de IAm en entornos reales, proporcionándoles para ello un marco conceptual común, un método, middleware, herramientas y una serie de soluciones predefinidas a problemas comunes de dichos sistemas. Así, vimos como el middleware orientado a servicios, que hemos descrito en los capítulos anteriores, posibilita la construcción de nuevos sistemas en base a servicios reutilizables, interoperables, adaptables y ubicuos, con independencia de las tecnologías concretas seleccionadas para implementar su funcionalidad. Todas estas características favorecen, con carácter general, la construcción de sistemas capaces de adaptarse a su contexto de ejecución, en base a la colaboración espontánea y desacoplada con los componentes existentes en cada escenario sin necesidad de un conocimiento previo de los mismos. Sin embargo, existe un número importante de requisitos adicionales necesarios para que un sistema pueda ser considerado de IAm y que podemos considerar en una capa de abstracción superior a los anteriores. Nos referimos a aspectos relacionados con la integración de los sistemas en los entornos físicos de las personas, la interacción natural con los usuarios, el aprendizaje y adaptación a las costumbres/preferencias de los usuarios, la movilidad de la funcionalidad ligada a la movilidad de los usuarios, la inteligencia social que mejore la aceptación de los sistemas y la "empatía" de los usuarios hacia los mismos, etc. Este capítulo se corresponde con las soluciones de la arquitectura HI3 para este tipo de aspectos de la IAm, que hemos ubicado en un segundo nivel de abstracción dentro de la aproximación propuesta para el proceso de desarrollo de sistemas de IAm.

En concreto, a lo largo de este capítulo se expondrán diferentes aproximaciones a problemas relacionados con nuestra definición de la Capa de Usuario y la Capa de Dispositivos en la arquitectura conceptual HI3, y que fueron identificados como aspectos clave a tratar en la construcción de cualquier entorno de IAm. Siguiendo la metodología de la arquitectura HI3, las soluciones propuestas en este nivel (modelos conceptuales, frameworks, herramientas, etc.) serán abstraídas e integradas, en último término, a través de servicios semánticos ubicuos y ontologías que definirán los conceptos manejados por dichas soluciones en un lenguaje neutral independiente de plataformas concretas.

## 7.2. Abstrayendo el acceso al entorno físico

Como quedó patente en el estudio previo del campo de la IAm realizado en el Capítulo 3, es necesario poblar los entornos físicos donde se desenvuelven las personas de dispositivos que permiten actuar sobre el entorno y monitorizar su estado y el de los usuarios, con el fin de que los sistemas de IAm puedan adaptar su comportamiento al contexto. Vimos, además, como la enorme fragmentación tecnológica existente y la carencia de un middleware de referencia dificulta enormemente esta tarea a los desarrolladores de nuevas aplicaciones. Así, ninguna de las tecnologías existente proporciona una solución todo-en-uno para la instrumentación de entornos de IAm. Con los desarrollos correspondientes a este apartado se trata de avanzar en este ámbito e integrar los mismos para proporcionar una aproximación a la resolución de la Capa de Dispositivos de la arquitectura HI3.

En la Figura 7.1 se presentan los principales componentes de la Capa de Dispositivos y las relaciones entre ellos. En un primer nivel de abstracción podemos considerar la existencia de diferentes tecnologías en el ámbito del IoT que, a través de middleware heterogéneo, proporcionan diferentes tipos de soluciones para abstraer, en mayor o menor grado, el acceso a dispositivos de sensorización/actuación. En concreto, se considera la Librería UniDA (del inglés, Uniform Device Access) [Varela et al., 2011], que será utilizada en la arquitectura HI3 como un framework de IoT para proporcionar un primer nivel de abstracción en el acceso a dispositivos. Esta librería proporciona una implementación en lenguaje Java de un modelo uniforme de abstracción de dispositivos y un protocolo de operación distribuida entre dispositivos.

En niveles superiores de abstracción a la Librería UniDA, en este trabajo se proponen dos nuevos elementos principales: un framework para la creación de dispositivos virtuales en base a la agregación y composición de otros dispositivos (framework DAAF) y un servicio para la integración de middleware heterogéneo de IoT bajo un modelo común de operación (Servicio UniDA).

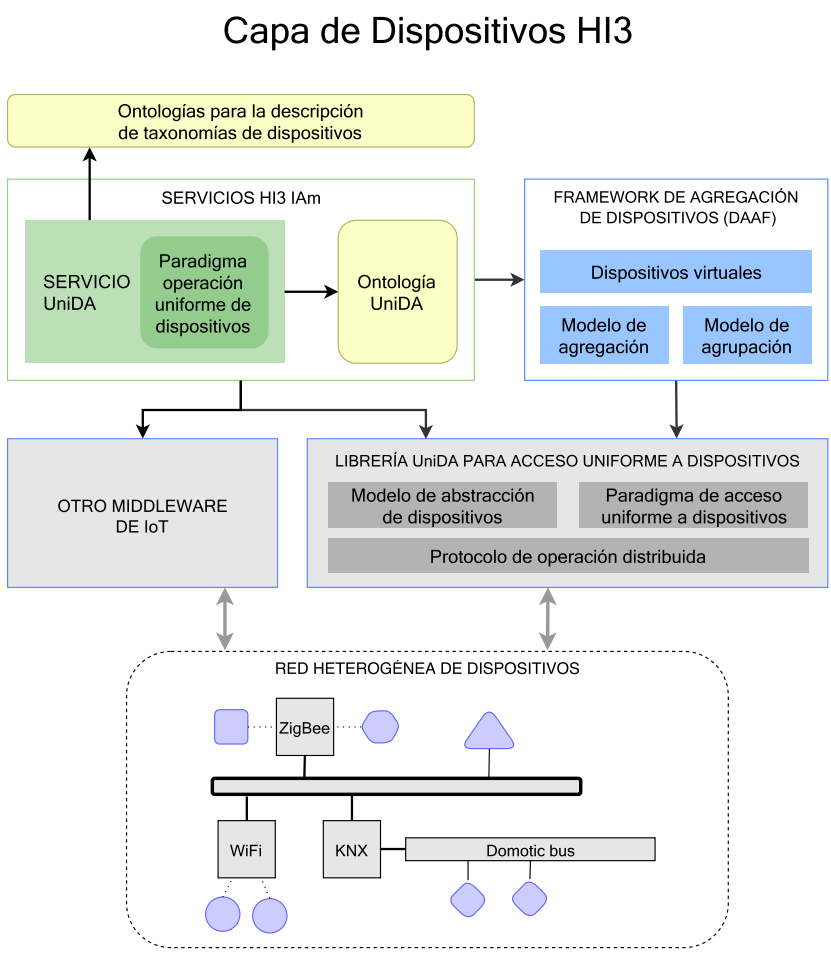



Figura 7.1: Visión general de la Capa de Dispositivos de la arquitectura HI3.

El framework DAAF proporciona capacidades para crear abstracciones de más alto nivel en el acceso a dispositivos a través de la definición de dispositivos virtuales con nuevas capacidades, definidos a partir de la agregación y la composición de otros dispositivos. Para el acceso final a los dispositivos físicos reales el framework DAAF ofrece la capacidad de integrarse con la Librería UniDA, aunque no es imprescindible para su uso. Por otro lado, el Servicio UniDA permite exportar la funcionalidad relacionada con el acceso uniforme a redes heterogéneas de dispositivos a través de un componente semánticamente descrito, ubicuo e interoperable.

A lo largo de la sección se hará una descripción pormenorizada de los diferentes trabajos realizados en esta tesis que se corresponden con la Capa de Dispositivos de la arquitectura HI3, ilustrando como, de forma conjunta, proporcionan una solución integral para el problema del acceso al entorno físico en sistemas de IAm.

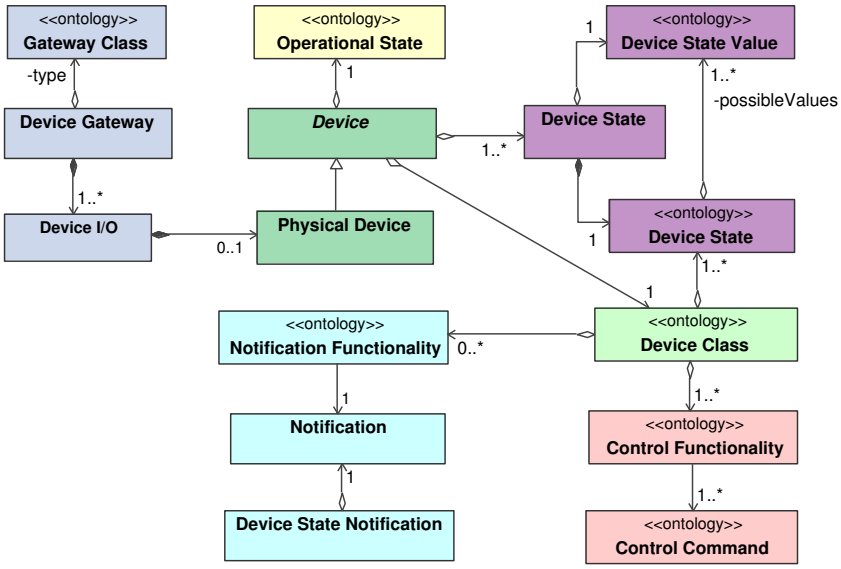



Figura 7.2: Modelo de dispositivos utilizado por la Librería UniDA.

### 7.2.1. Librería UniDA

La Librería UniDA actúa como un framework de IoT, proporcionando una visión homogénea de diferentes tecnologías de dispositivos a través de dos elementos: un modelo uniforme para la representación de los dispositivos y un paradigma uniforme para la interacción con los mismos.

Propone un modelo que abstrae una red de instrumentación fundamentalmente en base a las capacidades de sensorización/actuación de sus dispositivos. A mayores, utiliza una taxonomía de dispositivos para representar los diferentes tipos de dispositivos existentes y sus características. En concreto, utiliza la taxonomía de dispositivos proporcionada por la ontología DogOnt [Bonino and Corno, 2008].

En cuanto al modelo de representación de dispositivos, consta de los siguientes conceptos principales (ver Figura 7.2):

- **Dispositivo.** Representación de un dispositivo de la red de instrumentación. Contiene tanto la información descriptiva del dispositivo como la información de sus estado.
- **Dispositivo físico.** Representa un dispositivo hardware real. La diferencia con el concepto Dispositivo es que este último representa una visión más abstracta, que podría corresponderse, por ejemplo, con un dispositivo lógico.
- **Gateway de dispositivos.** Actúan como el punto de acceso de los dispositi-

vos a la red global de instrumentación, realizando una traducción entre el lenguaje de la red uniforme de dispositivos y el lenguaje particular de una tecnología concreta.

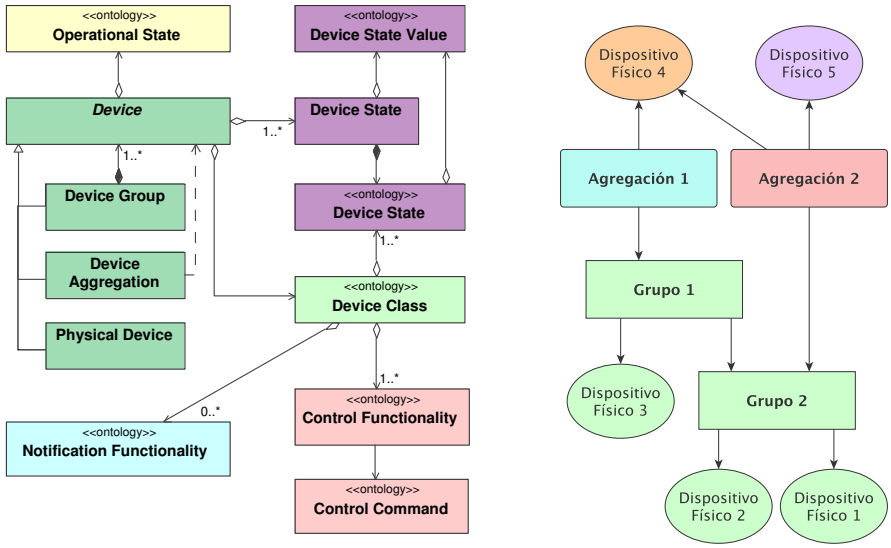
- **Dispositivo I/O.** Representa una conexión lógica de un gateway en la que se puede conectar un Dispositivo físico de un determinado tipo.
- **Clase de dispositivo.** Representa un tipo de un dispositivo definido en la ontología DogOnt.
- **Estado de dispositivo.** Representa un tipo de estado que tiene un dispositivo. El valor de un estado puede ser leído y opcionalmente escrito.
- **Comando de dispositivo.** En conjunto representan la funcionalidad de un dispositivo. Un dispositivo puede declarar y recibir comandos, que habitualmente tienen algún efecto sobre su estado interno y/o desencadenan una actuación sobre el entorno.
- **Notificación de dispositivo.** Representan cambios en el estado de un dispositivo que son enviadas en forma de evento por los dispositivos.

La implementación de la Librería UniDA proporciona una librería Java que exporta la funcionalidad de una red de instrumentación a través de una serie de fachadas. Estas fachadas permiten el acceso al modelo uniforme que representa la red de instrumentación y proporcionan un conjunto sencillo de operaciones abstractas para interaccionar con los dispositivos. Además, proporciona una realización del concepto Gateway que puede ser extendido para construir componentes software capaces de abstraer el acceso a diferentes tecnologías. Éstos trasladan los conceptos comunes manejados por el modelo uniforme de dispositivos y lo traducen a los conceptos manejados por cada tecnología de dispositivos concreta. También proporciona un protocolo basado en mensajes UDP para permitir la operación distribuida de la red de instrumentación, facilitando la interacción entre los distintos elementos de la red.

Se puede obtener un mayor detalle acerca del diseño de esta solución en [Varela et al., 2011]. Asimismo, la implementación de la misma ha sido publicada como código abierto bajo la licencia GNU Affero GPL v3, siendo accesible a través de un repositorio público GitHub disponible en [LibreriaUniDA, 2015].

### 7.2.2. Abstracción y agregación uniforme de dispositivos

En esta sección se presenta un framework que, a mayores de desacoplar las aplicaciones de las particularidades del acceso a las tecnologías de dispositivos, proporciona capacidades para la creación dinámica de abstracciones de más alto nivel para las dispositivos, definiendo dispositivos virtuales capaces de agregar otros dispositivos para alcanzar nuevas funcionalidades. El framework se denomina con




(a) Modelo conceptual.

(b) Ejemplo de grupos y agregaciones (mismo color para dispositivos con la misma funcionalidad).

Figura 7.3: Modelo de abstracción de dispositivos.

las siglas DAAF (del inglés Device Abstraction and Aggregation Framwork) y se construye haciendo uso de las capacidades de la Librería UniDA para obtener un acceso uniforme a una red de instrumentación. Así, en DAAF se define una extensión de los conceptos del modelo de dispositivos manejado por la Librería UniDA, para proporcionar una serie de abstracciones de más alto nivel.

Siguiendo la misma filosofía de la Librería UniDA, proporciona una separación entre el modelo abstracto que representa los conceptos de alto nivel que soportan la operación distribuida de dispositivos y una taxonomía que representa los tipos de dispositivos existentes y sus propiedades. En este caso también se utiliza la ontología DogOnt como taxonomía de dispositivos de referencia. En la Figura 7.3 se ilustra, de forma simplificada, la extensión del modelo abstracto de dispositivos propuesto por la Librería UniDA con la inclusión de dos nuevos conceptos:

- **Grupos de dispositivos.** Proporcionan una forma de crear conjuntos de dispositivos que comparten el mismo tipo de funcionalidad. El grupo resultante tiene la misma entidad conceptual que un dispositivo individual.

- **Agregaciones de dispositivos.** Permiten la creación de nuevos tipos de dispositivos virtuales con nuevas funcionalidades, que son implementadas por agregación de dispositivos que pueden ser heterogéneos en cuanto a sus estados y funcionalidades.

Para la realización concreta del modelo conceptual de dispositivos del framework DAAF se propone como elemento central un componente software denominado Dispositivo Virtual (DV). Se trata de una representación software del concepto de dispositivo que actúa como una abstracción de un dispositivo real, de un grupo de dispositivos o de una agregación de dispositivos, proporcionando capacidades en un nivel de abstracción superior (ver Figura 7.4). Un DV se compone principalmente de: un *tipo*, *estados* para representar valores correspondientes a sensorizaciones del entorno, y *comandos* para representar acciones que puede realizar sobre el entorno utilizando sus dispositivos hardware asociados. Además, tiene un *comportamiento* que define la lógica que dirige la operación del DV. Se proponen cuatro especializaciones de un DV:

- **Abstracción DV.** Esta clase de DV representa un dispositivo hardware real disponible en el sistema. Incluye comportamientos para interaccionar directamente con el dispositivo hardware y trasladar los conceptos manejados por la tecnología concreta del dispositivo al modelo abstracto de dispositivos del framework DAAF. Por tanto, este DV se utiliza para crear un envoltorio semánticamente anotado sobre un dispositivo hardware concreto y proporcionar acceso uniforme al mismo a través del framework DAAF.
- **UniDA DV.** El framework DAAF puede utilizarse también en combinación con una tecnología de acceso al hardware de más bajo nivel, como la Librería UniDA, que proporcione el primer nivel de abstracción de una red de instrumentación. Este tipo de DV puede incluir un comportamiento que realice algún tipo de adaptación adicional sobre los estados y operaciones del dispositivo abstracto original.
- **Grupo DV.** Componente software que conecta múltiples DV pertenecientes a un mismo tipo (device class) de la ontología de dispositivos, proporcionando un único punto de control para todos ellos. Por tanto, este DV continúa siendo del mismo tipo que los dispositivos agrupados que contiene, y exporta los mismos estados y comandos que éstos.
- **Agregación DV.** Componente que agrega otros DV que pueden ser heterogéneos. A diferencia de un Grupo DV, permite la creación de un DV de más alto nivel correspondiente a un tipo de dispositivo (device class) diferente al de sus dispositivos agregados. Tiene un comportamiento que gestiona los estados y comandos proporcionados por este nuevo tipo de dispositivo y los resuelve en base a interacciones con sus dispositivos agregados.

La estructura externa de cada tipo de DV es la misma, las únicas diferencias entre dos DV serán su Tipo y su Comportamiento. El tipo de un DV determina sus metadatos semánticos. Especifica el *device class* correspondiente en la ontología DogOnt, describiendo por tanto los comandos y estados que soporta. En el caso de un Agregación DV su tipo también proporciona una forma de restringir que clase de

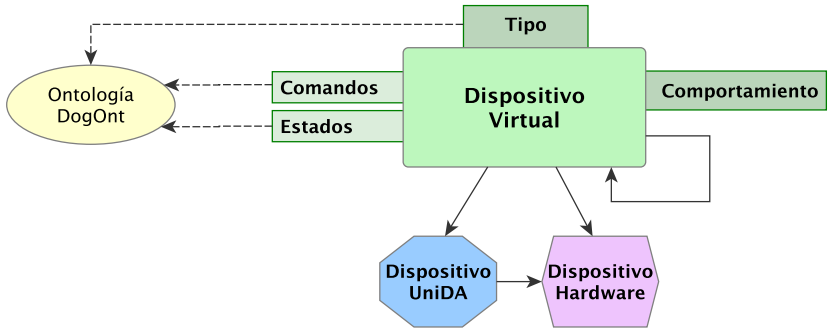



Figura 7.4: Dispositivo virtual.

dispositivos pueden ser agregados, en función de los estados y los comandos que soportan. El comportamiento de un DV determina cómo operará en tiempo de ejecución. Define, por ejemplo, estrategias para la agregación de datos de sensorización de sus dispositivos agregados, qué hacer cuando se recibe un comando, etc. El comportamiento puede ser programado específicamente por un desarrollador o puede ser seleccionado de un repositorio de comportamientos proporcionados por el sistema. Por defecto, este repositorio incluye los siguientes comportamientos: i) un comportamiento de grupo que redirige recursivamente todos los comandos recibidos en un Grupo DV a todos sus DV asociados; ii) un comportamiento de agregación que utiliza los metadatos semánticos disponibles en el tipo de un Agregación DV para asociar cambios en estados de unos DV con la ejecución de comandos en otros DV. Por ejemplo, si un tipo de Agregación DV define un estado de detección de presencia y un comando de encendido/apagado, y posteriormente se asocia este Agregación DV con un DV existente en el entorno que representa un detector de presencia y un DV que representa un pulsador de la luz, un comportamiento de este tipo puede ejecutarse automáticamente de modo que cuando recibe una notificación de presencia desencadene un comando en el dispositivo pulsador para encender la luz.

Un desarrollador pueden implementar nuevos tipos de DV desacoplados de dispositivos concretos. Para ello únicamente es necesario que defina un nuevo Tipo y que programe un conjunto de posibles comportamientos para controlar la operación del DV. Estos comportamientos presentan una estructura común, de modo que pueden ser asociados dinámicamente a los DV. Así, para el desarrollo de un comportamiento, es necesario implementar dos estrategias que incluyen métodos *callback* que son llamados automáticamente cada vez que el DV recibe un comando o una petición de lectura de un estado desde el exterior.

Desde el punto de vista de una aplicación, ésta puede utilizar un componente Repositorio DV (ver Figura 7.5) para descubrir los DVs disponibles en una instalación, buscándolos por su funcionalidad (descrita según la ontología DogOnt). Pos-

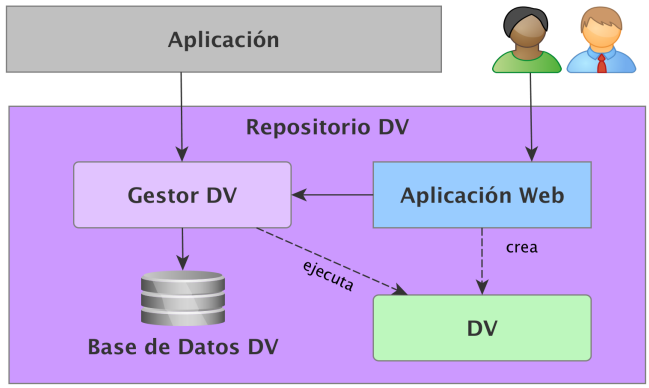



Figura 7.5: Repositorio de dispositivos virtuales.

teriormente puede interaccionar con estos DV utilizando su interfaz uniforme. La información referente a los comandos y estados soportados por un DV puede ser obtenida en la ontología DogOnt a partir del tipo especificado en el DV.

El framework DAAF se completa con un conjunto adicional de herramientas, proporcionadas por el Repositorio DV, que permiten la gestión y configuración de los DV disponibles en una instalación de un sistema concreto:

- Una base de datos para almacenar la información relativa a los DV definidos en el sistema.
- Un componente Gestor DV, encargado de gestionar el ciclo de vida de los DV, siendo capaz de configurar y ejecutar los DV definidos a tal efecto en la BD. Además, proporciona un API para que las aplicaciones puedan realizar consultas sobre los DV disponibles, registrar nuevos DV, etc.
- Una aplicación web que proporciona una interfaz de usuario que permite definir nuevos DV o consultar y configurar los existentes.

El Gestor DV tiene acceso a los diferentes tipos de DV soportados por el sistema y a los comportamientos asociados con cada tipo de DV. Utilizando esta información, la interfaz de usuario permite a los usuarios definir nuevos DV concretos de cualquiera de los tipos soportados por el sistema. Por ejemplo, para crear un Grupo DV o un Agregación DV el sistema solicitará que se asocien con otros DV concretos existentes. Además, dado que el framework DAAF puede integrarse con la Librería UniDA, es posible utilizar la interfaz de usuario para explorar la red de dispositivos a la que proporciona acceso la Librería UniDA y crear DV que agreguen dichos dispositivos. En la Figura 7.6 se muestra una captura de pantalla de la aplicación web desarrollada.

En conjunto el framework DAAF permite el desarrollo de aplicaciones reutilizables y adaptables al entorno gracias a un modelo común para una red de dispositivos

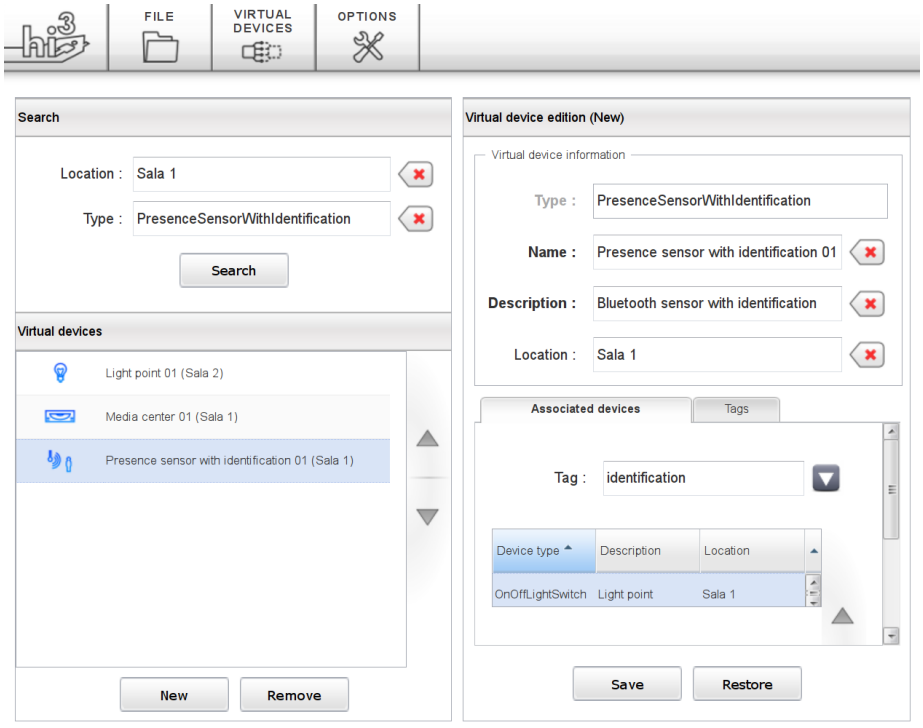



Figura 7.6: Aplicación web para la gestión de una instalación.

que desacopla las aplicaciones de las tecnologías concretas de dispositivos. Además, posibilita la definición dinámica de nuevos dispositivos virtuales de alto nivel, facilitando la agregación de dispositivos existentes para añadir nuevas funcionalidades al sistema sin necesidad de reprogramar el mismo.

### 7.2.3. Un servicio semántico para el acceso uniforme al entorno

Como vimos en el Capítulo 3, además de la Librería UniDa y el framework DAAF, existen numerosas aproximaciones que desde diferentes puntos de vista intentan lidiar con la enorme heterogeneidad de tecnologías de sensorización/actuación disponibles. No obstante, también vimos que, como ocurren en otras áreas relacionadas con la IAm, estamos lejos de disponer de una solución global de referencia en este ámbito. Por tanto, una opción para la arquitectura HI3 sería tratar de imponer una solución única de entre las existentes. Esto iría en contra de la visión integradora de este trabajo. Así, la aproximación propuesta para la Capa de Dispositivos de la arquitectura HI3 contempla la necesidad de integrar distintos frameworks de IoT existentes dependiendo de las tecnologías presentes en un entorno concreto.

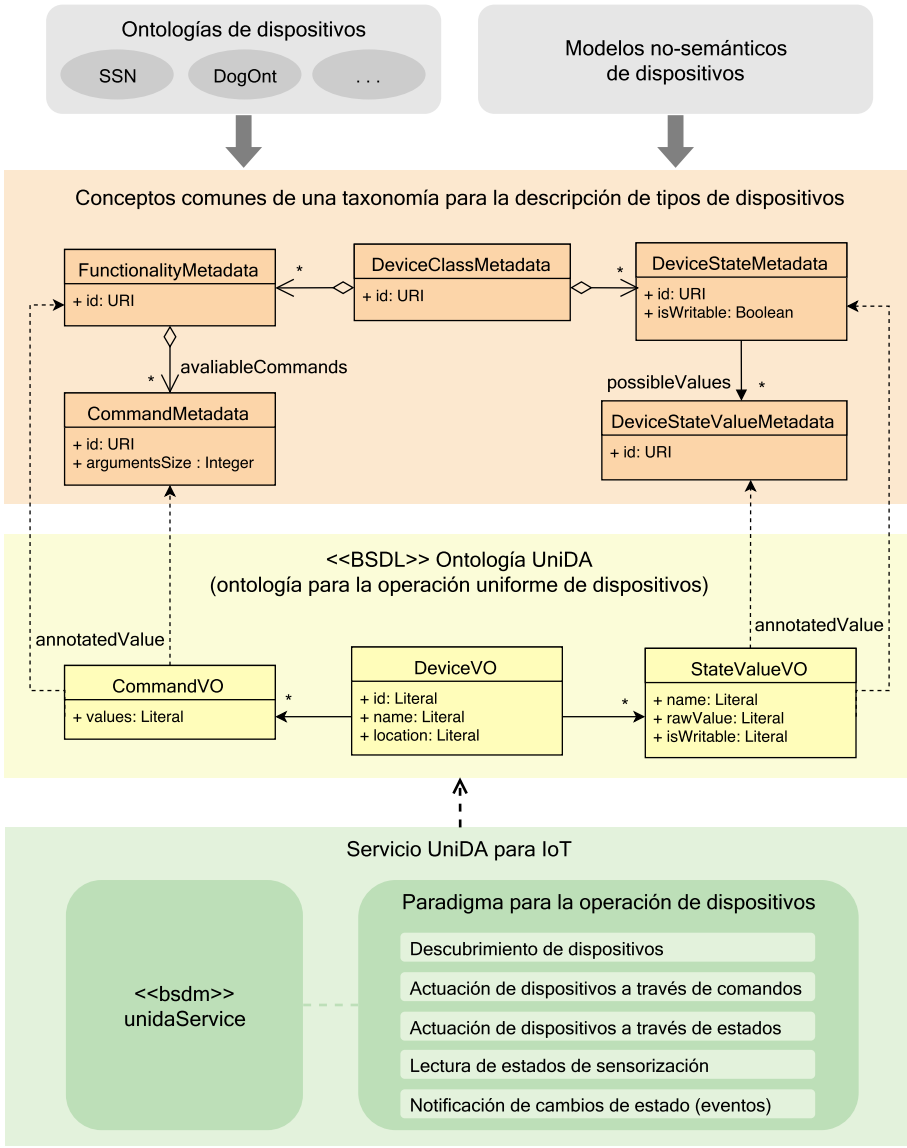



Figura 7.7: Abstracción del acceso al entorno físico a través del Servicio UniDA.

En consecuencia, en esta sección se describe una solución para el acceso uniforme al entorno físico basada en la utilización de servicios semánticos interoperables capaces de proporcionar acceso ubicuo a redes de dispositivos heterogéneos a través de un paradigma de operación de dispositivos uniforme y realizable a través de diferentes frameworks más concretos. Para ello se propone el Servicio UniDA, que está fundamentado en los siguientes conceptos:

- Se utilizan ontologías existentes únicamente para describir taxonomías de los tipos de dispositivos que pueden estar presentes en el entorno. Existen diferentes ontologías (orientadas a entornos domésticos, a redes de sensorización inalámbrica, etc.), pudiendo unas adaptarse mejor a la descripción de un determinado entorno que otras.
- Se propone una nueva ontología (ontología UniDA) que, en base a un grupo reducido de conceptos, permite capturar la información mínima necesaria para potencialmente operar cualquier dispositivo sensor/actuador existente. Es importante destacar que esta ontología se centra únicamente en el aspecto operativo de un dispositivo, asumiendo que en esencia un dispositivo sensor/actuador puede tener estados que pueden consultarse y en algunos casos actuarse, y puede tener funcionalidades más complejas no directamente vinculadas al valor de un estado.
- Se propone un paradigma común para la operación de dispositivos sensores y actuadores en base a una serie de operaciones que pueden realizarse sobre el modelo operativo de dispositivos definido en la ontología UniDA.
- El Servicio UniDA puede delegar en diferentes frameworks de IoT que proporcionan distintos modelos de abstracción de redes de dispositivos y resuelven el acceso a los mismos en un entorno concreto. Por tanto, consigue dotar a distintos frameworks de IoT de todas las capacidades de un servicio HI3.

En la Figura 7.7 se ilustran los conceptos que se acaban de introducir. En primer lugar, se tiene en cuenta la existencia de un grupo de aproximaciones de gran relevancia para la construcción de modelos, que permitan describir las características de dispositivos heterogéneos y la construcción de taxonomías de tipos concretos de dispositivos descritos en base a ese conjunto de características comunes. Además, algunas de estas aproximaciones proponen modelos semánticos definidos en lenguajes de ontologías estándar. Algunos de estos modelos cubren información referente a numerosos aspectos de los dispositivos, como la precisión de las observaciones de sensores en determinadas condiciones, aspectos relacionados con la energía, etc. No obstante, nuestro objetivo es quedarnos con un grupo reducido de conceptos relacionados con la operación de los dispositivos que puedan ser fácilmente asimilables a la mayoría de modelos de dispositivos existentes. Se asume por tanto, que otras tareas, relacionadas por ejemplo con la configuración de una red de instrumentación o el mantenimiento de dispositivos, se realicen a través de otras herramientas dependientes de un framework o tecnología concreta. Así, como vemos en la Figura 7.7, se han identificado cinco conceptos fundamentales en base a los que describir las capacidades de un sensor/actuador:

- **Clases de dispositivo.** Identifica un tipo de dispositivo que puede existir en el entorno (un pulsador, un sensor de presencia, un reproductor de audio, etc.). Un tipo de dispositivo definirá los tipos de estados que tiene (tanto

para sensorización como para actuación) y una serie de funcionalidades que ofrece.

- **Estados.** Especifican los tipos de estado que están definidos. Puede existir una jerarquía de tipos de estados, desde los más genéricos (discreto, continuo, binario, etc.) hasta los más concretos (encendido-apagado, intensidad de luz, etc.) Se consideran estados que únicamente pueden ser consultados (sensores) y estados que pueden ser consultados y modificados (actuadores).
- **Valores de un estado.** Para un tipo de estado concreto es posible indicar todos los valores posibles que puede tomar el mismo. Por ejemplo, un estado discreto de tipo *encendido-apagado* puede especificar que admite dos valores: *encendido* y *apagado*.
- **Funcionalidades.** Más allá de las funcionalidades directas relacionadas con leer un estado de sensorización y leer/escribir un estado de actuación de un dispositivo, es útil considerar que pueden existir funcionalidades más complejas, que involucran un comportamiento más elaborado en un dispositivo. Así, una clase de dispositivo podría especificar que dispone de funcionalidades de este tipo y que para ejecutarlas proporciona una serie de comandos.
- **Comandos de funcionalidades.** Especifican qué comandos tiene una funcionalidad y qué tipo de parámetros reciben para poder ser invocados.

Además, para dotar de mayor potencia descriptiva al modelo anterior, se admite la herencia (o especialización) en todos sus conceptos. Por tanto, asumimos que un modelo tan sencillo como este puede ser fácilmente trasladado a los conceptos manejados por cualquiera de las ontologías más relevantes existentes que proporcionan definiciones de taxonomías de dispositivos. Partiendo de este principio, se define una ontología, la ontología UniDA, para la representación homogénea y universal de las instancias de dispositivos existentes en un entorno concreto, y sobre las que nos permitirá operar el Servicio UniDA. Por tanto, esta ontología asume que a mayores se está utilizando alguna otra ontología existente que proporciona la taxonomía de posibles tipos de dispositivos. Por tanto los conceptos de la ontología UniDA que permiten representar dispositivos incluyen atributos que actúan como referencias (anotaciones semánticas) a otra ontología que se esté utilizando como taxonomía de dispositivos. De esta forma se consigue desacoplar la semántica del Servicio UniDA de una taxonomía concreta de dispositivos.

Finalmente, en la Figura 7.7 también vemos como el Servicio UniDA trabaja con conceptos de la ontología UniDA y proporciona un paradigma uniforme para la operación de los dispositivos así descritos. Las operaciones soportados por dicho paradigma son las siguientes:

- **GetConnectedDevices.** Descubrir los dispositivos conectados disponibles en el entorno en un momento dado.

- **GetConnectedDevicesByClass.** Descubrir los dispositivos conectados disponibles en el entorno en un momento dado, que pertenecen a una clase de dispositivo determinada.

- **ExecuteCommand.** Solicitar la ejecución de un comando en el dispositivo indicado. El comando concreto solicitado se identifica con un identificador de la funcionalidad y un identificador del comando que son referencias a conceptos de la ontología que describe la taxonomía de dispositivos.

- **CheckCurrentStateValue.** Solicitar la lectura del valor actual para un estado de un dispositivo.

- **CheckCurrentStatesValues.** Solicitar la información de los valores actuales para todos los estados de un dispotivo.

- **WriteState.** Solicitar la escritura de un estado de un dispositivo con un valor determinado.

- **SubscribeToChangesInState.** Solicitar la subscripción a eventos de notificación de cambios de valor en un estado de un dispositivo.

- **UnsubscribeToChangesInState.** Cancelar la suscripción a eventos de un estado de un dispositivo.

El Servicio UniDA realizará la traducción de este paradigma a las APIs de frameworks concretos de IoT. En el caso de la implementación de referencia de la arquitectura HI3 realizada, se ha implementado una realización concreta del Servicio UniDA que utiliza la Librería UniDA y el framework DAAF.

En el Listado 7.1 se incluye un fragmento de la especificación del modelo BSDM correspondiente al Servicio UniDA, en concreto un fragmento del perfil del servicio. En él se describen dos de sus funcionalidades (correspondientes a las operaciones SubscribeToChangesInState y CheckCurrentStatesValues del paradigma uniforme de operación de dispositivos). Con estos casos se ilustran dos tipos distintos de funcionalidades: AdvertisedSubscription (subscripción a eventos asíncronos, correspondiente al paradigma de interacción publicación/subscripción) y AdvertisedFunctionality (invocación asíncrona con una única respuesta, correspondiente al paradigma de interacción petición/respuesta). Por otro lado, en el Listado 7.2 se incluye un fragmento de la definición del grounding del servicio, correspondiente al grounding de la funcionalidad subscribeToChangesInState, basado en canales de mensajería asíncrona y configurado con el protocolo STOMP. Dicho grounding puede igualmente ser definido, y utilizado de forma transparente, sobre cualquiera de los otros protocolos soportados por defecto en la implementación de referencia de la arquitectura HI3 (WebSockets y JMS).

Listado 7.1: Fragmento de la descripción del perfil del Servicio UniDA.

```
<import prefix="bsdl" to="http://hi3project.com/broccoli/bsdl"/>
<import prefix="bsdm" to="http://hi3project.com/broccoli/bsdm"/>
<import prefix="bsdmProfile" to="http://hi3project.com/broccoli/bsdm/profile"/>
<import prefix="unidaOntology" to="http://hi3project.com/unida/ontology"/>
<import prefix="unidaService" to="http://hi3project.com/unida/service"/>
<import prefix="unidaProfile" to="http://hi3project.com/unida/service/profile"/>

<instance of="bsdm#serviceDescription" URI="unidaService#serviceDescription">
   <with property="profile">
   <instance of="bsdm#serviceProfile" URI="unidaService#serviceProfile">

      <with property="name" value="UniDA service profile"/>

      <with property="advertisedSubscription">
         <instance of="bsdmProfile#advertisedSubscription"
                   URI="unidaProfile#subscribeToChangesInState">

         <with property="name" value="subscribeToChangesInState"/>
            <with property="input">
               <with property="name" value="device"/>
               <with property="type" valueURI="unidaOntology#DeviceVO"/>
            </with>
            <with property="input">
               <with property="name" value="state"/>
               <with property="type" valueURI="unidaOntology#StateValueVO"/>
            </with>
            <with property="output">
               <with property="name" value="stateValue"/>
               <with property="type" valueURI="unidaOntology#StateValueVO"/>
            </with>
         </instance>
      </with>

      <with property="advertisedFunctionality">
         <instance of="bsdmProfile#advertisedFunctionality"
                   URI="unidaProfile#checkCurrentStatesValues">

         <with property="name" value="checkCurrentStatesValues"/>
            <with property="input">
               <with property="name" value="device"/>
               <with property="type" valueURI="unidaOntology#DeviceVO"/>
            </with>
            <with property="output">
               <with property="name" value="stateValues"/>
               <with property="type" valueURI="unidaOntology#StateValueVO"/>
            </with>
         </instance>
      </with>

      (. . .)
```

### Implementación del Servicio UniDA

A lo largo de este apartado se describirá la implementación concreta que se ha hecho del Servicio UniDA como parte de la implementación de referencia de la

Listado 7.2: Fragmento de la descripción del grounding del Servicio UniDA.

```
<import prefix="bsdm" to="http://hi3project.com/broccoli/bsdm"/>
<import prefix="bsdmGrounding"
     to="http://hi3project.com/broccoli/bsdm/grounding"/>
<import prefix="unidaService" to="http://hi3project.com/unida/service"/>
<import prefix="unidaProfile" to="http://hi3project.com/unida/service/profile"/>
<import prefix="unidaGrounding" to="http://hi3project.com/unida/service/grounding"/>

<with property="grounding">
   <instance of="bsdm#asyncMessageServiceGrounding"
             URI="unidaService#serviceGrounding">

      <with property="url" value="stomp://localhost:61701"/>

      <with property="functionalityGrounding">
         <instance of="bsdmGrounding#asyncMessageFunctionalityGrounding"
                   URI="unidaGrounding#subscribeToChangesInState">

            <with property="advertisedFunctionality"
                     of URI="unidaProfile#subscribeToChangesInState"/>
         </instance>
      </with>

      (. . .)

   </instance>
</with>
```

arquitectura HI3. Esta descripción servirá, además, para ilustrar paso a paso como se utilizan algunas de las capacidades del middleware orientado a servicios de la arquitectura HI3 descrito en el Capítulo 5 para la implementación de un servicio semántico complejo.

La implementación concreta de las funcionalidades del servicio, correspondientes a la especificación del modelo BSDM del mismo introducido anteriormente, se ha realizado en lenguaje Java. Además, se ha desarrollado una implementación que utiliza la Librería UniDA como framework concreto de IoT para el acceso a una red heterogénea de dispositivos. Como veremos, el Servicio UniDA, gracias a las capacidades del middleware SOA desarrollado, soporta de forma trivial diferentes implementaciones que podrían utilizar diferentes frameworks de IoT, o varios frameworks de forma simultánea, manteniendo una misma descripción semántica de sus capacidades. Esto permite a los desarrolladores de aplicaciones/servicios que necesitan acceder al entorno físico implementar sus componentes con independencia del entorno en el que se van a ejecutar.

En el Listado 7.3 se incluye un fragmento del descriptor en lenguaje BSDL del Servicio UniDA correspondiente a la descripción de la implementación. Como ya hemos visto, el modelo BSDM soporta una descripción abstracta de la implementación, permitiendo su extensión para soportar descripciones concretas asociadas a

Listado 7.3: Fragmento de la descripción de implementación del Servicio UniDA.

```
<import prefix="bsdm" to="http://hi3project.com/broccoli/bsdm"/>
<import prefix="bsdmImpl" to="http://hi3project.com/broccoli/bsdm/implementation"/>
<import prefix="unidaService" to="http://hi3project.com/unida/service"/>
<import prefix="unidaProfile" to="http://hi3project.com/unida/service/profile"/>
<import prefix="unidaImpl" to="http://hi3project.com/unida/service/implementation"/>

<with property="implementation">
   <instance of="bsdm#serviceImplementation"
          URI="unidaService#serviceImplementation">

      <with property="implementationType" value="javaJenaBeansBSDL"/>

      <with property="functionalityImplementation">
         <instance of="bsdmImpl#functionalityImplementationJena"
                URI="unidaImpl#subscribeToChangesInState">

            <with property="class"
                   value="com.hi3project.soc.test.unida.SemanticUniDA"/>
            <with property="method" value="subscribeToChangesInState"/>
            <with property="advertisedFunctionality"
                   ofURI="unidaProfile#subscribeToChangesInState"/>
         </instance>
      </with>

      <with property="functionalityImplementation">
         <instance of="bsdmImpl#functionalityImplementationJena"
                URI="unidaImpl#checkCurrentStatesValues">

            <with property="class"
                   value="com.hi3project.soc.test.unida.SemanticUniDA"/>
            <with property="method" value="checkCurrentStatesValues"/>
            <with property="advertisedFunctionality"
                   ofURI="unidaProfile#checkCurrentStatesValues"/>
         </instance>
      </with>

      (. . .)
```

diferentes lenguajes y plataformas en las que podría implementarse la funcionalidad de un servicio. En concreto, vimos como la arquitectura HI3 incluye por defecto un modelo de implementación para el lenguaje Java, que es el utilizado en este caso. Como hemos dicho, su uso no es obligatorio, pero veremos como puede facilitar notablemente el trabajo del programador del servicio. Así, volviendo al Listado 7.3, vemos como para las dos funcionalidades del Servicio UniDA ya utilizadas previamente como ejemplo (subscribeToChangesInState y checkCurrentStatesValues) se describe una implementación Java, en la que se indica la clase Java en la que está implementada cada funcionalidad y el método concreto a través del que se resuelve.

Antes de pasar a la implementación de los métodos Java que realizan las funcionalidades del servicio, ilustraremos como se implementa un modelo de objetos Java que permite manejar los conceptos del dominio utilizados por el servicio y que

Listado 7.4: Anotación del modelo de objetos nativo.

```
@Namespace("http://hi3project.com/unida/ontology#")
public class StateValueVO
{
   private String deviceId;
   private String stateId;
   private String stateValueId;
   private String stateValueRaw;
   private String isWritable;

   public StateValueVO() {}

   /** Construcción del concepto StateValueVO de la ontología UniDA a partir de objetos
    *  de la Librería UniDA (usada por el servicio como framework de IoT).
    */
   public StateValueVO(IDevice id, DeviceState deviceState)
   {
      this.deviceId = id.getId().toString();
      this.stateId = deviceState.getMetadata().getShortId();
      this.stateValueId = deviceState.getValue().getValueIdShort();
      this.stateValueRaw = deviceState.getValue().getValueRaw();
      this.isWritable = Boolean.toString(deviceState.getMetadata().isWritable());
   }

   @rawValue
   public String getStateValueRaw()
   {
      return stateValueRaw;
   }

   (. . .)
}
```

fueron definidos en la ontología UniDA (dispositivos, estados y comandos). Así, en el Listado 7.4 vemos como se implementa una clase StateValueVO que sirve para representar el concepto del mismo nombre de la ontología UniDA. El ejemplo sirve para ilustrar la capacidad de utilizar anotaciones en las clases Java para especificar que se corresponden con un concepto de una ontología. Utilizando dicha característica, las instancias de conceptos manejadas por un servicio en un lenguaje de ontologías neutral, como BSDL o OWL, se traducirán automáticamente a objetos del modelo Java proporcionado por el desarrollador, y viceversa, sin necesidad de ninguna implementación adicional por su parte (ver Figura 7.8). Dado que actualmente únicamente se ha realizado una implementación de referencia de estas herramientas para la plataforma Java y para la plataforma Android, un cliente implementado en otro lenguaje necesitaría tratar directamente con instancias de conceptos BSDL para interaccionar con el Servicio UniDA, siendo su responsabilidad la adaptación de dichas instancias a un modelo interno proporcionado por el software del cliente.

Utilizando un modelo de objetos Java anotado con una referencia a una ontología y una descripción de la implementación en el descriptor BSDM en la que se

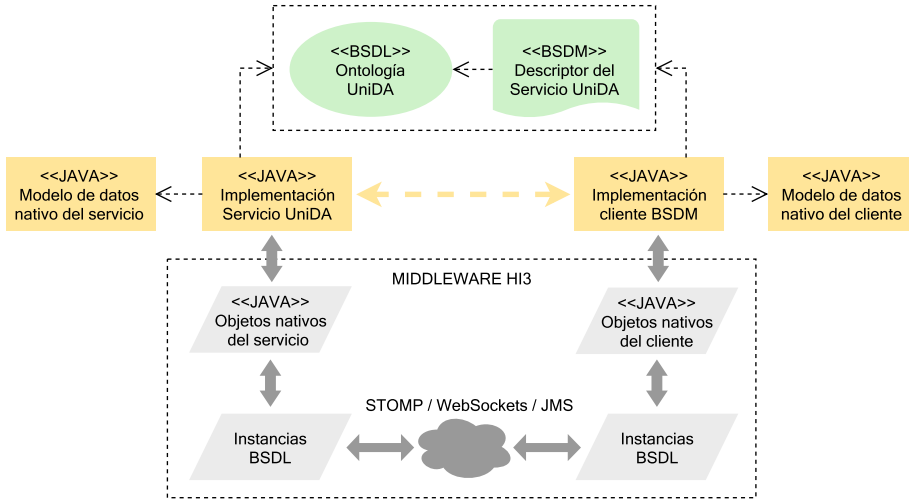



Figura 7.8: Interacción cliente-servicio en la implementación del Servicio UniDA.

indica la clase y método que implementan la funcionalidad se simplifica la programación de la implementación del servicio. Así, la ejecución de una funcionalidad del servicio UniDA se realizará de forma automática cuando se reciba una petición remota de un cliente, sin necesidad de que el desarrollador programe ninguna línea de código adicional.

En el Listado 7.5 se ilustra como se implementa la funcionalidad checkCurrentStatesValues del Servicio UniDA, que permite consultar los valores actuales de todos los estados de un dispositivo del entorno. Como vimos anteriormente, se trata de una funcionalidad asíncrona que produce un resultado puntual. Para resolverla, en este caso, el Servicio UniDA delega en una instancia de la Librería UniDA que se construye en la inicialización del servicio. Dado que la Librería UniDA tiene un funcionamiento fundamentalmente asíncrono, es necesario crear una *callback* para recibir las respuestas de ésta. Podemos ver en el código como se ha definido otra notación (@ResponseTo(...)) que permite indicar que el método del servicio que tiene la notación es el que va a proporcionar los resultados asíncronos, que han de ser enviados al cliente, correspondientes a una invocación previa de funcionalidad concreta. Utilizando esta notación, el desarrollador no necesita programar ninguna lógica para gestionar la interacción asíncrona con los clientes del servicio. Así, desde el punto de vista del desarrollador de un servicio la interacción ocurre de la siguiente forma:

- Para una funcionalidad dada, sabe que recibirá, automáticamente a través del grounding del servicio, una invocación en la clase y método que la implementa (definidos en la descripción de la implementación del modelo BSDM).
- Además, recibirá los Inputs de la funcionalidad como objetos nativos de su

Listado 7.5: Implementación de una funcionalidad del Servicio UniDA.

```
public class SemanticUniDA extends AJavaServiceImplementation
{
   private IUniDAInstantiationFacade unidaLibraryFacade;

   /** En la inicialización se instancia una fachada de acceso a la Librería UniDA. */
   public void initService() throws ServiceExecutionException
   {
      try {
         this.unidaLibraryFacade = new InMemoryUniDAInstantiationFacade();
         this.unidaLibraryFacade.initialize();
      } catch (InternalErrorException e) {
         throw new ServiceExecutionException("Error launching UniDA-Library", e);
      }
   }

   /** Implementación de la funcionalidad:
    *     "http://hi3project.com/unida/service/profile#checkCurrentStatesValues"
    */
   public void checkCurrentStatesValues(DeviceVO device)
   {
      IDevice unidaLibraryDevice =
         this.unidaLibraryFacade.getDeviceManageFacade().findById(device.getId());
      OperationTicket queryDeviceState =
         this.unidaLibraryFacade.getDeviceOperationFacade().asyncQueryDeviceStates(
               unidaLibraryDevice, new UniDALibraryDeviceOperationCallback());
   }

   @ResponseTo("checkCurrentStatesValues")
   public void processCurrentStatesValues(List<StateValueVO> values)
   {
      (. . .)

      // Utiliza la notación @ResponseTo(...) para saber que "values" debe ser procesado
      // como un resultado correspondiente a la funcionalidad indicada en la notación.
      this.processResultFor(requesterID, conversationID, values);
   }

   /** Callback para recibir las respuestas de la consulta a la Librería UniDA */
   class UniDALibraryDeviceOperationCallback implements IDeviceOperationCallback
   {
      @Override
      public void notifyQueryDeviceStatesResult(
                  OperationTicket ot, IDevice device, Collection<DeviceState> dss)
      {
         List<StateValueVO> values = new ArrayList<StateValueVO>();
         for (DeviceState ds : dss)
         {
            values.add(new StateValueVO(device, ds));
         }
         this.processCurrentStatesValues(values);
      }
   }

   (. . .)
}
```

modelo de datos local, gracias a las anotaciones que relacionan su modelo con las ontologías adecuadas.

- Si la funcionalidad es asíncrona, únicamente tiene que marcar con una notación un método en el que notificará que el servicio ha generado un resultado para una funcionalidad concreta. El middleware se encargará de gestionar el envío de cada resultado al cliente o clientes adecuados, a través del grounding y el canal correspondiente.
- De nuevo, el desarrollador no se tiene que preocupar de producir los resultados de una funcionalidad como instancias de la ontología correspondiente.

Por tanto, queda ilustrado como, utilizando las herramientas proporcionadas por el middleware orientado a servicios de la arquitectura HI3, se cumple en buena parte el objetivo de facilitar que los desarrolladores se centren en la implementación de nuevas funcionalidades, utilizando las tecnologías que conocen y manteniendo su código limpio de lógica de control relacionada con la comunicación.

Con lo ilustrado hasta el momento se abarcan todos los pasos relevantes que permitieron crear una implementación del Servicio UniDA en lenguaje Java para la implementación de referencia de la arquitectura HI3. Por último, esta implementación del Servicio UniDA se puede empaquetar en un único archivo .BSD utilizando las utilidades correspondientes del framework Vineyard. En este caso, el archivo empaquetado contiene el descriptor del servicio (serviceUnida.bsdm), los binarios de la implementación Java del servicio (implementation/serviceImplUnida.jar) y otras librerías necesarias para la implementación (implementation/lib/...). El servicio así empaquetado puede ser instalado en un contenedor Vineyard para su ejecución.

#### Implementación de un cliente del Servicio UniDA

Por último, en este apartado se ilustrar como se implementa un cliente (servicio o aplicación) que desee descubrir y utilizar funcionalidades ofrecidas por el Servicio UniDA, utilizando el middleware de la arquitectura HI3.

En primer lugar, un cliente deberá construir un objetivo de búsqueda (una RequestedFunctionality de BSDM) con el que describa la funcionalidad en la que está interesado. Este objetivo podrá construirse tanto programáticamente como a través de un descriptor en lenguaje BSDL como el mostrado en el Listado 7.6. En el ejemplo concreto que utilizamos aquí, se describe un cliente interesado en una funcionalidad que le permita obtener notificaciones continuadas del cambio de valor en un estado de un dispositivo especificado a través de la ontología UniDA. Si el cliente realiza esta búsqueda en un registro de servicios en el que esté registrado algún Servicio UniDA, obtendrá como respuesta dos funcionalidades que encajan con dicha descripción: checkCurrentStateValue y subscribeToChangesInState. Sin embargo, la primera opción proporciona un resultado puntual y la segunda opción

Listado 7.6: Ejemplo de un objetivo de búsqueda adecuado para una funcionalidad del Servicio UniDA.

```
<instance of="http://hi3project.com/broccoli/bsdm/profile#requestedFunctionality"
   URI="http://hi3project.com/unida/client#requestForSubscriptionToStateChange">

   <with property="input">
      <with property="name" value="Device"/>
      <with property="type"
     valueURI="http://hi3project.com/unida/ontology#DeviceVO"/>
   </with>
   <with property="input">
      <with property="name" value="State"/>
      <with property="type"
     valueURI="http://hi3project.com/unida/ontology#StateValueVO"/>
   </with>
   <with property="output">
      <with property="name" value="State value data"/>
      <with property="type"
     valueURI="http://hi3project.com/unida/ontology#StateValueVO"/>
   </with>
</instance>
```

es una funcionalidad de subscripción como demanda el cliente.

En el Listado 7.7 se ilustra la interfaz que proporciona el middleware SOA de la arquitectura HI3 para realizar una búsqueda de funcionalidades a través de un registro de servicios HI3. Podemos ver como a través de la clase *ClientRequester* se puede indicar el descriptor de un objetivo de búsqueda y las respuestas llegarán de forma asíncrona en forma de elementos *FunctionalitySearchResult* del modelo BSDM.

En el Listado 7.8 se incluye el código que permite al cliente invocar una de las funcionalidades a partir de un elemento FunctionalitySearchResult obtenido en la búsqueda. En este caso se correspondería con la funcionalidad subscribeToChangesInState. Dado que la funcionalidad declara dos entradas obligatorias correspondientes a conceptos de la ontología UniDA, es necesario proporcionar como Inputs instancias de los mismos. Para ello, el cliente puede utilizar una herramienta de tipo IParameterConverter, proporcionada por el framework Broccoli, que permite convertir automáticamente objetos de un modelo nativo anotado (en este caso en lenguaje Java) a instancias de una ontología BSDL (también se soporta OWL). Una vez construidos los Inputs de la funcionalidad, simplemente se ejecuta a través del descriptor, proporcionando un *callback* en el que recibir asíncronamente los resultados de ésta.

De esta forma, otros servicios y aplicaciones pueden descubrir dinámicamente funcionalidades de instancias del Servicio UniDA que les proporcionen acceso a los dispositivos disponibles en un momento dado. Por otro lado, los desarrolladores

Listado 7.7: Código para la búsqueda de funcionalidades a través de un resgistro.

```
final ClientRequester clientRequester = new ClientRequester(clientName);

clientRequester.askForFunctionality(
   new SemanticLocator(requestedFunctionalityDescriptor),
   new IAsyncMessageClient() {

      @Override
      public void receiveMessage(IMessage msg) throws ModelException
      {
         if (msg instanceof SearchFunctionalityResultMessage)
         {
            SearchFunctionalityResultMessage sfr =
               (SearchFunctionalityResultMessage) msg;

            for (IFunctionalitySearchResult searchResult : sfr.getSearchResults())
            {
               // Hacer algo con las funcionalidades
               // que ha dado como resultado la búsqueda.
            }
         }
      }
   });
```

pueden implementar aplicaciones y servicios portables a diferentes entornos físicos, evitando atar su implementación a un entorno y a unas tecnologías concretas.

### 7.2.4. Visión global del acceso al entorno físico en la arquitectura HI3

Toda implementación de una arquitectura HI3 debe proporcionar una solución que permita a los desarrolladores de nuevos componentes funcionales lidiar convenientemente con la heterogeneidad del entorno físico. Esta solución pasará, en el nivel más alto de abstracción, por la interacción con instancias del Servicio UniDA. Así, vimos que el Servicio UniDA ha sido construido con dos objetivos fundamentales: i) lidiar con el problema de la heterogeneidad de middleware no interoperable en el ámbito de la IoT; ii) proporcionar a los desarrolladores un elemento de abstracción de más alto nivel que les evite depender de frameworks con capacidades de interoperabilidad y acceso ubicuo más limitadas. En concreto, en la implementación de referencia de la arquitectura HI3 que se ha realizado, se incluye una implementación de un Servicio UniDA capaz de operar con redes de instrumentación abstraídas, a más bajo nivel, a través de la Librería UniDA y el framework DAAF.

No obstante, creemos que no todos los comportamientos entre dispositivos deberían resolverse pasando por elementos de alto nivel, como servicios y aplicaciones. Algunos comportamientos que son fundamentalmente reactivos deberían po-

Listado 7.8: Ejecución de una funcionalidad obtenida a través de una búsqueda en un registro de servicios.

```
public void executeSubscriptionToDeviceStateEvents(
        IFunctionalitySearchResult searchResult, DeviceVO device, StateValueVO state)
{
    final IParameterConverter parameterConverter = this.getParameterConverter();

    // Construir parámetros de entrada (BSDM Inputs) para ejecutar funcionalidad.
    Collection<IAnnotatedValue> inputs = new ArrayList<>();
    inputs.add(parameterConverter.createAnnotatedValue(device));
    inputs.add(parameterConverter.createAnnotatedValue(state));

    // Ejecución de una funcionalidad de subscripción a partir del descriptor BSDM.
    searchResult.getServiceDescription().subscribeTo(
        searchResult.getAdvertisedFunctionalityName(),
        inputs,
        new IAsyncMessageClient()
        {
            @Override
            public void receiveMessage(IMessage msg) throws ModelException
            {
                if (msg instanceof FunctionalityResultMessage)
                {
                    FunctionalityResultMessage resultMsg =
                        (FunctionalityResultMessage) msg;

                    if (resultMsg.getResult().size() > 0)
                    {
                        IResult result = resultMsg.getResult().iterator().next();
                        if (result instanceof IOutputValue)
                        {
                            Object outputToObject =
                                parameterConverter.outputToObject((IOutputValue) result);

                            if (outputToObject instanceof StateValueVO)
                            {
                                // Llegó un resultado asíncrono (BSDM OutputValue)
                                // correspondiente a la ejeución de la funcionalidad.
                                notifyNewValue((StateValueVO) outputToObject);
                            }
                        }
                    }
                }
            }
        });
}
```

der resolverse de forma distribuida y autónoma entre los propios dispositivos. Así, por ejemplo, un usuario podría desear que cuando encienda su reproductor multimedia para ver cine en su salón, automáticamente se regule la iluminación a su gusto y se baje la persiana. En este caso, sería deseable que este comportamiento quedase fijado como parte de su red de instrumentación, sin depender de una aplicación que ejecute dicha configuración. Las características interesantes de es-

te ejemplo son: i) el comportamiento deseado involucra dispositivos heterogéneos que a priori no podrían interaccionar por sí mismos (un reproductor multimedia y una persiana); ii) el comportamiento ocurre dentro de un mismo entorno lógico, una casa familiar. La primera característica implica que se necesitará un framework que resuelva el acceso entre dispositivos heterogéneos. La segunda nos habla de que es razonable que los usuarios de ese entorno puedan fijar comportamientos autónomos en el mismo.

Así, la Librería UniDA, tal y como la introdujimos en la Sección 7.2.1 se ha extendido, en este trabajo, para soportar la creación de reglas sencillas de comportamiento autónomo entre los dispositivos de una red de instrumentación. Dichas reglas permiten que ante un cambio de estado en un dispositivo se produzcan actuaciones relacionadas con estados de otros dispositivos. Se admite la definición de condiciones sencillas para el disparador de un cambio de estado. Estas reglas pueden crearse de forma centralizada a través de una fachada de la Librería UniDA, pero se almacenan y ejecutan de forma distribuida en la red de dispositivos y gateways UniDA. Una regla que se activa por el cambio de un estado de un dispositivo es almacenada en el propio dispositivo, en caso de ser compatible con el protocolo UniDA, o en el gateway que adapta una tecnología concreta no compatible al modelo UniDA. El dispositivo o gateway desde el que se dispara la regla es responsable de enviar de forma autónoma una operación de actuación sobre los dispositivos a los que afecta dicha regla.

En un escenario en el que ocurren comportamientos autónomos entre dispositivos de una red de instrumentación UniDA, ni el Servicio UniDA ni las aplicaciones necesitan intervenir en la ocurrencia de dichos comportamientos, una vez que han sido definidos. Esto aumenta la robustez de la solución, puesto que podría caerse el Servicio UniDA, la instancia de la Librería UniDA o aplicaciones de alto nivel sin afectar a la ejecución de los comportamientos autónomos definidos.

Por otro lado, para desarrollar comportamientos de más alto nivel, utilizando las capacidades de la Capa de Dispositivos, otros servicios y aplicaciones pueden descubrir dinámicamente instancias del Servicio UniDA que les proporcionen acceso a los elementos físicos de entornos tanto locales como remotos, favoreciendo la movilidad y adaptación al contexto de ejecución. Así, por ejemplo, una aplicación HI3 ejecutándose en un *smartphone* de un usuario podría estar permanentemente utilizando un Servicio UniDA que le permite el acceso a los dispositivos del usuario en su hogar, pero cuando el usuario entra en el edificio donde trabaja y se conecta a su red WiFi esta aplicación podría descubrir un nuevo Servicio UniDA a través del que interaccionar con el nuevo entorno en el que se encuentra el usuario.

Vemos, por tanto, como utilizando los diferentes elementos de la Capa de Dispositivos es posible acceder de forma uniforme a capacidades para controlar el entorno físico desde diferentes niveles de abstracción según las necesidades.

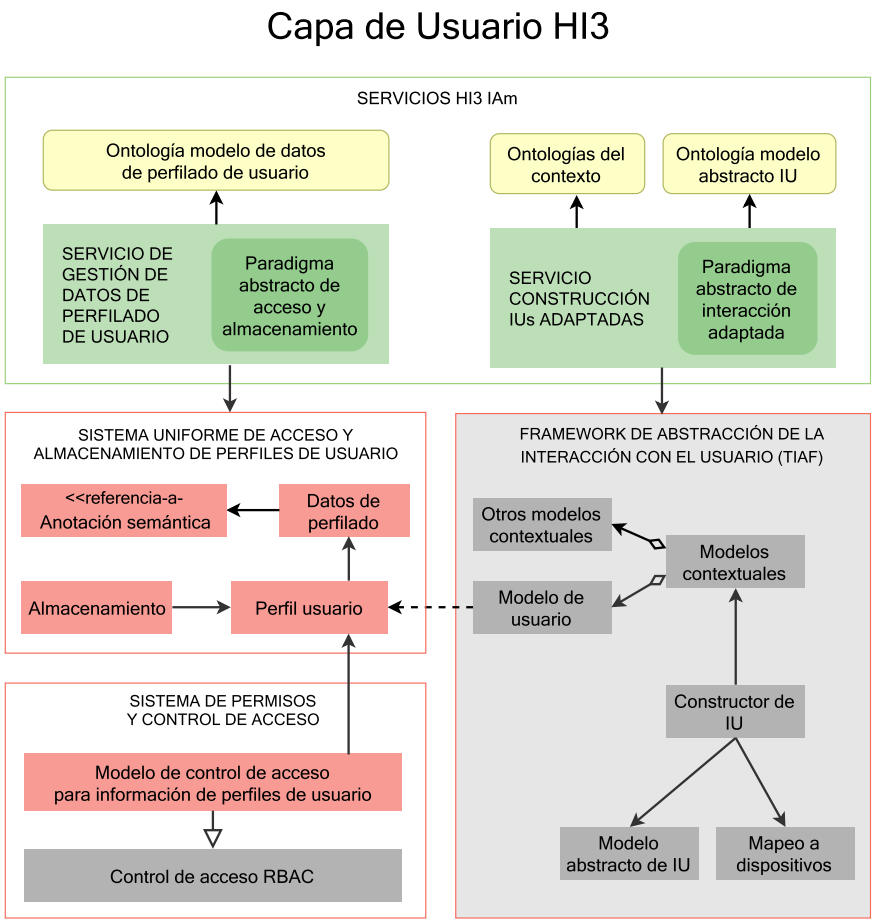



Figura 7.9: Elementos principales de la Capa de Usuario de la arquitectura HI3.

## 7.3. Creando un entorno humanizado para el usuario

En el Capítulo 4 vimos como en el núcleo central alrededor del que establecimos los requisitos y objetivos fundamentales de la arquitectura HI3 situamos a los usuarios y a los entornos en los que éstos realizan su actividad. En esta sección trataremos una serie de aspectos que consideramos clave para lograr que los usuarios perciban los entornos y los sistemas de IAm como humanizados y adaptados a sus necesidades.

Que la tecnología y los sistemas de IAm se adapten al usuario es un objetivo muy amplio que involucra diversos aspectos y debe ser abordado desde distintas perspectivas, constituyendo en su conjunto uno de los mayores retos de la IAm. Como parte de este trabajo se tratan algunos de esos aspectos y se propone un conjunto de

soluciones, que conceptualmente se corresponden con la Capa de Usuario de la arquitectura HI3, y que se organizan de acuerdo a la estructura mostrada en la Figura 7.9. Como punto de partida, abordaremos la necesidad de nutrir a las aplicaciones de IAm con información del perfil de los usuarios para posibilitar una adaptación inteligente de su comportamiento a las necesidades, características y habilidades de éstos. Así, se propone un modelo uniforme para el acceso y almacenamiento de información de perfilado de usuario proveniente de fuentes heterogéneas, de modo que aplicaciones concretas puedan acceder a ella de forma homogénea y ubicua. Considerando que, con carácter general, cuando hablamos de perfil estamos refiriéndonos a información sensible de los usuarios, también se propone un mecanismo general para gestionar el control de acceso a este tipo de información.

Además, en la Capa de Usuario HI3 también se considera otro aspecto central de la IAm, la interacción natural y adaptada a los usuarios. Así, para incorporar este tipo de capacidades a la arquitectura HI3, se parte de la integración del framework Dandelion TIAF [TIAF, 2015]. Este framework proporciona una implementación en lenguaje Java de un sistema distribuido con capacidades para la construcción de interfaces de usuario (IU) abstractas que son resueltas en tiempo de ejecución a través de los dispositivos de interacción más adecuados para un contexto determinado. Sobre el framework TIAF se desarrolla el Servicio UIB (del inglés User Interface Builder), que, gracias a las capacidades de dicho framework, proporciona la funcionalidad y las ontologías necesarias para que las aplicaciones resuelvan su interacción con el usuario de forma abstracta y ubicua.

A lo largo de la sección se hará una descripción detallada de este conjunto de soluciones, ilustrando como proporcionan herramientas que facilitan el desarrollo de aplicaciones capaces de adaptar su comportamiento a distintas situaciones y usuarios.

### 7.3.1. Haciendo accesible el perfil del usuario

Hemos expuesto repetidas veces que uno de los objetivos centrales de los sistemas de IAm es que éstos sean capaces de adaptar su comportamiento a los usuarios y su contexto. Para ello es imprescindible disponer de información de las características, preferencias y costumbres de los usuarios, y ser capaces de relacionarlas dinámicamente con el contexto de ejecución de un sistema determinado. Decimos que esta información en su conjunto define el perfil de un usuario. Aunque puede contener información estática, en general en un entorno de IAm hemos de considerar el dinamismo inherente al perfil de un usuario. Y este dinamismo ha de entenderse desde dos puntos de vista: i) cambios en información previamente conocida del usuario; ii) incorporación continua de nueva información que permita un refinamiento progresivo del perfil de cada usuario.

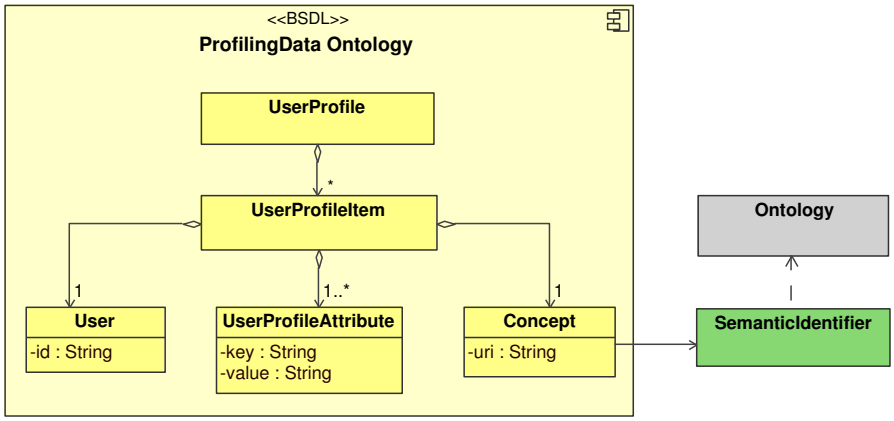



Figura 7.10: Una ontología para el modelado uniforme de información del perfil de los usuarios.

En entornos de IAm, dónde los contextos de ejecución son enormemente amplios, complejos y dinámicos, y dónde además la tecnología debe hacerse fundamentalmente invisible, los procesos de obtención y aprendizaje de características, hábitos y preferencias de los usuarios tendrán que hacerse principalmente a través de la monitorización de su actividad y de las propias interacciones del usuario con el sistema. Además, en estos entornos, donde múltiples aplicaciones conviven y se integran en el contexto de los usuarios, es posible y necesario que el aprendizaje de los perfiles de usuario sea distribuido y tenga su origen en distintas fuentes. Este tipo de necesidades deben ser consideradas en la arquitectura HI3, aunque el estudio y desarrollo de sistemas concretos de perfilado de usuarios excede el alcance de esta tesis. Sin embargo, en el ámbito del presente trabajo, se propone, como parte de la arquitectura HI3, un modelo general para facilitar el acceso e integración de información del perfil de los usuarios proveniente de diferentes frameworks y sistemas heterogéneos de perfilado. De este modo, otras aplicaciones y servicios concretos podrán sacar partido de ese tipo de información para adaptar su comportamiento al usuario.

Como primer paso de la solución propuesta, se trata de definir un modelo semánticamente descrito y suficientemente genérico que sea eficiente y eficaz para manejar información proveniente de orígenes heterogéneos y organizada de formas que a priori se desconocen. Para ello se ha definido una ontología (ver Figura 7.10) que facilitará que diversos sistemas que realicen perfilado de usuario puedan exportar elementos de información relacionados con el perfil de los usuarios utilizando los siguientes conceptos básicos:

- **UserProfileItem.** Representa un elemento de información del perfil de un usuario que incluye una serie de atributos relacionados con un mismo concep-

to. El perfil global de un usuario concreto estará formado por todos los UserProfileItem correspondientes a dicho usuario. Estos elementos típicamente serán generados y modificados a partir de distintas fuentes de forma progresiva en el tiempo.

- **User.** Representa un identificador del usuario al que pertenece un elemento de información UserProfileItem.
- **Concept.** Representa una referencia a un concepto de una ontología externa. Permite anotar semánticamente un UserProfileItem sin imponer una ontología concreta para ello.
- **UserProfileAttribute.** Representa un par clave-valor que permite especificar un atributo concreto del perfil de un usuario relacionado con un Concepto concreto. Constituye una representación de la información plana e independiente de tecnologías, que además puede facilitar la optimización de procesos de búsqueda de datos. Idealmente los UserProfileAttribute de un UserProfileItem tendrán una correspondencia semántica con la descripción del Concept en la ontología correspondiente.

Este modelo permite el dinamismo de la información en cuanto a su estructura, ya que en cualquier momento es posible añadir nuevos atributos a un elemento de información. No sólo los datos se van completando a lo largo del tiempo, sino que los propios conceptos permanecen siempre abiertos y se van construyendo a medida que, desde diferentes fuentes, se aportan nuevos atributos que los definen. Esta es una de las principales características buscadas para este modelo, en oposición a la estructura estática de los datos, definida de antemano por el programador, presente en la mayoría de aplicaciones.

De igual forma que este modelo común permite que diferentes sistemas dentro de un entorno HI3 exporten información relacionada con perfiles de usuarios, otros componentes podrán acceder a esta información para adaptar su comportamiento al usuario, permaneciendo agnósticos del origen de la misma. Para ello, como parte de la Capa de Usuario de la arquitectura HI3 se construye un sistema homogéneo de almacenamiento y recuperación de información de perfilado de usuarios fundamentado en un servicio semántico (Servicio UPDS, del inglés User Profiling Data Service) que utiliza la anterior ontología. En la Figura 7.11 se ilustra la arquitectura global del sistema planteado.

Las funcionalidades principales propuestas para el Servicio UPDS son las siguientes:

- **RegisterData.** Añade un ProfileDataItem al almacenamiento de información de perfilado de usuarios.
- **DeleteData.** Elimina la información relacionada con el perfil de un usuario indicado como Input.

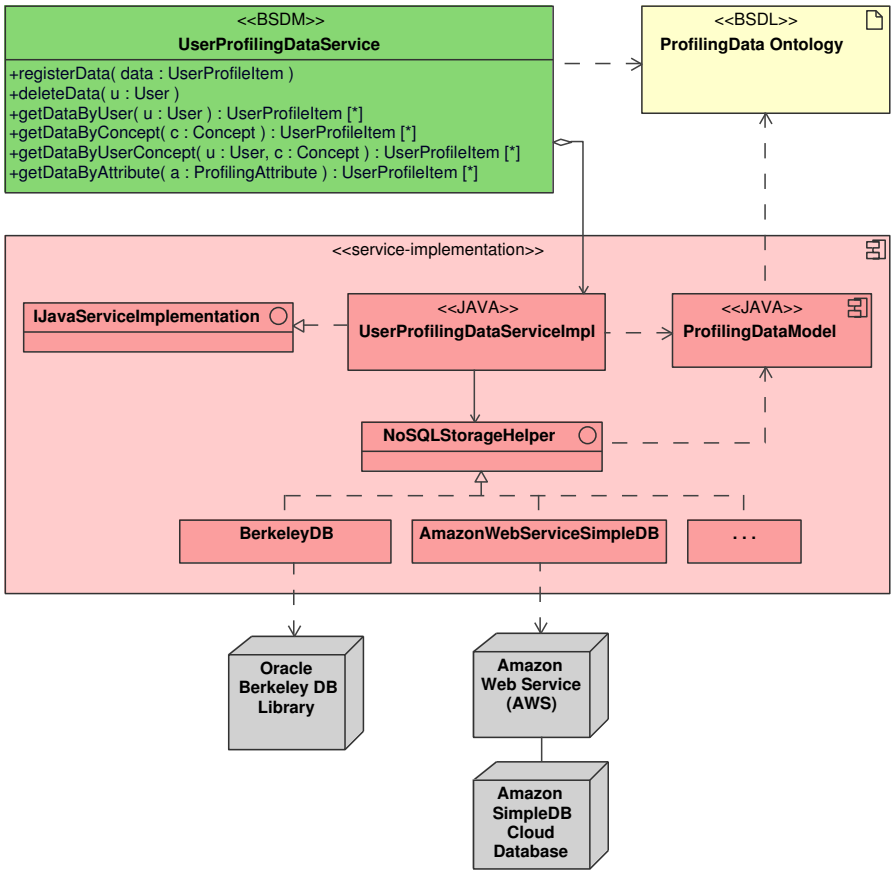



Figura 7.11: Arquitectura del sistema homogéneo de almacenamiento y recuperación de información de perfilado de usuarios.

- **GetDataByUser.** Obtiene como Output la lista de elementos de información relacionados con un usuario.
- **GetDataByConcept.** Obtiene como Output la lista de elementos de información relacionados con un concepto de una ontología externa. Por ejemplo, consultar la información relacionada con capacidades visuales de las personas.
- **GetDataByUserAndConcept.** Igual a la anterior pero filtrada por usuario.
- **GetDataByAttribute.** Obtiene como Output la lista de elementos de información que contienen un atributo dado con un valor específico. Por ejemplo, obtener información de todos los usuarios para los que se cumpla "wakeUpHabit=early".

El modelo que hemos propuesto demanda una solución para la persistencia de

Listado 7.9: Ejemplos de interacción con una base de datos SimpleDB.

```
# Crear elementos con atributos clave–valor
simpledb put mydomain item1 key1=value1 key2=value2

# Añadir nuevos atributos a un elemento
simpledb put mydomain item1 key3=value3

# Obtener todos los elementos y sus atributos que se corresponden con una consulta dada:
simpledb select "select * from mydomain where key1=value1"
```

la información que se corresponda con las características de éste, especialmente en cuanto al dinamismo en la estructura de la información. Sistemas tradicionales, como las bases de datos relacionales, se adaptan mal a un contexto en el cuál se desconoce de antemano el tipo y la estructura de los datos. Así, en este caso, se propone utilizar una base de datos NoSQL de tipo Clave/Valor [Pokorny, 2013]. Este tipo de bases de datos la información se organiza en elementos compuestos de tuplas atributo-valor, y no es necesario predefinir ni cambiar esquemas si se añaden atributos posteriormente. Permiten lidiar eficientemente con grandes cantidades de datos no estructurados estáticamente, favoreciendo la escalabilidad horizontal demandada por nuestro modelo. Ejemplos de sistemas de bases de datos NoSQL Clave/Valor son Amazon SimpleDB [SimpleDB, 2015], Redis [Redis, 2015], Oracle Berkeley DB [BerkeleyDB, 2015] y LevelDB [LevelDB, 2015]. Amazon SimpleDB puede, además, ser utilizada como almacenamiento en la nube, siendo accesible a través de un servicio web.

A modo ilustrativo, en el Listado 7.9 se describen algunos ejemplos de interacción con una base de datos Amazon SimpleDB. Utilizando esta base de datos, para almacenar información correspondiente al modelo propuesto bastaría con crear elementos en los que dos tuplas clave-valor estén reservadas para almacenar los atributos correspondientes al identificador de usuario y a la referencia al concepto de una ontología externa.

Al utilizar un servicio HI3 para proporcionar acceso a la información del perfil se facilita la movilidad de aplicaciones. Éstas pueden realizar un descubrimiento de Servicios UPDS del entorno a través de los que pueden obtener información que les permita adaptar su comportamiento a un nuevo contexto. Además, esta solución promueve el desacoplamiento entre los componentes que obtienen y producen información del perfil de los usuarios y los componentes que la utilizan.

Existe, sin embargo, otro factor muy importante a tener en cuenta en este escenario: la privacidad y la seguridad en el acceso a información de los usuarios que puede ser sensible. Así, como norma general, el Servicio UPDS debe operar en un entorno controlado en el que el acceso a recursos de información está gobernado por un sistema de permisos. En concreto, desde este trabajo se propone la

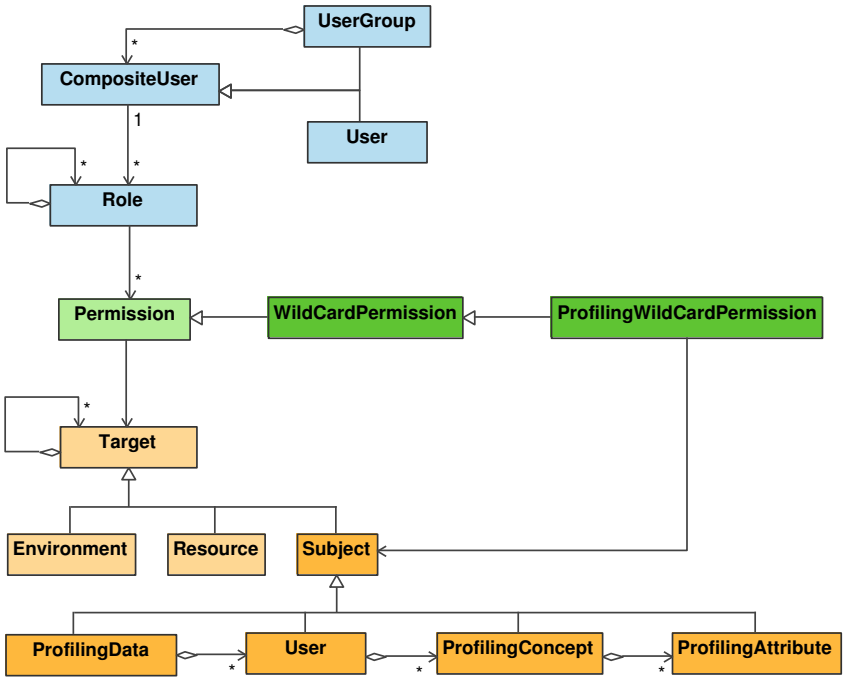



Figura 7.12: Modelo de control de acceso basado en roles propuesto para la arquitectura HI3.

utilización de un sistema de control de acceso basado en roles (RBAC, del inglés Role-Based Access Control) [Ferraiolo et al., 2003]. En este tipo de sistemas, los permisos para realizar ciertas operaciones son asignados a roles. A los usuarios se les asignan roles, a través de los que adquieren sus permisos. Finalmente los permisos se declaran sobre aquellos recursos a los que se quiere limitar el acceso.

En la Figura 7.12 se muestra el modelo general propuesto para gestionar el control de acceso a recursos en la arquitectura HI3. Se incluye una visión del modelo general con la extensión de aquellos conceptos necesarios para regular el acceso a los elementos de información del Servicio UPDS (elementos con un color más intenso en el diagrama). Tomando como inspiración las recomendaciones del sistema XACML (del inglés, eXtensible Access Control Markup Language) [XACML, 2013], en el modelo propuesto se dividen los elementos sobre los que se pueden aplicar los permisos (Target) en diferentes categorías. Así, se propone un modelo jerárquico para los Target, de modo que los permisos de un elemento padre se heredan en los hijos. En concreto, para controlar el acceso a la información de nuestro modelo de datos de perfilado de usuario, se define una jerarquía de cuatro Target (ProfilingData, User, ProfilingConcept y ProfilingAttribute) sobre los que se pueden aplicar permisos, que se corresponden directamente con la organización de la información

Listado 7.10: Permisos estilo Wild-Card de Apache Shiro.

```
# Permiso sobre un 'subsubrecurso' hijo de 'subrecurso' que a su vez es hijo de 'recurso'
recurso:subrecurso:subsubrecurso

# Permiso sobre todos los hijos de 'recurso'
recurso:*

# Permiso sobre un 'susubrecurso' que sea hijo de cualquier hijo de 'recurso'
recurso:*:subsubrecurso
```

Listado 7.11: Definición de permisos estilo Wild-Card para la información del modelo de datos de perfilado de usuario de la arquitectura HI3.

```
# Permiso para acceder a un atributo concreto relacionado con un concepto y de un usuario.
ProfilingData:userId01:conceptId01:attributteKey01

# Permiso para acceder a todos los datos de un usuario concreto.
ProfilingData:userId02:*:*

# Permiso para acceder a todos los atributos de todos los usuarios
# relacionados con un concepto concreto.
ProfilingData:*:conceptId03:*

# Permiso para acceder a todos los datos de perfilado de todos los usuarios.
ProfilingData:*:*:*
```

en el modelo de datos de perfilado.

Como formato para la definición de los permisos se propone un estilo inspirado en otro conocido framework de control de acceso, Apache Shiro [ApacheShiro, 2015]. Éste propone un modelo de permisos jerárquicos, denominado WildCardPermission, que permite definir permisos con la notación mostrada en el Listado 7.10. Utilizando este formato sobre el modelo de control de acceso que hemos propuesto aplicado al modelo de datos de perfilado de usuario, se pueden definir permisos jerárquicos de la forma mostrada en el Listado 7.11.

De acuerdo con el carácter privado y personal de los datos relacionados con el perfil de una persona, se propone que a su conjunto se les aplique un acceso restringido. Por tanto, cualquier componente que desee acceder a datos de este tipo a través del Servicio UPDS debe exhibir un permiso, de tipo Wild-Card adecuado para la información correspondiente a la consulta que pretende realizar.

Además, dado que está basado en los principios generales de un sistema RBAC, este modelo de control de acceso planteado para la información de los perfiles de usuario es fácilmente integrable en sistemas globales de seguridad ya existentes que incluyen mecanismos de autenticación y herramientas para la gestión de usuarios, roles y permisos.

### 7.3.2. Construyendo interfaces de usuario adaptadas

Recordemos que uno de los objetivos principales de un sistema de IAm es que la tecnología prácticamente desaparezca para los usuarios y la interacción con éstos sea natural, integrada en el entorno y escasamente intrusiva. Este tipo de interacción natural debe utilizar conceptos cotidianos que no obliguen a los usuarios a aprender nuevos conceptos relacionados con las tecnologías informáticas. Además, la interacción debe ser adaptada a las características de cada usuario y de forma general a la situación concreta en la que se produce dicha interacción.

Los escenarios donde un sistema de IAm puede operar son enormemente diversos y complejos, y esto también tiene su reflejo en la forma de abordar la interacción con el usuario. Cada escenario puede disponer de recursos de interacción diferentes, características diferentes (iluminación, ruido, restricciones de seguridad, etc.), y a su vez en un momento dado puede involucrar a usuarios con diferentes características y habilidades. Por tanto, a menudo se hace necesario que los sistemas de IAm utilicen realizaciones diferentes de su IU en diferentes escenarios.

En este apartado, se aborda esta problemática, tratando de incorporar a la arquitectura HI3 soluciones que faciliten la creación de interfaces de usuario capaces de adaptarse en tiempo de ejecución a las características del entorno y de los usuarios. Para ello, se propone la utilización de un framework existente, denominado Dandelion TIAF (Threefold Interaction Abstraction Framework) [Varela et al., 2012] [Varela, 2015]. El framework TIAF proporciona una arquitectura distribuida en la que las interfaces de usuario están desacopladas tanto lógica como físicamente de los recursos que, en último término, llevan a cabo la interacción. La implementación de las IUs se realiza utilizando un grupo de conceptos de interacción abstractos. En tiempo de ejecución, estos conceptos abstractos son mapeados a recursos de interacción reales. Este mapeo es realizado por una serie de algoritmos que, utilizando modelos de contexto del escenario, se encargan de generar una IU adaptada al escenario a partir de la IU abstracta.

El framework TIAF dispone de una implementación Java que permite su despliegue en un entorno de interacción distribuido correspondiente al ámbito de una red local. Para la integración de este framework en la arquitectura HI3 se propone un servicio para la construcción dinámica de IUs adaptables (Servicio UIB, del inglés User Interface Builder) capaz de exportar la funcionalidad deseada de forma ubicua. La solución se completa con la definición de una serie de modelos conceptuales comunes utilizando un lenguaje de ontologías neutral.

En la Figura 7.13 se ilustra la arquitectura general de la solución propuesta. Vemos como el Servicio UIB proporciona dos grupos lógicos de funcionalidades semánticamente descritas:

- **Paradigma Abstracto de Interacción con el Usuario.** Este paradigma pro-

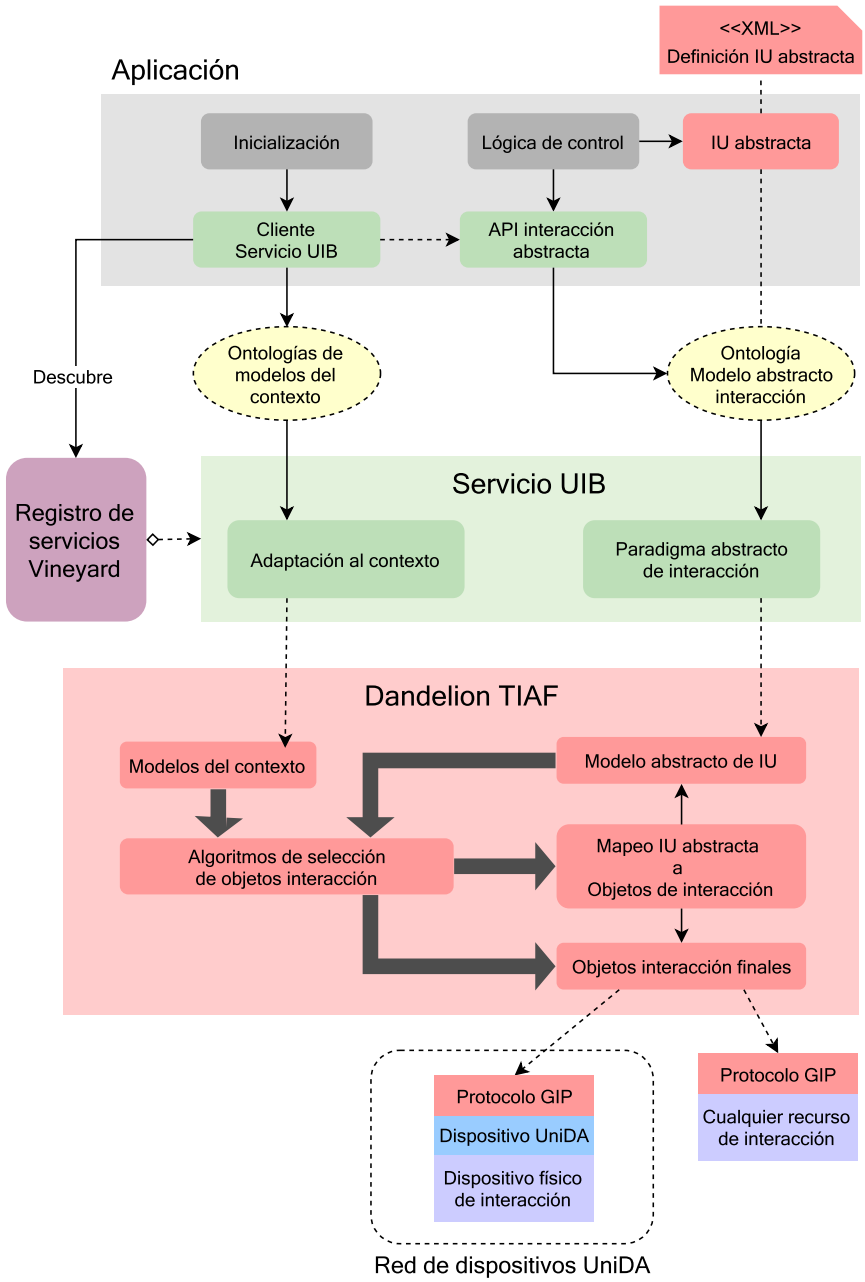



Figura 7.13: Arquitectura general del sistema de interacción adaptada al contexto en la arquitectura HI3.

porciona a los desarrolladores de aplicaciones un conjunto de operaciones genéricas sobre las que programar su interacción con una IU también definida de forma abstracta. En tiempo de ejecución, el framework TIAF mapeará la interfaz genérica a recursos concretos de interacción existentes en el entorno y trasladará las operaciones del Paradigma Abstracto de Interacción con el Usuario a acciones equivalentes en dichos recursos de interacción.

- **Capacidad de adaptación dinámica al contexto.** El servicio UIB también provee de funcionalidades para que las aplicaciones proporcionen, en tiempo de ejecución, información acerca del contexto en el que se va a realizar la interacción con el usuario. Esta información junto con la definición abstracta de la IU permite al framework TIAF realizar un mapeo dinámico a los recursos de interacción concretos más adecuados al contexto.

Desde el punto de vista de una aplicación, el Servicio UIB le proporciona mecanismos para resolver la interacción con el usuario de forma ubicua y con independencia de las tecnologías concretas utilizadas por la implementación del framework TIAF. Así, volviendo a fijarnos en la Figura 7.13, vemos como una aplicación descubrirá en el entorno un servicio UIB a través del registro de servicios de su plataforma y a continuación podrá utilizar la funcionalidad adecuada para proporcionar la información contextual que le permitirá crear una interfaz de usuario adaptada al escenario. Para ello, el servicio ofrece una serie de ontologías que permiten a las aplicaciones crear los modelos del contexto en un lenguaje neutral. Por otro lado, el desarrollador habrá definido su interfaz de usuario de forma abstracta y habrá programado la lógica de control de su aplicación para que resuelva las acciones de interacción con el usuario a partir de elementos de la IU abstracta y operaciones correspondientes al Paradigma Abstracto de Interacción con el Usuario. En tiempo de ejecución se enlazarán dichas operaciones con las funcionalidades concretas de un Servicio UIB descubierto a través del registro.

En la Figura 7.14 se muestra un mayor detalle de los modelos y ontologías necesarios para interaccionar con el Servicio UIB, así como de las funcionalidades concretas que éste proporciona. En primer lugar, para resolver la interacción de forma abstracta es necesario disponer de un modelo de los elementos genéricos sobre los que se construye la interfaz de usuario abstracta. Así, el Modelo Abstracto de Interacción proporciona una descripción genérica de la interfaz de usuario en términos de sus capacidades. El framework TIAF propone la utilización de un Modelo Abstracto de Interacción basado en el Abstract User Interface Model del proyecto UsiXML [UsiXML, 2015], propuesto por el W3C para la definición de un Lenguaje de Especificación de Interfaces estándar para interfaces de usuario abstractas. En consecuencia, como parte del Servicio UIB se construye una ontología para este modelo. El concepto central es el AbstractInteractionUnit (AIU). Es una representación general de un componente de interacción, asimilable a un *widget* en la interfaces de usuario convencionales. Cada AIU agrupa a un conjunto de elementos básicos de

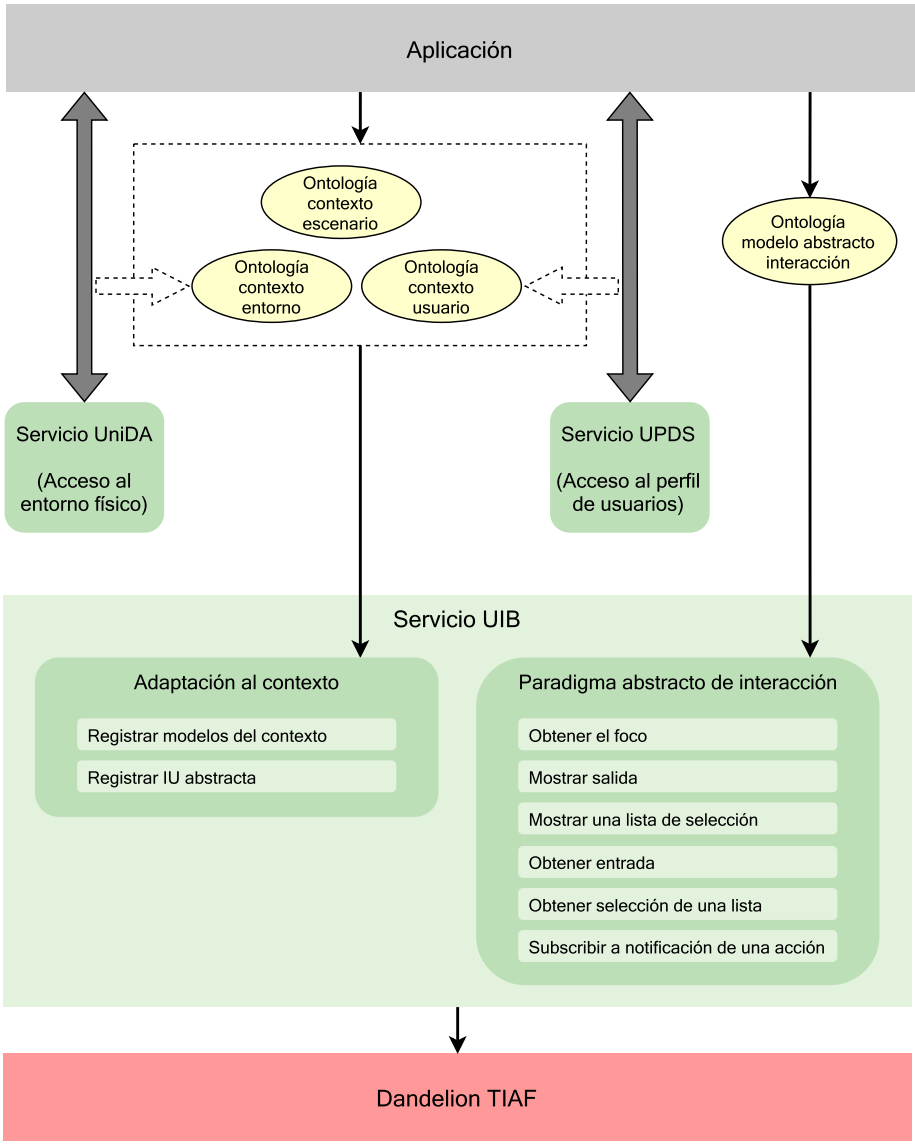



Figura 7.14: Detalle de la interacción de una aplicación con el Servicio UIB.

interacción denominados Facet (elementos de entrada, elementos de salida, listas de selección y eventos). Además, los AIU pueden componerse para organizarse en jerarquías.

El Modelo de Interacción Abstracto permite la definición de la organización de una IU abstracta, que cualquier aplicación puede enviar al Servicio UIB a través de la ontología definida. Esta descripción de la organización en elementos de la

IU abstracta necesita ser combinada con un conjunto de acciones de interacción sobre dichos elementos. De acuerdo con el modelo de UsiXML y TIAF se definen los siguientes tipos de interacciones: Entrada de datos del usuario, Salida de datos, Selección por parte del usuario de un elemento de entre una lista y obtener el Foco (la atención del usuario). En el caso de la realización concreta del Servicio UIB, se propone un conjunto de operaciones equivalente exportado a través de un conjunto de funcionalidades del servicio, componiendo en su conjunto lo que hemos denominado como Paradigma Abstracto de Interacción (ver Figura 7.14). Estas funcionalidades utilizan los conceptos de la ontología Modelo Abstracto de Interacción para describir sus entradas y salidas:

- **Obtener foco.** Todos los elementos de interacción permiten la opción de obtener el foco sobre ellos. Se resuelve con una funcionalidad de tipo AdvertisedFunctionality del modelo BSDM, que recibe como parámetro de entrada el elemento de interacción (Facet) concreto.
- **Mostrar salida.** Acción para enviar datos a un Facet de Entrada/Salida. Se resuelve con una funcionalidad de tipo AdvertisedFunctionality, que recibe como entradas el Facet correspondiente y los datos de salida.
- **Mostrar una lista de selección.** Acción para presentar una colección concreta de datos en un Facet Lista de Selección. Se resuelve con una funcionalidad de tipo AdvertisedFunctionality, que recibe como entradas el Facet concreto, la colección de datos y el valor por defecto.
- **Obtener entrada.** Acción que indica que se requiere de entradas por parte del usuario en un elemento de Entrada/Salida. Se resuelve con una funcionalidad de tipo AdvertisedMultiResultFunctionality, que recibe como parámetro de entrada el Facet concreto sobre el que se aplica, y produce como salidas notificaciones correspondientes con los datos de entrada.
- **Obtener selección de una lista.** Acción que indica que se requiere de una entrada por parte del usuario en un Facet Lista de Selección. Se resuelve con una funcionalidad de tipo AdvertisedMultiResultFunctionality similar a la anterior.
- **Subscribir a notificación de una acción.** Acción que permite la subscripción a un elemento Facet que proporciona notificaciones de eventos desencadenados por el usuario. Se resuelve con una funcionalidad de tipo AdvertisedMultiResultFunctionality que envía un resultado de notificación cada vez que el usuario desencadena la acción.

A través de la interacción remota con las funcionalidades del Servicio UIB correspondientes al Paradigma Abstracto de Interacción, una aplicación resuelve de forma abstracta sus operaciones de interacción con la IU, y ésta interacción es trasladada por el framework TIAF a dispositivos de interacción reales disponibles en el entorno.

Podemos observar por tanto, que al igual que ocurría en el caso del Servicio UniDA, el Servicio UIB requiere de diferentes tipos de funcionalidades, atendiendo al tipo de interacción que tiene lugar entre el cliente y el servicio. Incluyendo, en este caso, funcionalidades asíncronas durables que posibilitan una comunicación basada en eventos con múltiples respuestas.

Previamente mencionamos que el framework TIAF dispone de algoritmos capaces de seleccionar los recursos de interacción físicos más adecuados para resolver una IU abstracta en función del contexto. Para aprovechar esta capacidad, el Servicio UIB proporciona una funcionalidad que permite a las aplicaciones indicar en tiempo real información sobre el contexto de ejecución. Así, como parte del Servicio UIB se han desarrollado tres ontologías correspondientes a los modelos de contexto soportados actualmente por el framework TIAF, a su vez inspirados en el proyecto MyUI [MyUI, 2015]: contexto del entorno (e.j. ruido, espacio, visibilidad), contexto del usuario (características, preferencias y capacidades) y contexto del escenario (tipo de actividad realizada por el usuario, lugar, etc.).

El framework TIAF no especifica nada acerca de cómo las aplicaciones pueden obtener la información contextual que permitirá adaptar las IU. No obstante, aprovechando las capacidades de la arquitectura HI3 (como se ilustra en la Figura 7.14), en este trabajo proponemos que las aplicaciones utilicen el Servicio UniDA, subscribiéndose a los estados de sensores del entorno que puedan aportarles datos relevantes, para construir el Modelo de Contexto del Entorno en tiempo real. Asimismo, las aplicaciones podrán utilizar el Servicio UPDS de la arquitectura HI3 para construir el Modelo de Contexto del Usuario. Recordemos que, como se describió en el Apartado 7.3.1, el Servicio UPDS proporciona un modelo uniforme para el acceso ubicuo a información relacionada con el perfil de los usuarios.

La implementación del Servicio UIB se realiza siguiendo el mismo esquema descrito en detalle para el Servicio UniDA en la Sección 7.2.3. Así, se ha realizado una implementación del mismo en lenguaje Java junto con una implementación de las ontologías descritas en BSDL, pasando a formar parte de la Capa de Usuario en la implementación de referencia de la arquitectura HI3.

## 7.4. Resumen

Partiendo del núcleo de la arquitectura HI3 construido en base al middleware orientado a servicios descrito en el Capítulo 5, a lo largo de este capítulo se han descrito diferentes trabajos realizados con el fin de dotar a la arquitectura HI3 de nuevas capacidades para la resolución de requisitos de IAm de más alto nivel que, de forma general, hemos considerado como prioritarios para este tipo de sistemas. En concreto, nos hemos centrado en aspectos que faciliten la construcción

de entornos de IAm, dónde tanto sus componentes físicos como lógicos se adapten dinámicamente al contexto y a los usos y preferencias de los usuarios. Por tanto, persiguiendo la creación de entornos humanizados.

Las soluciones propuestas se han organizado en torno a dos elementos centrales de cualquier sistema de IAm, los usuarios y los entornos físicos en los que éstos se desenvuelven, desarrollando así la Capa de Usuario y la Capa de Dispositivos de la arquitectura conceptual HI3. En ambas capas se integraron frameworks ya existentes a la vez que se definieron e implementaron nuevos frameworks y modelos conceptuales. En un nivel superior de abstracción, en todos los casos se construyeron servicios semánticos y ontologías a través de los que se abstrae de tecnologías heterogéneas el acceso ubicuo a la funcionalidad de los frameworks y herramientas subyacentes. Así, se desarrolló un servicio que proporciona un modelo de operación uniforme de redes de dispositivos realizable sobre diferentes frameworks de IoT particulares (Servicio UniDA), un servicio que provee un modelo uniforme para el acceso y almacenamiento de información relacionada con perfiles de usuarios (servicio UPDS), y un servicio que posibilita la construcción de interfaces de usuario abstractas que son resueltas sobre dispositivos de interacción reales en tiempo de ejecución (servicio UIB).

Una aplicación que utilice conjuntamente las capacidades de los servicios UIB, UniDA y UPDS dispondrá de un grupo de herramientas poderoso para enfrentarse a dos de los grandes retos de la IAm: la adaptación al contexto y la resolución ubicua de su comportamiento. Así, los desarrolladores disponen de facilidades para construir aplicaciones capaces de trasladar su comportamiento de un entorno a otro, gracias a la posibilidad de descubrir los servicios de IAm proporcionados por la arquitectura HI3 en cada entorno, que les permitan adaptarse a diferentes contextos sin necesidad de haber sido pre-programadas de forma ad-hoc para cada situación.

Asimismo, a través de la explicación detallada del proceso de definición e implementación del Servicio UniDA se ilustró de forma pormenorizada como un desarrollador puede implementar nuevos servicios utilizando las capacidades proporcionadas por el middleware orientado a servicios de la arquitectura HI3.

# Capítulo 8

# Evaluación experimental

En este capítulo se describen dos casos concretos de sistemas de IAm en los que se han utilizando los conceptos y herramientas proporcionados por la arquitectura HI3 para su desarrollo. A través del análisis de estos casos se trata de ilustrar en mayor medida algunas de las capacidades más relevantes de la arquitectura HI3 y las posibilidades de aplicación de la misma a la construcción de sistemas de IAm en entornos reales. Todos los desarrollos experimentales realizados en esta tesis se enmarcan en el contexto de proyectos de I+D surgidos como fruto de la colaboración entre nuestro grupo de investigación y empresas del sector.

Asimismo, a lo largo del capítulo se realiza una discusión sobre el modo en como se resolverían algunas de las soluciones aquí presentadas utilizando las aproximaciones propuestas por las arquitecturas de IAm más destacadas que fueron analizadas en la sección 3.4. En concreto, nos centraremos en los proyectos Amigo, CHIL, PERSONA y SOPRANO, por ser los más ambiciosos y recientes. En el caso de los tres últimos proyectos, la comparación se ha realizado en función de toda la documentación técnica de diseño que se ha podido recuperar, ya que no existen implementaciones de referencia accesibles públicamente. En el caso de Amigo, se ha podido contar tanto con la documentación como con las fuentes de las implementaciones existentes.

## 8.1. Entornos inteligentes ONIZE

Idealmente, la IAm abarca cualquier entorno en el que las personas se desenvuelven en su día a día. Por tanto, cualquier entorno de este tipo puede poblarse de tecnologías y dispositivos que trabajen de forma no intrusiva para hacer la vida de las personas más cómoda, segura, agradable o saludable. Por ejemplo, estos en-

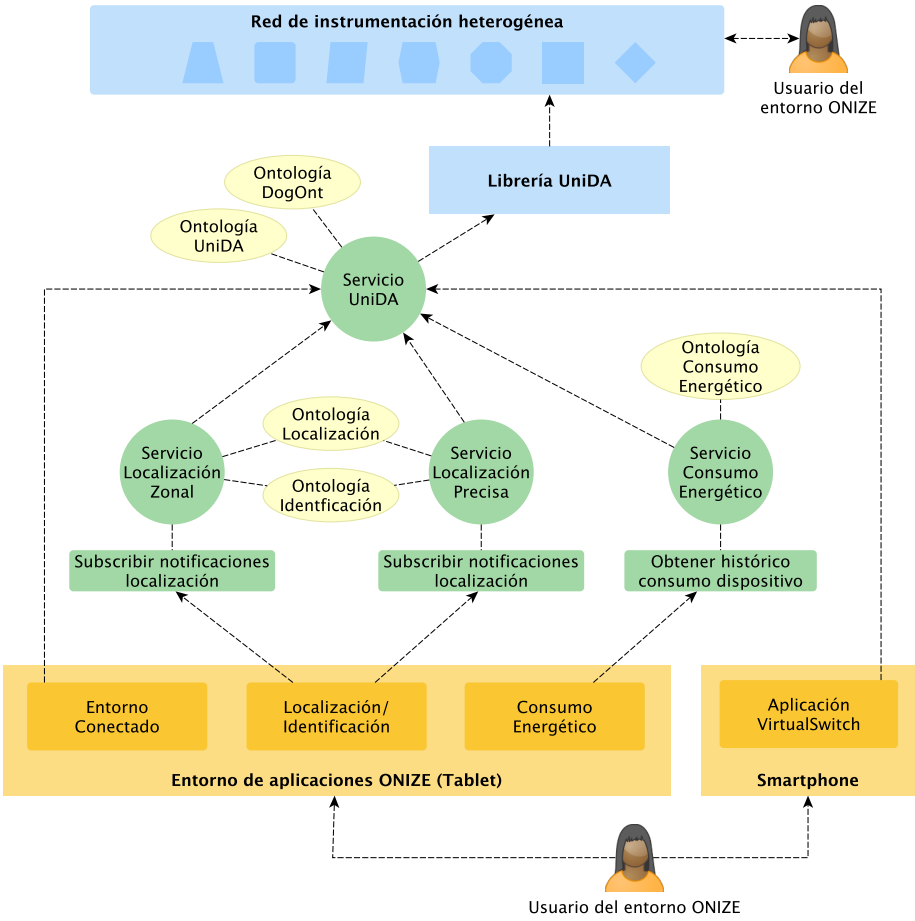



Figura 8.1: Visión lógica de los principales componentes funcionales del sistema ONIZE desarrollado.

tornos pueden monitorizar la actividad o determinados parámetros médicos de los habitantes, anticipando problemas de salud o simplemente mejorando su confort. Es de esta forma como se crean lo que en el ámbito de la IAm y la Computación Ubicua se denominan Entornos Inteligentes.

Los dispositivos de entornos inteligentes deben conectarse a través de redes y protocolos para compartir información y colaborar en la resolución de tareas. Por tanto, uno de los problemas más importantes, que pone en jaque la viabilidad de estos entornos y que está ralentizando su implantación, es la falta de compatibilidad entre diferentes protocolos de comunicación y tecnologías de dispositivos.

En esta sección presentamos un caso de aplicación de la arquitectura HI3 en el ámbito de la construcción de entornos inteligentes. En concreto, el desarrollo de este caso de aplicación está enmarcado dentro de una colaboración entre el

Grupo Integrado de Ingeniería de la Universidade da Coruña y MyTech Ingeniería Aplicada S.L. para la investigación y desarrollo de tecnologías en este ámbito. Uno de los frutos de esta colaboración ha sido la creación de una serie de tecnologías, agrupadas bajo la denominación ONIZE, enfocadas a facilitar la instrumentación de entornos de IAm gracias a la integración transparente de todo tipo de dispositivos en los mismos. En el marco de las tecnologías ONIZE, en esta tesis se ha desarrollado un conjunto homogéneo de aplicaciones y servicios, soportados por la arquitectura HI3, que facilitan la creación de un entorno inteligente, dónde los usuarios pueden controlar el entorno de forma integrada y pueden beneficiarse de nuevos servicios que surgen de la integración de dispositivos con independencia de sus tecnologías propietarias y su ubicación física. Además, para validar estos desarrollos se ha construido un entorno experimental utilizando tanto tecnologías heterogéneas representativas de elementos de instrumentación existentes como tecnologías desarrolladas como parte del propio proyecto ONIZE. Como resultado de los trabajos de investigación conducentes al desarrollo de la tecnología ONIZE se obtuvo, asimismo, una patente internacional [Paz Lopez et al., 2012].

En la Figura 8.1 se proporciona una visión lógica de los sistemas desarrollados como parte de esta tesis dentro del contexto del proyecto ONIZE. Así, partimos de una visión uniforme de los dispositivos del entorno físico proporcionada por la Capa de Acceso a Dispositivos de la arquitectura HI3, y sobre esta base se introduce el concepto de una plataforma o entorno de aplicaciones ONIZE para dispositivos móviles, que potencialmente podrá funcionar como una tienda integrada de aplicaciones destinadas al control y automatización de un entorno físico de IAm. En concreto, se desarrollaron las tres aplicaciones que pueden verse en la Figura 8.1:

- **Entorno conectado.** Esta aplicación está diseñada para descubrir un Servicio UniDA que esté ejecutándose en el entorno en el que es desplegada y utilizarlo para obtener acceso a la red de instrumentación existente con independencia de las tecnologías heterogéneas subyacentes. Proporciona una interfaz de usuario (IU) homogénea a través de la que monitorizar y controlar cualquier elemento del entorno físico. Se trata de una aplicación que está disponible por defecto en la plataforma de aplicaciones ONIZE.

- **Localización e identificación.** Es fundamental poder identificar y ubicar a las personas presentes en un entorno para adaptar los comportamientos y el propio entorno a éstos. Así, esta aplicación está orientada a la identificación y posicionamiento de los usuarios en entornos interiores como oficinas, casas, residencias de ancianos, etc. Para ello, esta aplicación descubrirá y utilizará alguno de los servicios de identificación/localización disponibles (se han desarrollado dos servicios basados en la utilización de redes inalámbricas).

- **Consumo energético.** Esta aplicación proporciona información relacionada con el consumo energético instantáneo e histórico. Utiliza una ontología de

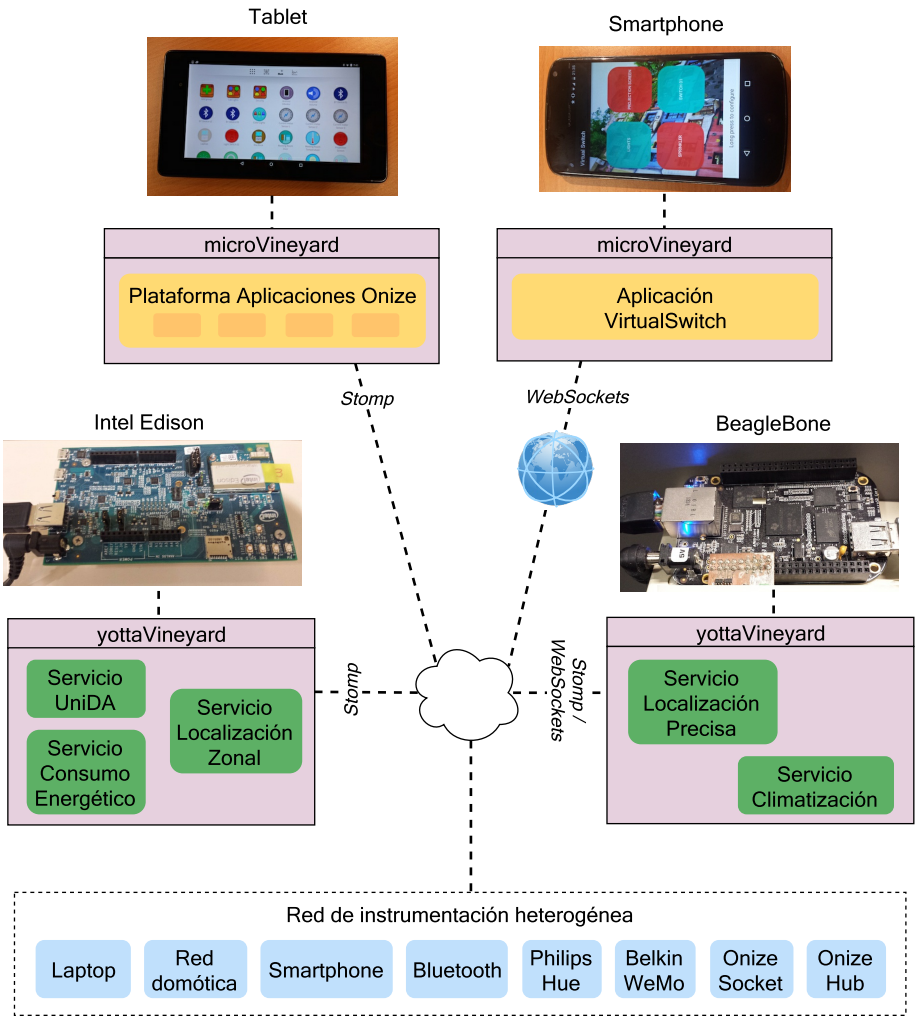



Figura 8.2: Diagrama de despliegue del sistema ONIZE desarrollado.

consumo energético para encontrar en el entorno un servicio que le proporcione este tipo de información. Se ha desarrollado, asimismo, un servicio de consumo energético que colabora con el Servicio UniDA para descubrir los enchufes del entorno capaces de proporcionar datos de consumo.

Además, se ha desarrollado una aplicación para dispositivos de tipo *smartphone* con sistema operativo Android, denominada VirtualSwitch, que permite la asignación rápida y dinámica de botones virtuales a acciones de tipo binario sobre dispositivos individuales o sobre agrupaciones de dispositivos.

En la Figura 8.1 podemos observar las relaciones entre estas aplicaciones y los

servicios desarrollados, así como ejemplos de algunas de las funcionalidades que proporcionan dichos servicios y las ontologías relacionadas.

Para el despliegue de este sistema en el entorno experimental construido se ha elegido una estructura como la de la Figura 8.2. En este caso, para ilustrar mejor las capacidades de la arquitectura HI3 se han utilizado dos contenedores completos yottaVineyard, en los que se ejecutan los diferentes servicios del sistema. En una instalación no experimental podría utilizarse un único contenedor yottaVineyard. En nuestro caso, se han desplegado los contenedores en dos computadores SBC (del inglés, Single-Board Computer) diferentes (en concreto un Intel Edison [IntelEdison, 2015] y un BeagleBone Black [BeagleBoneBlack, 2015]). En ambos casos se trata de dos SBC de muy reducido tamaño, bajo consumo y sin necesidad de refrigeración activa, adecuados para la ejecución ininterrumpida de instancias de la arquitectura HI3 y para la integración no intrusiva de los mismos en prácticamente cualquier entorno. Ambos contenedores se encuentran federados, de modo que a través del registro de uno se pueden descubrir también los servicios del otro. Para esta instalación, de entre los protocolos disponibles en la implementación de referencia se ha seleccionado STOMP, de modo que la federación se configura únicamente indicando la URL correspondiente al otro contenedor (stomp://ip:puerto).

Por otro lado, en una tableta Android se ha desplegado un contenedor microVineyard y la plataforma de aplicaciones ONIZE, mientras que en un *smartphone* Android se ha desplegado otro contenedor microVineyard y la aplicación VirtualSwitch. Ambos contenedores necesitan conectarse a uno de los contenedores yottaVineyard para poder acceder a la funcionalidad de un registro de servicios. En el desarrollo experimental realizado se le ha atribuido un ámbito de ejecución local a la plataforma de aplicaciones que se ejecuta en la tableta Android. Así por ejemplo, estas aplicaciones podrían utilizarse en la red local de una vivienda o un edificio para el control y monitorización de dicho entorno. Por otro lado, se ha determinado que la aplicación VirtualSwitch podrá acceder al entorno también a través de Internet utilizando una conexión de red móvil. Por tanto, en el último caso es necesario utilizar un grounding de tipo WebSockets que posibilite este tipo de comunicación. Con este escenario se ilustran las capacidades de la arquitectura para la convivencia de múltiples protocolos de forma transparente. Así, a pesar de que ambos contenedores yottaVineyard están federados utilizando el protocolo STOMP y el contenedor del *smartphone* se conecta sobre WebSockets, un componente de este último podrá realizar de forma transparente búsquedas a través de su registro que incluyan como resultado servicios de cualquiera de los otros contenedores.

Discutiremos en este punto como se podría desplegar este mismo sistema utilizando algunas de las arquitecturas de IAm existentes. La primera dificultad con la que nos encontraríamos en todos los casos está relacionada con el despliegue de componentes de la arquitectura en una variedad de entornos computacionales, incluyendo dispositivos móviles. En este sentido, tanto PERSONA como SOPRANO

dependen de la plataforma de servicios OSGi [OSGi Alliance, 2015] para la ejecución, gestión y descubrimiento de sus componentes. En el caso de CHIL, la dependencia es de la plataforma de agentes Jade [JADE, 2015]. Tanto OSGi como Jade están a su vez ligados a su ejecución en una Máquina Virtual Java, con lo que se limita su portabilidad a otros entornos. Por el contrario, la solución menos restrictiva es Amigo, puesto que, al igual que la arquitectura HI3, no está estructuralmente ligado a una plataforma concreta.

La decisión de no imponer una estructura compleja y rígida para la arquitectura, permitiendo al desarrollador elegir con qué elementos de la misma resuelve sus funcionalidades, junto con un nivel superior de abstracción de todas las capas conceptuales en base a dos únicos componentes (servicios y aplicaciones), proporciona una clara ventaja a Amigo y HI3 de cara a la generalidad de la solución y a su implantación en una gran diversidad de escenarios. En estos casos, incluso si un desarrollador no dispone de una implementación de la arquitectura para la plataforma computacional que desea utilizar, dispone de opciones más sencillas que realizar una nueva implementación completa de la arquitectura: i) decidir implementar únicamente el modelo y el lenguaje de descripción de servicios para construir un servicio o aplicación interoperable; ii) construir un servicio o aplicación interoperable utilizando otras tecnologías soportadas por la arquitectura (e.j., OWL-S, OWL y algún protocolo de comunicación soportado) si están disponibles. Sin embargo, en el caso del sistema ONIZE, si quisiéramos portar el modelo de servicios de Amigo a Android nos encontraríamos con numerosas dificultades debido al alto número de dependencias de éste con otras tecnologías no portables u obsoletas. Obviando este obstáculo, sería posible desplegar un sistema ONIZE con la arquitectura Amigo similar al realizado con la arquitectura HI3, pero utilizando por ejemplo el protocolo SOAP/HTTP para una comunicación de tipo RPC entre servicios y aplicaciones.

En el caso de PERSONA, sería posible desplegar los servicios del sistema como servicios OSGi descritos según el modelo OWL-S y con capacidad de comunicarse entre ellos a través de un modelo de invocación RPC. Las aplicaciones ONIZE tendrían que implementarse al margen del middleware PERSONA y podrían utilizar la URL única que proporciona PERSONA para posibilitar un acceso externo y limitado a su plataforma OSGi. En el caso de SOPRANO, nos encontramos con la misma dependencia de OSGi, pero agravada con la falta absoluta de soporte para la interacción con sistemas externos. Por último, en la arquitectura CHIL, los servicios del sistema ONIZE serían implementados como agentes dentro de instancias de la plataforma Jade, incluyendo descripciones de sus capacidades en lenguaje OWL (aunque CHIL no utiliza ningún modelo de descripción semántica de servicios propiamente dicho). De nuevo, las aplicaciones ONIZE para dispositivos móviles deberían implementarse al margen de la plataforma, siendo factible implementar una comunicación de tipo RPC sobre SOAP para acceder a dichos servicios.

Volviendo a la implementación del caso de ejemplo ONIZE utilizando la arqui-

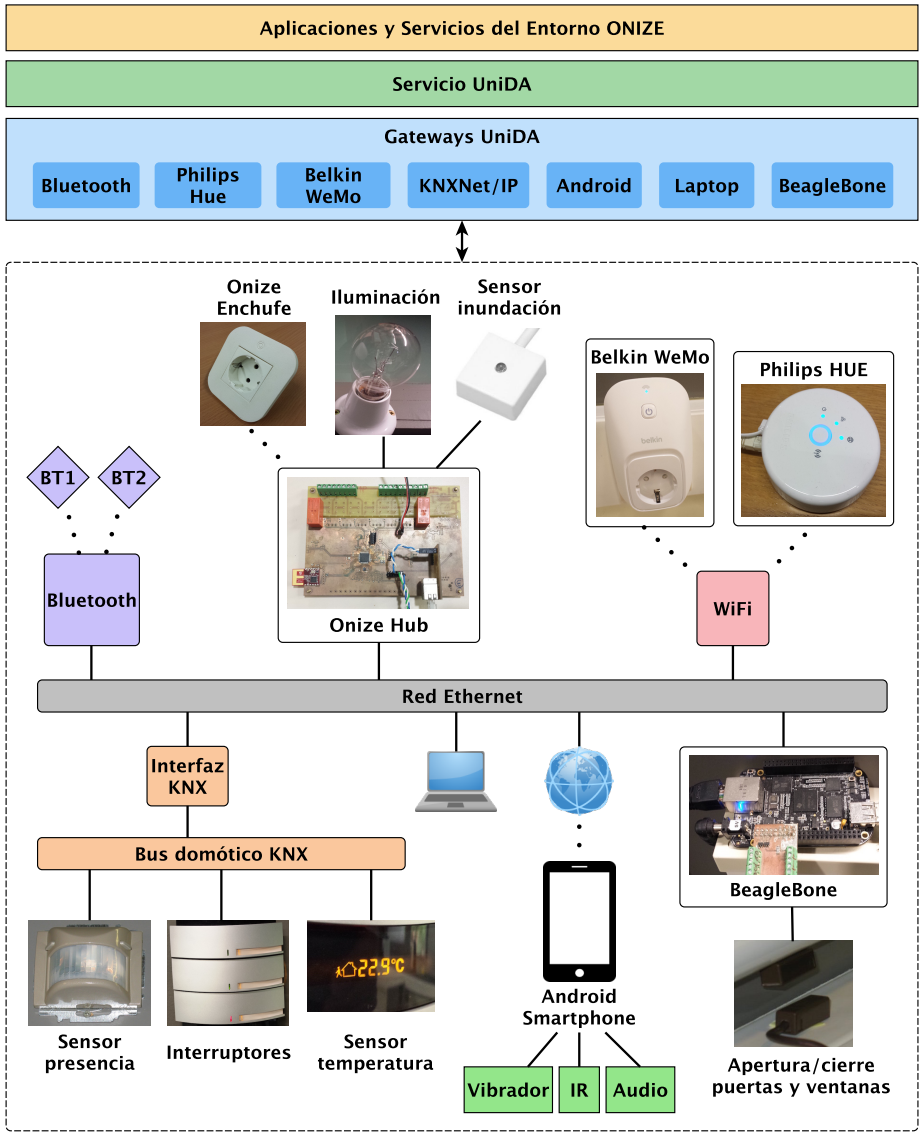



Figura 8.3: Entorno físico experimental heterogéneo.

tectura HI3, en lo referente al entorno físico experimental utilizado para validar el desarrollo, se ha tratado de dotar al mismo de una amplia variedad de tecnologías que son susceptibles de ser implantadas en diferentes tipos de entorno, como puede ser una vivienda, un entorno de trabajo, un edificio público, etc. El entorno concreto utilizado se corresponde con un área de un laboratorio de investigación de la Universidade da Coruña que comprende tres salas. En la Figura 8.3 se ilustra en parte la variedad de tecnologías de redes de comunicación y dispositivos presentes

en el entorno experimental. Dicho entorno estaba previamente equipado con una red Ethernet, una red WiFi, una red domótica KNX, un sistema Philips HUE (permite el control inalámbrico de elementos de iluminación a través de una red ZigBee) y una serie de dispositivos ONIZE basados en tecnologías de microcontrolador que son directamente compatibles con el protocolo de la Librería UniDA. Este entorno se extendió para los casos experimentales de esta tesis con otros elementos, como enchufes Belkin WeMo controlables inalámbricamente, antenas Bluetooth utilizadas para localización en interiores o una placa BeagleBone Black utilizada para controlar sensores/actuadores digitales (e.j. sensores de apertura de puertas y ventanas).

Para que todos estos dispositivos estén accesibles a través de la Librería UniDA ha sido necesario implementar nuevos Gateways UniDA en los casos de aquellas tecnologías para los que no se disponía de uno. Recordemos que estos gateways traducen los conceptos manejados por otras tecnologías a los conceptos del modelo de dispositivos UniDA [Varela et al., 2011]. Así, por ejemplo, para ilustrar que se pueden tratar de forma homogénea dispositivos que a priori pertenecen a ámbitos muy distintos, se ha desarrollado un gateway para ordenadores portátiles que proporciona un dispositivo Laptop con cuatro estados: estado de la batería, nivel de la batería, volumen y estado encendido-apagado. De esta forma, una aplicación que utilice el Servicio UniDA podría acceder remotamente a modificar el volumen de un portátil exactamente de la misma forma (en un lenguaje neutral y utilizando la misma funcionalidad del servicio UniDA) que si necesitase, por ejemplo, modificar la intensidad de una luz regulable controlada por una red domótica. No obstante, para que todo esto sea posible, también hemos tenido que extender la taxonomía de dispositivos proporcionada por la ontología DogOnt [Bonino and Corno, 2008] con definiciones para todos los dispositivos que no existían en la misma.

Una vez definidos y desplegados todos los elementos del entorno experimental, las aplicaciones y servicios del sistema ONIZE que necesiten acceder a alguno de los dispositivos del entorno físico lo harán interaccionando con el Servicio UniDA que se ejecuta en uno de los contenedores yottaVineyard.

En los siguientes subapartados se describe el detalle de las aplicaciones más relevantes que se han desarrollado en esta tesis como parte de la plataforma ONIZE para entornos inteligentes.

### 8.1.1. Un entorno conectado

La aplicación Entorno-Conectado proporciona una interfaz homogénea para la monitorización y control de un entorno inteligente ONIZE. Para ello, esta aplicación al iniciarse tratará de encontrar, a través del registro de servicios de su contenedor, un Servicio UniDA que le permita descubrir los dispositivos existentes en el entorno y que le proporcione las funcionalidades que necesita para acceder a los mismos. Por

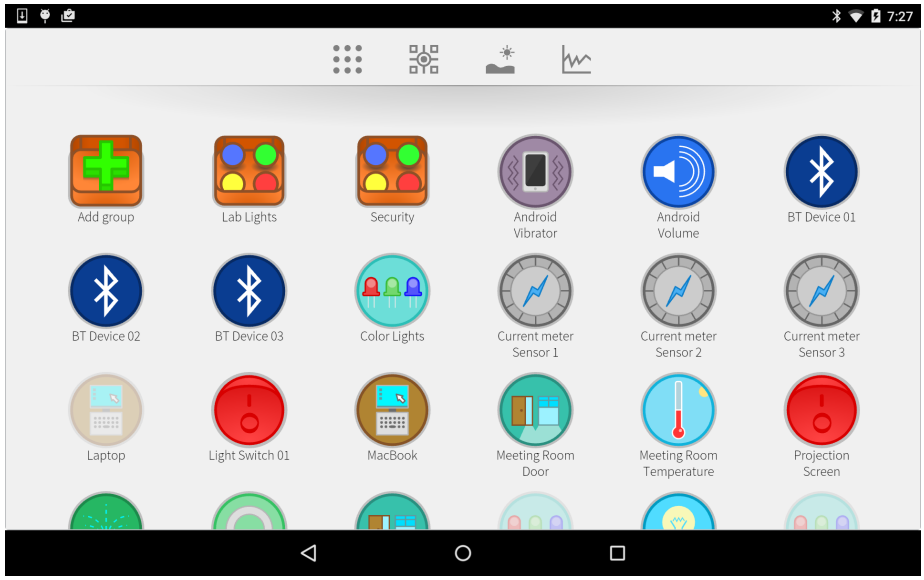



Figura 8.4: Captura de pantalla de la aplicación Entorno-Conectado en la que se proporciona una vista de los dispositivos descubiertos en el entorno.

tanto, esta aplicación se ha desarrollado con independencia del entorno concreto en el que vaya a ser desplegada y ejecutada.

En la Figura 8.4 se muestra una captura de pantalla de la plataforma de aplicaciones ONIZE desarrollada, que está ejecutándose en una tableta Android. La IU de la plataforma se ha diseñado de forma que con un deslizamiento en horizontal se pasa a la siguiente/anterior aplicación disponible en la plataforma. En esta captura de pantalla se encuentra activa la aplicación Entorno-Conectado. Se puede ver una lista de dispositivos descubiertos, presentando en tono apagado aquellos que estuvieron accesibles en el pasado en este entorno y no lo están actualmente (por ejemplo porque están apagados/desconectados). Así, en la imagen podemos ver dispositivos pertenecientes a la red domótica KNX (e.j., Meeting Room Temperature, Light Switch 01, Projection Screen), dispositivos exportados por un smartphone Android (Android Vibrator, Android Volume), sensores Bluetooth, etc. Además, la aplicación incluye la posibilidad de organizar los dispositivos en grupos, de modo que se facilite la gestión de un entorno por parte del usuario.

En la Figura 8.5 se incluyen dos capturas de pantalla en las que se ve la información detallada que muestra la aplicación para el caso de dos dispositivos distintos. La información más interesante es la que tiene que ver con los tipos de estados del dispositivo y los valores actuales de los mismos. La meta-información del dispositivo se corresponde con su descripción en la ontología DogOnt, mientras que los valores de los estados los obtiene en tiempo real la aplicación a través de la funcionali-

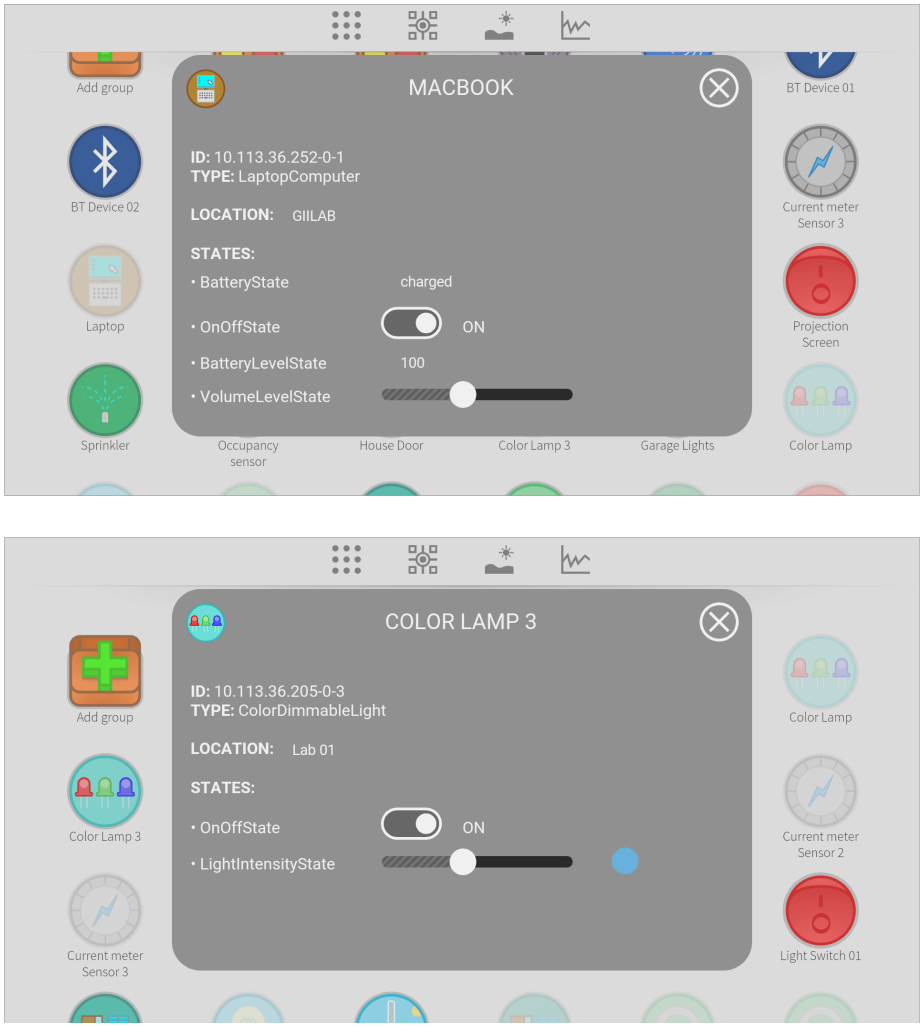



Figura 8.5: Capturas de pantalla de la aplicación Entorno-Conectado en la que se muestra la información detallada en tiempo real de dos dispositivos del entorno, a la vez que se permite actuar sobre sus estados modificables.

dad correspondiente del Servicio UniDA. Además, podemos ver como los estados que son modificables pueden actuarse directamente desde la aplicación (LightIntensityState en Color Lamp 3, por ejemplo). Utilizando el paradigma uniforme para la operación de dispositivos del Servicio UniDA, la aplicación solicita la actuación sobre el estado de un dispositivo con independencia de tecnologías de dispositivos y redes de comunicación subyacentes.

Con esta aplicación los usuarios pueden tener una visión homogénea y en tiempo real del estado de todos los dispositivos del entorno, al mismo tiempo que pue-

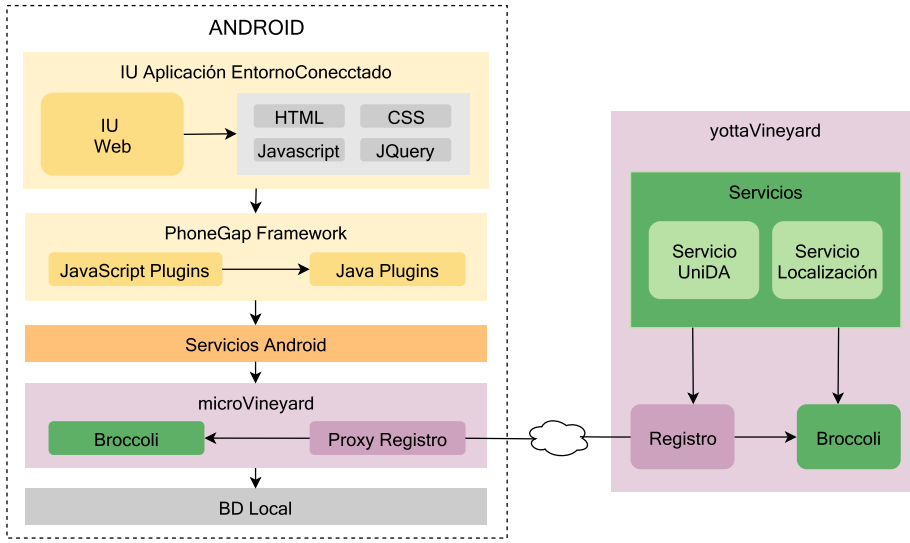



Figura 8.6: Estructura de una aplicación Android de la plataforma ONIZE.

den actuar sobre ellos de forma remota. Además, dado que todos los dispositivos, sus estados y funcionalidades tienen una misma representación lógica, es posible establecer fácilmente reglas de comportamiento que relacionen dispositivos heterogéneos entre sí. Por ejemplo, un usuario podría desear que cuando cambie el estado de apertura de la puerta de entrada (OpenCloseState) se le avise automáticamente en el smartphone Android, actuando para ello sobre el estado del dispositivo Android Vibrator (OnOffState).

No se entra en detalle de como la aplicación Entorno-Conectado utiliza el Servicio UniDA gracias a las capacidades de la arquitectura HI3, dado que ya fue explicado en el Capítulo 7. Sin embargo sí explicaremos cuál es la estructura software que se ha definido para el desarrollo de aplicaciones dentro de la plataforma ONIZE, utilizando como ejemplo la aplicación Entorno-Conectado (ver Figura 8.7). Aunque todas las aplicaciones presentadas en este caso experimental han sido desarrolladas sobre la plataforma Android, se ha propuesto una arquitectura para las mismas basada en el Framework PhoneGap [PhoneGap, 2015], que facilita la migración de aplicaciones entre diferentes plataformas móviles (Android, iOS, Windows Phone, etc.) gracias a la utilización de tecnologías Web estándar para el desarrollo de la IU. Así, cada aplicación del entorno ONIZE construirá su IU en base a este framework, mientras que su lógica de control se comunicará con las herramientas proporcionadas por el contenedor microVineyard. En la implementación de referencia del contenedor microVineyard para el sistema Android, es un componente Servicio de la API Android el encargado de ejecutar automáticamente el contenedor en el arranque de la tableta o smartphone. A través de este contenedor microVineyard (único

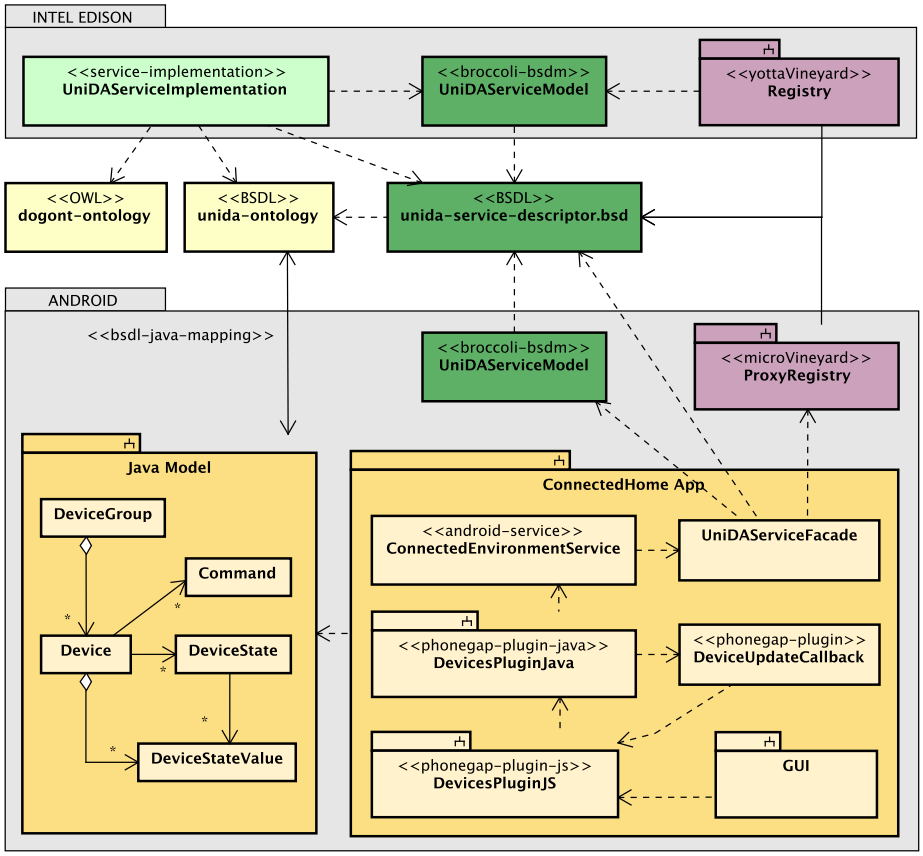



Figura 8.7: Relación entre la aplicación Entorno-Conectado y el Servicio UniDA.

para todas las aplicaciones) la lógica de control de la aplicación Entorno-Conectado puede acceder al registro de servicios para descubrir servicios que necesita (en este caso el Servicio UniDA).

En la Figura 8.7 se muestra un mayor detalle de la arquitectura software de la aplicación Entorno-Conectado, prestando especial atención a los elementos que permiten su interacción con el Servicio UniDA utilizando las capacidades de la arquitectura HI3. Podemos observar como la aplicación proporciona su propia implementación del modelo uniforme de dispositivos descrito por la ontología UniDA, de acuerdo con las características que necesita. Utiliza las capacidades del modelo de servicios BSDM para mapear automáticamente la información necesaria para la interacción con el Servicio UniDA, descrita en el lenguaje neutral BSDL, a objetos de su modelo de datos local.

Si analizamos las capacidades de otras arquitecturas de IAm para abordar este mismo escenario, encontraremos diferencias significativas entre ellas. Amigo, por

ejemplo, no proporciona un modelo universal de acceso a dispositivos. Siguiendo su filosofía, sería necesario implementar un servicio distinto para abstraer el acceso a cada uno de los tipos de tecnologías disponibles en el entorno físico. Por lo tanto, a pesar de que una aplicación ONIZE podría descubrir servicios semánticos e interoperables de este tipo para controlar y adaptarse al entorno, tendría que lidiar con funcionalidades heterogéneas para interaccionar con cada tipo de dispositivo.

En el caso de la arquitectura CHIL, cada tipo de tecnología de dispositivos es abstraída a través de una API concreta. A mayores, los dispositivos tienen una representación en forma de un *agente proxy* dentro de la plataforma de agentes Jade. Estos agentes podrían describir las capacidades del dispositivo al que representan utilizando alguna ontología OWL, de forma similar a lo realizado en la arquitectura HI3 (aunque CHIL no proporciona ningún modelo concreto para la operación uniforme de dispositivos). A partir de ese momento sería posible descubrir los *proxys* de dispositivos existentes a través del registro DF de Jade. Esta aproximación hace a la solución excesivamente dependiente de la plataforma Jade y de su registro DF, que además penaliza el rendimiento con un elevado número de agentes.

La arquitectura PERSONA incluye una ontología OWL concreta para la descripción uniforme de dispositivos. Además, cada dispositivo sería abstraído a través de un componente uniforme capaz de publicar eventos de cambio en el estado de un sensor en el bus de interacción de la arquitectura, posibilitando que otros componentes se subscriban a los mismos. Para los dispositivos actuadores, este componente recibe invocaciones de cambios de estado a través de un bus de comunicación RPC de la arquitectura. Esta solución desacoplaría en gran medida a las aplicaciones ONIZE de las particularidades del entorno físico, pero para ello obligaría a la implementación de dichas aplicaciones como componentes OSGi dependientes de su middleware de comunicación basado en distintos tipos de buses.

En el caso de SOPRANO, los dispositivos del entorno estarían representados en base a sus estados de acuerdo con la ontología del contexto en torno a la que se organiza toda la arquitectura. Además, cada dispositivo podría tener su correspondencia en un servicio OSGi descrito semánticamente con el modelo DIANE (modelo basado en OWL-S con posibilidad de manejar información difusa). Así, ante un cambio de estado en un sensor se actualizaría el estado en las instancias de la ontología del contexto. A pesar de los aspectos interesantes de la utilización de información difusa en el modelo DIANE, la solución de acceso al estado del entorno físico está demasiado ligada a la ontología de representación del contexto definida para entornos de personas dependientes, y de nuevo sin posibilidades de interoperabilidad y descubrimiento de servicios fuera de la plataforma OSGi de SOPRANO.

A diferencia de los tres casos anteriores, en la arquitectura HI3 el Servicio UniDA proporciona un acceso uniforme y ubicuo al entorno físico independiente de otros componentes de la arquitectura y que facilita la interoperabilidad con otros

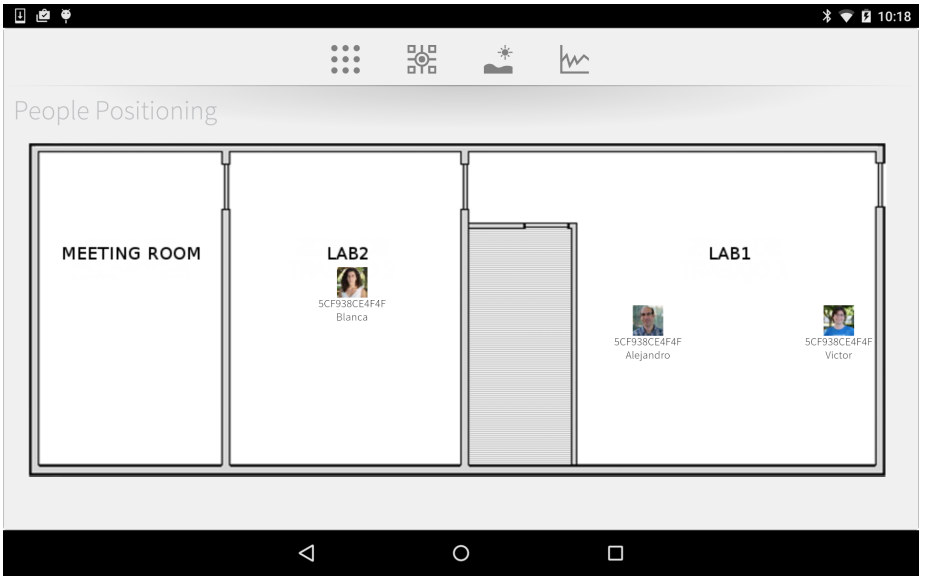



Figura 8.8: Captura de pantalla de la aplicación de localización/identificación.

sistemas. Además, en este ejemplo de aplicación, tenemos un caso en el que se utiliza en un mismo servicio una ontología descrita en lenguaje OWL y una ontología en lenguaje BSDL. Este tipo de interoperabilidad con más de un lenguaje de ontologías o más de un modelo de servicios únicamente está también presente en la arquitectura Amigo.

### 8.1.2. Localización e identificación de personas

La localización e identificación de las personas presentes en un entorno es un pre-requisito fundamental para crear entornos inteligentes. Un sistema sin conciencia de las personas presentes en el entorno será incapaz de adaptar convenientemente su comportamiento al contexto. Además, de acuerdo con los principios de la IAm, la localización y la identificación deberá realizarse de forma que no resulte intrusiva para el usuario. Por tanto, este tipo de soluciones deberían formar parte de cualquier arquitectura de IAm y, siguiendo los principios de la arquitectura HI3, deberían proporcionar su funcionalidad a través de servicios que idealmente referencien a ontologías estándar de identificación y localización.

Existen numerosas aproximaciones al problema de identificar y localizar personas tanto en entornos interiores como exteriores. Algunas de las aproximaciones más interesantes para entornos de IAm están basadas en tecnologías inalámbricas, GPS o dispositivos *wearables*. Como parte de este trabajo, se ha propuesto una sencilla ontología de identificación y posicionamiento y se han desarrollado dos servicios

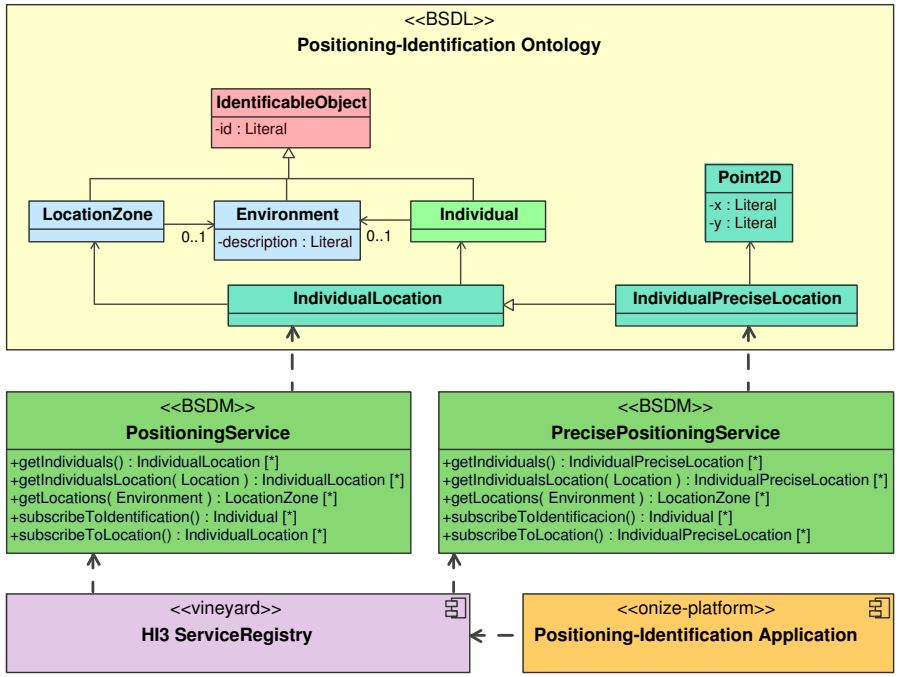



Figura 8.9: Ontología de localización e identificación utilizada en ONIZE.

para localización en interiores basados en la tecnología Bluetooth. No obstante, la resolución final del proceso de localización podría sustituirse por otros algoritmos y sistemas existentes basados en la utilización de redes inalámbricas convencionales o incluso combinar varias tecnologías para alcanzar mejores resultados.

En la Figura 8.8 se puede ver una captura de pantalla de la IU correspondiente a la aplicación desarrollada para la localización e identificación de usuarios. Esta aplicación tiene la misma estructura software explicada en el caso de la aplicación Entorno-Conectado. Por tanto, no constituye una aplicación Android individual sino que es un módulo de la plataforma ONIZE. La aplicación se configura con el mapa y las coordenadas de las estancias del entorno para poder proporcionar una visualización gráfica de la localización de las personas. Asimismo, la aplicación no está vinculada a ningún servicio, a ningún entorno físico ni a ninguna tecnología en particular. Utiliza cualquier servicio que en un momento determinado sea descubrible a través del registro de su contenedor y que proporcione funcionalidades a través de las que obtener información de posicionamiento y/o identificación. El único conocimiento prefijado que tiene es la ontología de identificación y localización.

En la Figura 8.9 se muestran los conceptos y relaciones de la ontología de localización e identificación desarrollada. Gira en torno a dos conceptos principales:

Listado 8.1: Dos objetivos de búsqueda de funcionalidades de localización/identificación descritos en lenguaje BSDL.

```
<import prefix="bsdmProfile" to="http://hi3project.com/broccoli/bsdm/profile"/>
<import prefix="ontology" to="http://hi3project.com/positioning/ontology"/>

<!-- Primer objetivo de búsqueda de funcionalidades -->

<instance of="bsdmProfile#requestedFunctionality"
   URI="http://hi3project.com/positioning/client#requestForIndividualsInLocation">

   <with property="output">
      <with property="name" value="Individuos identificados y localizados"/>
      <with property="type" valueURI="ontology#IndividualPreciseLocation"/>
   </with>
</instance>

<!-- Segundo objetivo de búsqueda de funcionalidades -->

<instance of="bsdmProfile#requestedFunctionality"
   URI="http://hi3project.com/positioning/client#requestForIndividualsInLocation">

   <with property="output">
      <with property="name" value="Individuos identificados y localizados"/>
      <with property="type" valueURI="ontology#IndividualInLocation"/>
   </with>
   <with property="preferred">
      <with property="output">
         <with property="name" value="Localización precisa preferiblemente"/>
         <with property="type" valueURI="ontology#IndividualInPreciseLocation"/>
      </with>
   </with>
</instance>
```

individuos identificables y localizaciones de individuos en el entorno. En el caso de las localizaciones, se distingue entre aquellas que únicamente especifican un zona discreta (una estancia por ejemplo) y aquellas que especifican una localización precisa (incluye coordenadas de posicionamiento en una zona). Además, tanto las localizaciones como los individuos pueden ir calificados con una identificación del entorno con el que están relacionados. Esto último facilita a las aplicaciones la gestión de múltiples entornos y el filtrado de información por entorno. La ontología ha sido descrita en lenguaje BSDL y es utilizada por los dos servicios de identificación-posicionamiento desarrollados: PrecisePositioningService y PositioningService. En la Figura 8.9 podemos ver las principales funcionalidades de estos servicios. Ambos permiten consultar información tanto de identificación como de posicionamiento de individuos, con la diferencia de que el servicio PrecisePositioningService es capaz de proporcionar una localización precisa, con coordenadas, y el servicio PositioningService únicamente proporciona una localización a nivel de zona.

Veremos más adelante como se han implementado estos servicios. No obstante, con independencia de las tecnologías concretas que estos servicios utilicen para

resolver la identificación y la localización, nos van a servir para ilustrar algunas de las características de la arquitectura HI3 que permiten desarrollar aplicaciones capaces de adaptarse a su contexto de ejecución sin necesidad de estar programadas ad-hoc para un escenario concreto. Así, se han realizado varios experimentos con diferentes configuraciones del entorno experimental, partiendo de una distribución de componentes como la que describimos al comienzo de esta sección (Figura 8.2), en la que los servicios de localización/identificación se despliegan en dos contenedores yottaVineyard.

- En primer lugar se configuró la aplicación de posicionamiento/identificación para que al iniciarse intentase buscar, a través de su registro de servicios, funcionalidades que proporcionen localizaciones precisas de individuos. En el Listado 8.1 se puede ver el primer objetivo de búsqueda expresado en BSDL utilizado por la aplicación en este caso. Dado que el objetivo son funcionalidades con una salida correspondiente al concepto IndividualPreciseLocation, el *matchmaker* del registro encuentra únicamente como funcionalidades válidas las del servicio PositioningPreciseService. Además, la aplicación puede elegir si prefiere una funcionalidad de *petición/respuesta* o una funcionalidad de *subscripción*.

- En un segundo caso, se configuró la aplicación de modo que en su inicio especifique el segundo objetivo de búsqueda del Listado 8.1. Este objetivo especifica que a la aplicación le sirve cualquier funcionalidad que proporcione salidas de tipo IndividualLocation, pero que tiene preferencia por aquellas que proporcionen salidas IndividualPreciseLocation. En este caso el *matchmaker* da como resultado funcionalidades de ambos servicios, ya que cualquier concepto que especialice a IndividualLocation es válido como salida. No obstante, el *ranker* puntúa mejor las funcionalidades del servicio PositioningPreciseService debido a la preferencia especificada en el objetivo. Esta es la solución por defecto implementada para la aplicación, en la que ésta es capaz de trabajar con cualquiera de los dos tipos de servicio que encuentre en el entorno, pero siempre que pueda proporcionará una localización precisa.

- En un tercer caso se probó a eliminar del entorno el servicio de localización precisa, para comprobar como con el mismo objetivo de búsqueda del caso anterior, la aplicación puede encontrar y adaptarse a trabajar con el servicio de localización no precisa.

En lo referente a la implementación concreta de un servicio de localización-identificación no intrusiva para interiores, se decidió utilizar un conjunto de balizas Bluetooth repartidas por el entorno, de modo que utilizando la información de los dispositivos Bluetooth detectados por cada baliza se hace una estimación de la posición de cada dispositivo. De esta forma, basta con que cada persona lleve encima algún tipo de dispositivo Bluetooth (un smartphone, una pulsera, etc.) del

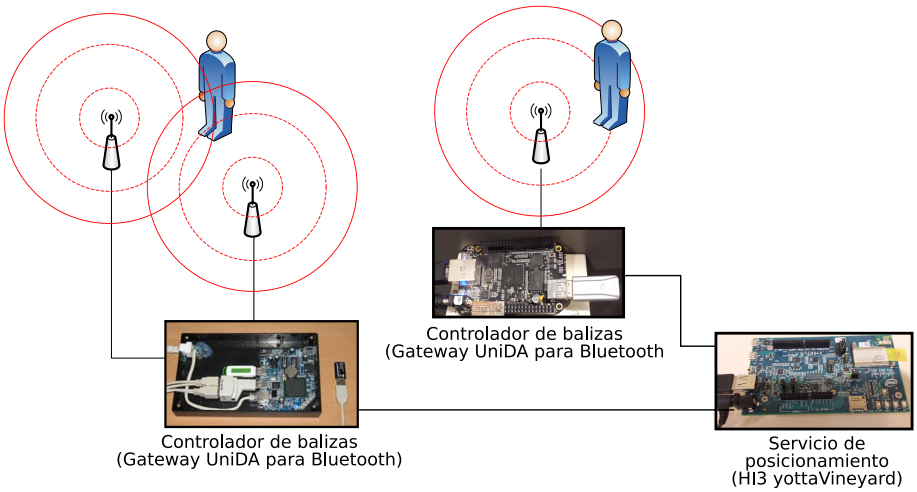



Figura 8.10: Sistema de localización/identificación utilizando balizas Bluetooth.

que se conoce su identificador (la dirección MAC) para que la persona pueda ser localizada e identificada.

El cálculo de la posición se hace realizando una triangulación basada en la utilización de información de la localización de las balizas, del número de balizas presentes en cada zona y de las intensidades de señal con que distintas balizas detectan a los dispositivos. Para el acceso a las distintas antenas Bluetooth instaladas en el entorno, el servicio de localización/identificación interacciona con el Servicio UniDA. Para ello, se ha desarrollado un Gateway UniDA para antenas Bluetooth que exporta cada antena como un dispositivo UniDA que tiene un estado que se corresponde con los identificadores (direcciones MAC) de aquellos dispositivos Bluetooth que se encuentran al alcance de dicha antena en cada momento. Además, este gateway pone a las antenas Bluetooth en un modo periódico de escaneo de dispositivos (inquiry scan). El servicio de localización, solicitará una funcionalidad de un Servicio UniDA disponible en el entorno para obtener la información de todos los dispositivos pertenecientes a la clase BluetoothMACSensor (clase de dispositivos que hemos definido en la ontología DogOnt para especificar una baliza Bluetooth). A partir de ahí podrá subscribirse a eventos de cambio de estado de esos dispositivos y utilizar la información de las balizas para generar información de localización e identificación cuando se lo solicite una aplicación u otro servicio.

En la Figura 8.10 se ilustra la organización del sistema de localización basado en Bluetooth desarrollado. A partir de él se desarrollaron los dos servicios previamente descritos, uno que trata de realizar una triangulación más precisa utilizando intensidades de señal y otro que únicamente proporciona localización con una granularidad de zona (sala o estancia).

El sistema de localización e identificación así construido nos permite evaluar algunas de las capacidades de la arquitectura HI3 en comparación con otras arquitecturas existentes. En este caso, obviaremos problemáticas ya tratadas previamente, como la relacionada con el despliegue de la solución en dispositivos móviles.

En este caso de aplicación nos encontramos con servicios que requieren de funcionalidades que se resuelven de forma natural con un modo de interacción de tipo publicación/subscripción, mientras que la arquitectura Amigo soporta únicamente modelos de invocación de tipo RPC. Esto obligaría, por ejemplo, a que fuesen los clientes de una funcionalidad de identificación los que preguntasen de forma activa para saber si se ha producido alguna nueva identificación de un usuario en vez de recibir eventos asíncronos cada vez que se produzca una. En el caso de PERSONA sí podrían gestionarse este tipo de interacciones, registrando el componente correspondiente en un bus de interacción basado en eventos, sin embargo perdemos la interoperabilidad con otros sistemas. En el caso de SOPRANO la interacción con los servicios sigue un modelo de invocación petición/respuesta, pero sería posible implementar un servicio de localización que modificase la ontología del contexto para reflejar un evento de identificación o localización. De esta forma, la aplicación podría a su vez ser notificada de estos cambios en el contexto. Sin embargo esta solución es poco escalable y poco desacoplada al depender de instancias de la ontología del contexto para representar todo el estado del sistema. En el caso de CHIL, no se soporta un verdadero modelo de descripción de servicios, por lo que la interacción podría resolverse con comunicaciones ah-hoc entre los agentes que representen a los servicios y aplicaciones involucrados.

Por último, otro aspecto diferencial de la solución de la arquitectura HI3 frente a los mecanismos de descripción de funcionalidades de las arquitecturas evaluadas es la posibilidad de indicar de forma sencilla objetivos de búsqueda de funcionalidades con valores *opcionales* o *preferidos* cuando se utiliza el modelo BSDM.

### 8.1.3. Monitorización del consumo energético

Los aspectos relacionados con la monitorización y control del consumo energético constituyen otro foco de interés en prácticamente cualquier entorno inteligente que consideremos. Así, como parte de la plataforma ONIZE, se ha tratado de sacar provecho de dispositivos existentes en este ámbito para incorporar a la plataforma un sistema de monitorización del consumo energético no intrusivo. En concreto se ha utilizado un enchufe, compatible con el zócalo de pared estándar, que fue desarrollado como parte de las tecnologías hardware ONIZE y que posibilita su control remoto para activar o cortar la corriente del mismo y para obtener datos en tiempo real de consumo. Este dispositivo, como todo el hardware ONIZE es compatible con el protocolo UniDA. Por otro lado se utilizó un enchufe externo Belkin WeMo con

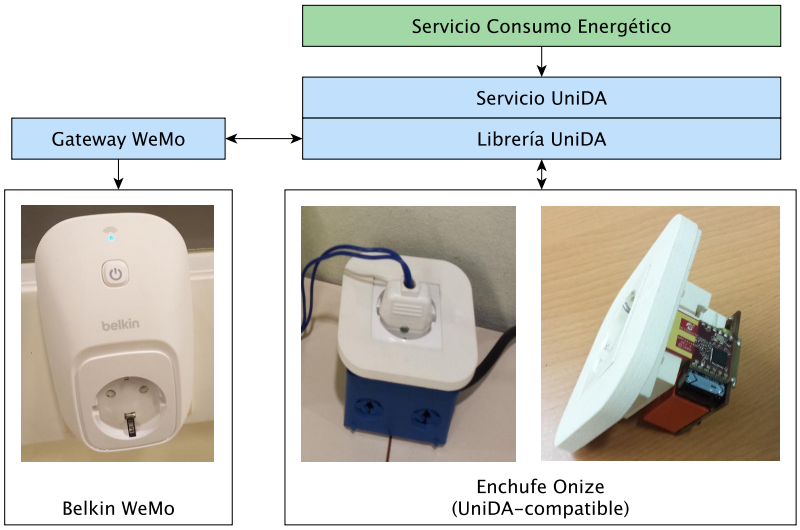



Figura 8.11: Enchufes eléctricos con posibilidad de control remoto utilizados.

capacidad para ser controlado remotamente, pero que no proporciona datos de consumo. En la Figura 8.11 se puede ver ambos dispositivos, así como los componentes software que proporcionan acceso a los mismos. Así, integrando el dispositivo de Belkin a través de un Gateway UniDA, se consiguió un acceso uniforme a ambos dispositivos a través del Servicio UniDA.

El control (activación/desactivación) de estos enchufes puede realizarse a través de la aplicación Entorno-Conectado, como ocurre con cualquier otro dispositivo abstraído con el Servicio UniDA. Asimismo, gracias a las tecnologías de la Capa de Acceso a Dispositivos de la arquitectura HI3 sería sencillo programar aplicaciones que implementen lógicas de control complejas e independientes de que en el futuro se instalen nuevos enchufes o se sustituyan las tecnologías actuales por otras. Sin embargo, para hacer accesible la información de consumo energético histórica y agregada ha sido necesario desarrollar un nuevo servicio que procese y almacene los datos de consumo instantáneo. En la Figura 8.12 se muestra la ontología desarrollada para representar la información de consumo energético y las funcionalidades del servicio. La información histórica puede obtenerse para un enchufe concreto o agregada para todos los enchufes del entorno, y puede presentarse en intervalos horarios, diarios o mensuales. Este servicio, hace uso de las funcionalidades del Servicio UniDA para descubrir los enchufes monitorizables y para subscribirse a la información de consumo de los mismos.

Por último, en la Figura 8.13 se muestra una captura de pantalla de la aplicación de monitorización del consumo energético desarrollada como parte de la

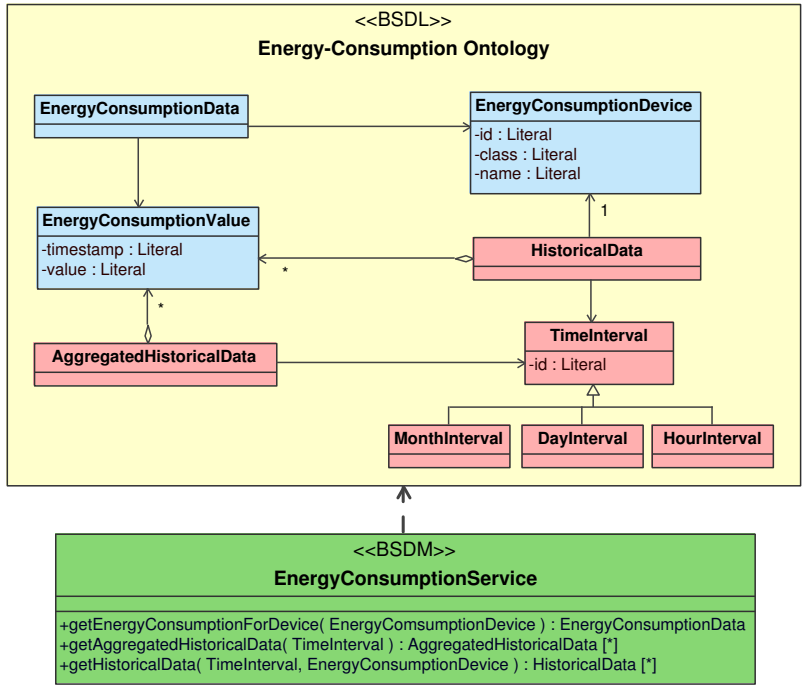



Figura 8.12: Ontología y servicio de consumo energético.


Figura 8.13: Aplicación de monitorización del consumo energético.

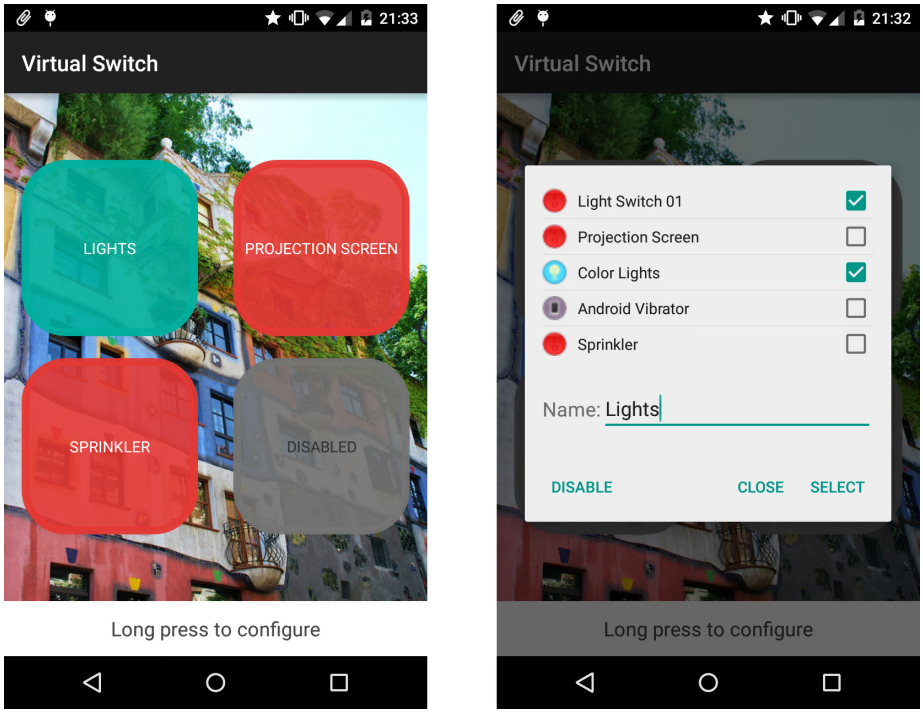



Figura 8.14: Aplicación para smartphones Android que permite crear interruptores virtuales asociados a dispositivos del entorno.

plataforma ONIZE para dispositivos Android. La aplicación permite visualizar tanto el consumo energético instantáneo como históricos de consumo para los distintos enchufes monitorizables presentes en el entorno. Para ello descubrirá y hará uso del servicio EnergyConsumptionService que se ha desarrollado.

### 8.1.4. Creando interruptores virtuales

Como vimos, con la aplicación Entorno-Conectado es posible controlar y monitorizar el entorno experimental desplegado con independencia de tecnologías concretas y de que se añadan nuevos dispositivos o desaparezcan otros existentes. En este apartado presentamos una sencilla aplicación para smartphones Android que permite a los usuarios definir en su dispositivo personal pulsadores virtuales que realicen acciones sobre dispositivos individuales o sobre grupos de dispositivos del entorno con estados compatibles. Esta aplicación se ha realizado sobre un contenedor microVineyard Android que proporciona un grounding de comunicación de tipo WebSocket para posibilitar la comunicación a través de Internet. Así, el usuario puede, por ejemplo, programar una serie de pulsadores virtuales para dispositivos de su hogar y hacer uso de los mismos desde cualquier parte.

En la Figura 8.14 se muestran dos capturas de pantalla de la aplicación desarrollada. En la primera pantalla pueden verse cuatro pulsadores (verde = estado binario activado, rojo = estado binario desactivado, gris = pulsador no asignado a ningún dispositivo). Los pulsadores pueden definirse sobre estados binarios de dispositivos UniDA. En la segunda captura de pantalla puede verse como se está definiendo una asociación de un pulsador con dos dispositivos de entre los disponibles (Color Lights y Light Switch 01), de modo que la actuación del pulsador actuará sobre los estados de ambos dispositivos.

Para su funcionamiento, esta aplicación, que se ha denominado VirtualSwitch, necesita descubrir funcionalidades de un Servicio UniDA del entorno que le permitirá resolver las siguientes operaciones: i) obtener información de aquellos dispositivos que tengan un estado binario actuable; ii) subscribirse a eventos de cambio de estado de los dispositivos para los que se crea un pulsador virtual; iii) actuar sobre los estados de los dispositivos asociados a pulsadores virtuales.

El conjunto de todas estas funcionalidades, resueltas a través de las aplicaciones y servicios que se han descrito, soportadas la arquitectura HI3 y desplegadas en un entorno experimental adecuado, constituyen un prototipo de un entorno inteligente de IAm que creemos es interesante para continuar desarrollando y probando nuevos conceptos y herramientas que puedan contribuir a construir entornos cada vez más adaptables a las necesidades, características y preferencias de las personas.

## 8.2. Un asistente virtual para personas dependientes

Las sociedades actuales tienden hacia un mayor envejecimiento de la población y en consecuencia tienden también hacia un aumento en el número de personas con enfermedades crónicas, con discapacidades o con limitaciones para realizar una vida plenamente independiente. En este contexto, las innovaciones tecnológicas tienen un campo de oportunidad para mejorar la calidad de vida de estas personas, proporcionándoles un mayor grado de autonomía y seguridad en su vida diaria. Así, hoy en día muchos investigadores están poniendo el foco en identificar formas en las que la tecnología puede aumentar la independencia, permitiendo que estas personas puedan continuar viviendo en sus hogares. Además, se ha demostrado que proporcionar formas de asistencia que únicamente atienden a aspectos funcionales y de salud puede disminuir la percepción que tiene el individuo de sus habilidades para llevar una vida independiente, mientras que otras intervenciones más centradas en necesidades emocionales y sociales pueden mejorar su actitud hacia una vida más independiente y satisfactoria [Secker et al., 2003].

En este ámbito, nuestro grupo de investigación ha colaborado con la empresa SCIO Innovation Technologies en el desarrollo de un sistema orientado al cuidado


Figura 8.15: Capturas de pantalla y mando a distancia del sistema OMNI.

en el hogar de personas mayores y personas dependientes. El sistema se denomina OMNI Asistente Virtual, y ha sido diseñado con el objetivo principal de aumentar el grado de autonomía de estas personas (patente [Bermúdez Pestonit et al., 2014]). Sus funcionalidades se centran en fomentar aspectos sociales y comunicativos a través de video-llamadas con familiares, amigos o centros de tele-asistencia; monitorización remota de la salud; y control de la rutina diaria a través de recordatorios y seguimiento de acciones.

El sistema se despliega en un dispositivo *set-top box* conectado a una pantalla de televisión. Las diferentes capacidades del sistema se integran de forma natural como nuevos canales de TV sin modificar el paradigma conocido de uso de una TV. Además, el sistema actúa de forma proactiva, haciendo preguntas al usuario para determinadas acciones y esperando una respuesta simple: 'Si' o 'No'. En la Figura 8.15 se muestran dos capturas de pantalla que ilustran funcionalidades del sistema, junto con el sencillo mando a distancia que permite a los usuarios interaccionar con el mismo. En la primera captura de pantalla se muestra un mensaje de recordatorio del sistema que se superpone al contenido normal de la TV y espera una respuesta del usuario. La segunda captura muestra un nuevo canal de la TV que está asociado con la función de iniciar una video-llamada con un contacto determinado.

Sobre la base del sistema OMNI se ha desarrollado una extensión que forma parte del trabajo de esta tesis. El objetivo principal de esta extensión es añadir un sistema general no intrusivo para la monitorización y alerta de incidencias ocurridas en el entorno de la persona dependiente. Sus principales componentes se ilustran en la Figura 8.16. Como podemos ver, el sistema opera desde dos puntos de vista: el de la persona dependiente y su hogar, y el de otra persona que se responsabilice de la persona dependiente (un familiar, personal de un centro de tele-asistencia, etc.). Es importante reseñar que la persona dependiente juega un papel fundamentalmente pasivo, de forma que el sistema no es intrusivo y es el entorno el que trabaja en pro del usuario. Así, el sistema desarrollado se fundamenta en los siguientes conceptos:

- En primer lugar, se basa en la idea de poblar el entorno de la persona dependiente con servicios que permitan monitorizar aspectos relacionados con su actividad, la seguridad del hogar o aspectos de salud. Éstos pueden ser incorporados explícitamente o pueden utilizar elementos ya existentes en el entorno, como el propio sistema OMNI.
- En segundo lugar, se define un modelo estándar para la representación y notificación de alertas e incidencias, independiente de los servicios y elementos concretos que las produzcan. Para ello se define una ontología para la descripción de alertas y un grupo de funcionalidades estándar, que pueden ser incorporadas a servicios ya existentes para dotarlos de esta nueva capacidad.
- La persona responsable de la persona dependiente dispone de una aplicación personal capaz de descubrir y subscribirse remotamente a cualquier servicio

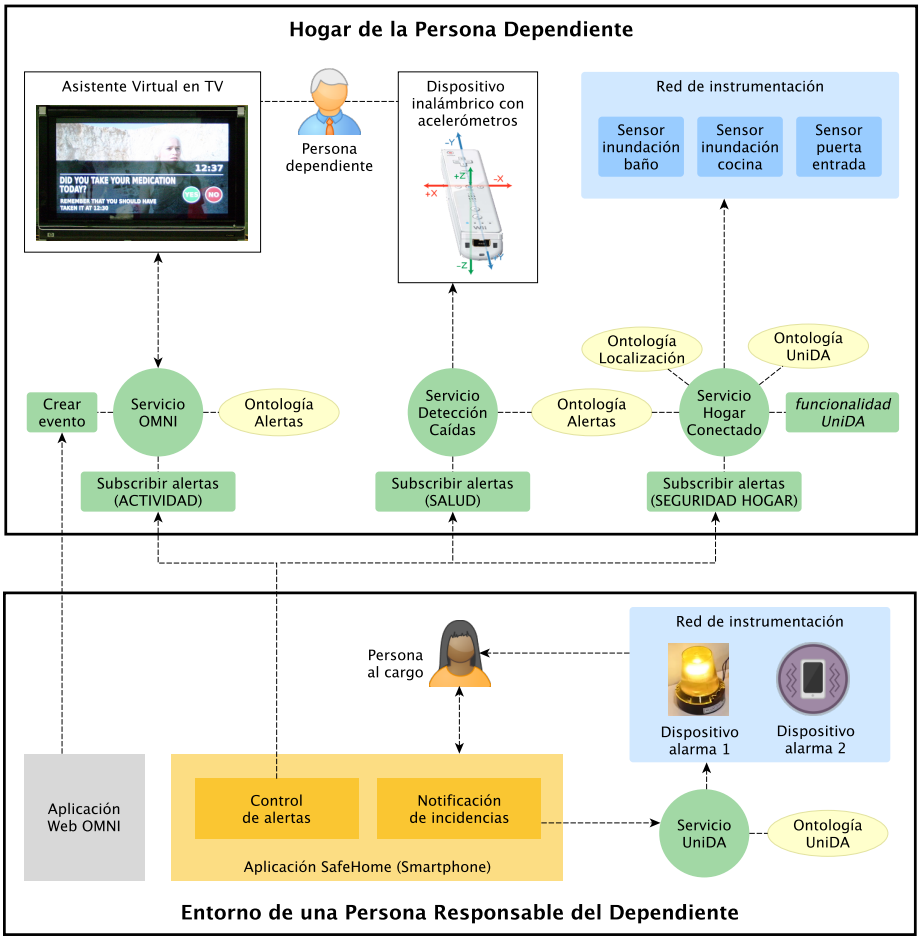



Figura 8.16: Arquitectura de la extensión desarrollada para el sistema OMNI.

del hogar del dependiente que incorpore capacidades estándar para la notificación de alertas. Además, esta persona podrá configurar su aplicación para que utilice elementos de su propio entorno físico para notificarle alertas de carácter grave. Para disponer de esta última capacidad, la persona al cargo debe tener una instancia de un Servicio UniDA ejecutándose en su entorno, de modo que le proporcione acceso a dispositivos a través de los que realizar las notificaciones. De esta forma podrán utilizarse métodos de notificación adecuados a las características del usuario que está al cargo, por ejemplo utilizando una señal luminosa si el usuario tiene dificultades auditivas, etc.

Creemos que este tipo de solución supone una mejora en el grado de independencia de estas personas desde dos puntos de vista. En primer lugar, en el ámbito de la seguridad de la persona dependiente, gracias a la monitorización continua de

parámetros en su entorno y a la notificación de incidencias detectadas a una persona al cargo sin necesidad de que esté presente en el domicilio del dependiente. En segundo lugar, en el ámbito emocional, dado que la simple existencia de esta solución puede mejorar la confianza de la persona dependiente para continuar viviendo en su hogar con un grado importante de autonomía, a la vez que puede aportar tranquilidad a sus seres queridos.

En la extensión del sistema OMNI que se ha realizado se proponen tres dominios de actuación para la monitorización del entorno de la persona dependiente: monitorización de la actividad, monitorización de aspectos relacionados con la salud y monitorización de aspectos relacionados con la seguridad del hogar. Veremos las soluciones concretas que se han propuesto dentro de estos dominios, y que se ilustran globalmente en la Figura 8.16.

En el ámbito de la monitorización de la actividad se propone aprovechar la propia interacción de la persona dependiente con el sistema OMNI como información de entrada para monitorizar ciertos aspectos de su actividad. Así, se desarrolló el Servicio OMNI que es capaz de detectar una falta continuada de actividad por parte del usuario en relación con interacciones demandadas por la plataforma OMNI TV, lo cuál podría ser un indicativo de alguna situación anómala. Asimismo, el servicio puede alertar de eventos/recordatorios importantes que el usuario no ha confirmado, como recordatorios de tomas de medicación. Creemos, además, que sería posible e interesante incorporar a este servicio algoritmos más complejos capaces de extraer patrones de actividad de la persona dependiente y detectar cambios significativos en los mismos a lo largo del tiempo, como posible síntoma de un problema de mayor calado en la salud física o mental de la persona, no obstante el desarrollo de esta tarea ha quedado fuera del alcance de este trabajo.

En el ámbito de la monitorización de la seguridad en el hogar se propone utilizar la instrumentalización del entorno para la detección de situaciones de riesgo. Así, en el entorno experimental desplegado se han incluido sensores de inundación y un sensor de apertura/cierre de puerta que permitiría detectar si la persona se deja la puerta exterior de casa abierta, por ejemplo. Se ha desarrollado una versión ampliada del Servicio UniDA, que hemos denominado Servicio HogarConectado, que incluye las capacidades del anterior y que además utiliza la ontología de alertas para notificar alertas sobre eventos en sensores que se le hayan configurado. Adicionalmente, utiliza también la ontología de localización e identificación desarrollada para el entorno ONIZE, para proporcionar información de la localización del lugar en el que se produce la alerta, cuando esta información esté disponible.

Relacionado con el ámbito de la monitorización de la salud, nuestro grupo de investigación ha realizado diferentes trabajos enfocados a la utilización de innovaciones tecnológicas para mejorar la calidad de vida de personas mayores. En ellos se ha puesto de manifiesto la gran relevancia de los incidentes relacionados con

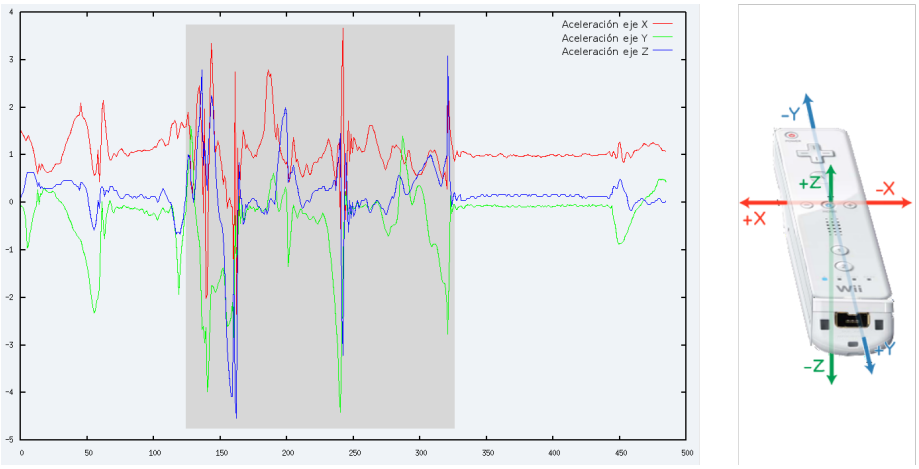



Figura 8.17: Señales de los acelerómetros en los ejes X, Y, Z durante una caída.

caídas en situaciones de soledad dentro del hogar o incluso en la habitación de una residencia. En muchas ocasiones las caídas se producen por una rotura de cadera o por una bajada del nivel de azúcar y pueden dejar a la persona en una situación de enorme estrés o con pérdida de conciencia, y sin posibilidad de pedir ayuda. Así, se propone un Servicio de Detección de Caídas, capaz de detectar la ocurrencia de situaciones de este tipo y generar alertas inmediatas. Para ello, se utiliza un sistema prototipo desarrollado en nuestro grupo de investigación basado en la utilización de un dispositivo con acelerómetros y comunicación inalámbrica que es portado por el usuario. En la Figura 8.17 se muestra una gráfica de las señales provenientes de los acelerómetros con una zona gris correspondiente a una situación de caída, junto con un mando de la consola Wii utilizado como dispositivo para los experimentos y dotado de acelerómetros y comunicación Bluetooth. El sistema utiliza técnicas de Redes de Neuronas Artificiales para intentar caracterizar las señales de caídas y podría ser combinado con otra información del entorno, como la detección del sonido de la caída y de lamentos, para disminuir el número de falsos positivos. En nuestro caso, el servicio de detección de caídas se complementa con el uso de la ontología y funcionalidades de notificación de alertas estándar desarrollada.

Con el fin de dotar a todos estos servicios de capacidades estándar para la publicación de alertas, se ha definido la ontología que se puede ver en la Figura 8.18. Existe una taxonomía de alertas que identifican el tipo de alerta producido. Éstas incorporan adicionalmente una descripción, un nivel de gravedad e información del instante temporal en que se produjo la alerta. A todos los servicios se les han incorporados dos funcionalidades de subscripción a alertas, a mayores de las funcionalidades relacionadas con su comportamiento concreto. Con la particularidad de que cada uno de los servicios genera alertas de diferente tipo. Las funcionalida-

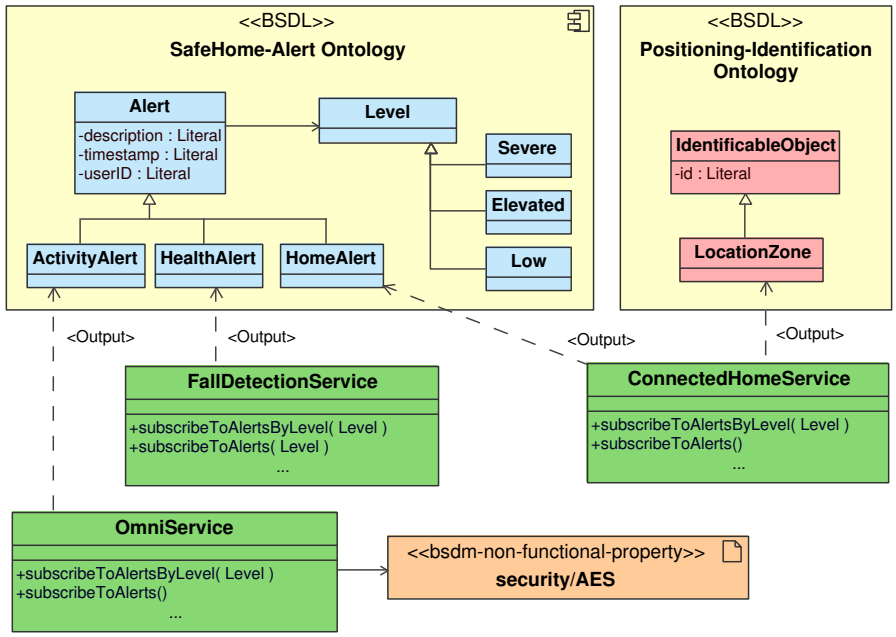



Figura 8.18: Ontología y servicios que proporcionan funcionalidades de alertas.

des de alerta del servicio ConnectedHome producen además como resultado una localización relativa a la ubicación del sensor que detecta la alerta (referida a la ontología de posicionamiento/identificación definida para la plataforma ONIZE). En el caso del Servicio OMNI, se ha añadido además una propiedad no-funcional para indicar que se trata de funcionalidades seguras que cifran la información sensible del usuario. En el Listado 8.2 se incluye un fragmento del descriptor BSDL del Servicio OMNI correspondiente a una de estas funcionalidades.

Una solución como esta, que permite dotar a servicios de dominios heterogéneos de capacidades homogéneas para la notificación de alertas relacionadas con su tarea, contempla algunos elementos interesantes para comparar el modo de resolverla en diferentes arquitecturas. La solución es altamente desacoplada de cualquier entorno o sistema concreto, ya que los clientes de los servicios probablemente no tengan conocimiento de antemano de estos ni tampoco entiendan otras funcionalidades que puedan ofrecer al margen de las relacionadas con alertas. Parece, por tanto, que el problema se adaptará mejor a una solución más general como Amigo. Al igual que la arquitectura HI3, Amigo permitiría crear aplicaciones capaces de interoperar con servicios de alertas de terceros implementados en distintas plataformas y descritos en distintos modelos de servicios. Además, Amigo es el único caso en el que es posible describir semánticamente un servicio con más de una funcionalidad y con propiedades no-funcionales. PERSONA y SOPRANO utilizan únicamente

Listado 8.2: Descripción de una funcionalidad de alertas del Servicio OMNI.

```
<import prefix="bsdmP" to="http://hi3project.com/broccoli/bsdm/profile"/>
<import prefix="alertOntology" to="http://hi3project.com/safehome/ontology"/>
<import prefix="omniServ" to="http://hi3project.com/safehome/omni/service"/>

<with property="advertisedSubscription">
<instance of="bsdmP#advertisedSubscription"
      URI="omniServ/profile#subscribeToAlerts">
   <with property="name" value="subscribeToAlerts"/>
   <with property="output">
      <with property="name" value="alert"/>
      <with property="type" valueURI="alertOntology#ActivityAlert"/>
   </with>
   <with property="nonFunctionalProperty">
      <instance of="bsdmP/nonFunctionalProperties/security/AES">
         <with property="keyLength" value="64"/>
      </instance>
   </with>
</instance>
```

descripciones semánticas de servicios basadas en OWL-S, el cuál no soporta dichas características. Sin embargo, en HI3 sería posible que una aplicación descubriese y utilizase de forma transparente tanto funcionalidades de alertas descritas en BSDM como en OWL-S, por ejemplo, asumiendo que estas últimas tendrían una menor riqueza semántica en su descripción.

Una limitación de Amigo para la implementación de esta solución es la ya mencionada carencia de un modo de interacción de tipo publicación/subscripción. Sin embargo, la solución presentada aquí, desacoplada de los clientes de los servicios de alerta y basada en la publicación asíncrona de eventos de alerta, parece la más general y adecuada. En PERSONA y SOPRANO sería posible realizar una implementación con una funcionalidad similar (sin utilizar propiedades no-funcionales) pero quedaría atada a sus plataformas de servicios OSGi y sus particularidades de funcionamiento interno, sacrificando por tanto la interoperabilidad con servicios y aplicaciones de alertas implementados en otras tecnologías. Por otro lado, en el caso de CHIL los servicios de alertas se implementarían como agentes Jade y podría definirse una ontología OWL para describir la información de alertas proporcionada por estos. En este caso la interoperabilidad con otros sistemas también se resiente debido a la ausencia de un modelo semántico de descripción de servicios y a la dependencia de elementos concretos de la plataforma de agentes.

En la Figura 8.19 vemos un diagrama de despliegue de la extensión desarrollada para el sistema OMNI utilizando la arquitectura HI3. En el hogar del dependiente tenemos un contenedor yottaVineyard que ejecuta todos los servicios desarrollados, incluyendo la Capa de Dispositivos para controlar el entorno físico. En el entorno de una persona al cargo del dependiente (e.j. hogar de un familiar) tenemos otro

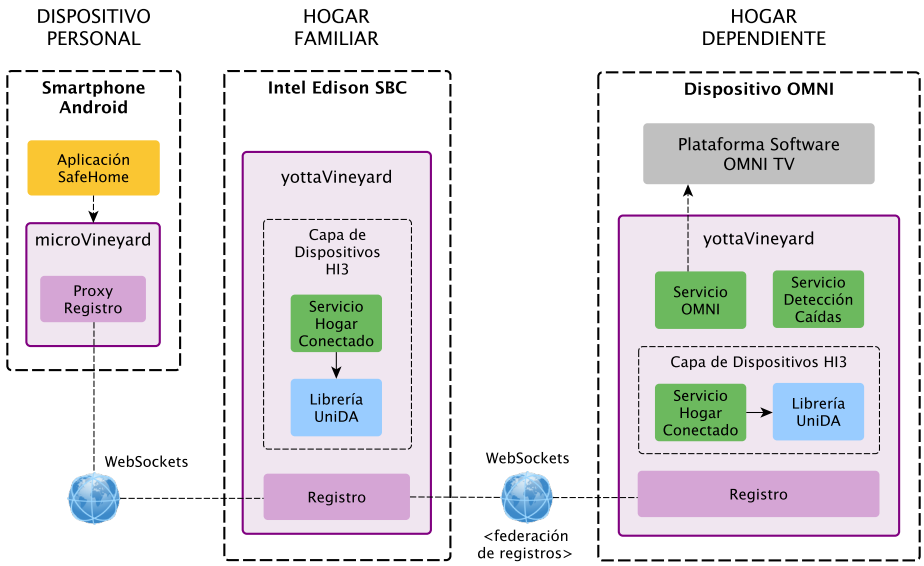



Figura 8.19: Despliegue de la extensión del sistema OMNI para la monitorización del entorno de un dependiente en base a componentes de la arquitectura HI3.

contenedor yottaVineyard federado con el anterior. En él se ejecutan servicios propios de este usuario, incluyendo un Servicio UniDA para acceso al entorno físico. Por último, un contenedor microVineyard se ejecuta en un dispositivo smartphone Android de la persona al cargo. Sobre él se ejecuta la aplicación SafeHome, capaz de descubrir, a través de la federación de registros, los servicios de monitorización existentes en el entorno remoto de la persona dependiente y suscribirse a sus alertas. La aplicación se ha desarrollado totalmente desacoplada del entorno y de los servicios concretos de alerta. Así, utilizando las capacidades de la arquitectura HI3, la aplicación SafeHome puede soportar, de forma transparente, la monitorización de más de un hogar de personas dependientes simplemente a través de la federación de nuevos contenedores de servicios OMNI de otros hogares.

En la Figura 8.20 vemos dos capturas de pantalla de la aplicación SafeHome para smartphones Android. En su pantalla principal recibe alertas de todos los servicios a los que está subscrita, y las identifica según su nivel de gravedad. En una segunda pantalla permite definir dispositivos del entorno del usuario de la aplicación que desea que se utilicen para notificar alertas graves recibidas. En el ejemplo experimental se ha asociado con una sirena luminosa actuada remotamente a través de un enchufe controlable ONIZE.

Para poder recibir alertas y monitorizar el entorno de la persona dependiente, la aplicación SafeHome trata de descubrir, a través del registro, servicios con funcionalidades que proporcionen información de alertas. En el Listado 8.3 se incluye dos

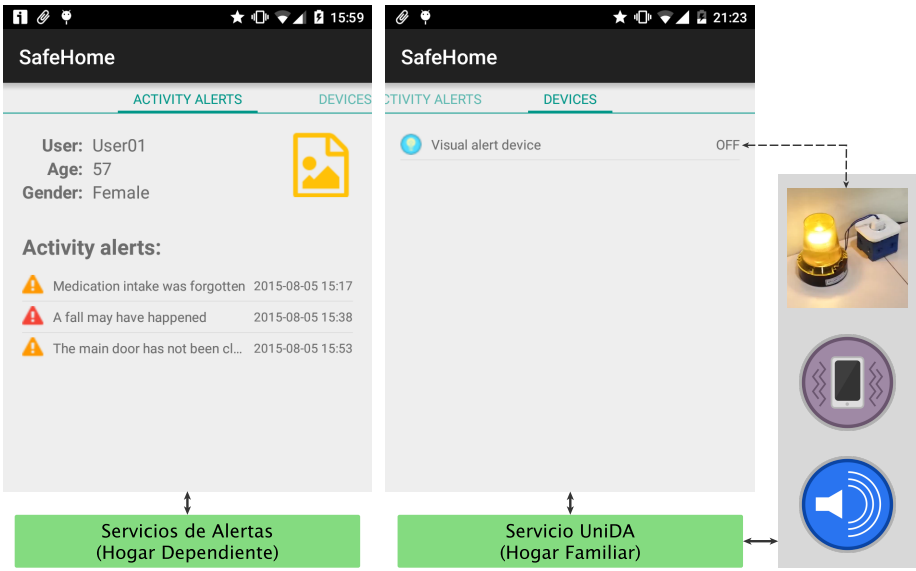



Figura 8.20: Capturas de pantalla de la aplicación SafeHome.

ejemplos de búsqueda que se han utilizado en la aplicación SafeHome como parte del entorno experimental construido. La primera permite hacer una búsqueda de funcionalidades que generen resultados de cualquier tipo de alerta, filtrados por el nivel de alerta indicado en un Input. En el segundo caso se especifica un objetivo de búsqueda para cualquier funcionalidad que produzca alertas de tipo ActivityAlert, indicando, con una propiedad no-funcional, que se demanda que la funcionalidad incorpore una característica de seguridad para el cifrado de la información.

Tras ejecutar una búsqueda de funcionalidades en el registro, la aplicación SafeHome puede fácilmente subscribirse a todas las funcionalidades de notificación de alerta que están disponibles en ese momento, y que por tanto obtuvo como resultado de la búsqueda. En el Listado 8.4 se muestra un fragmento de código de la aplicación SafeHome que le permite invocar todas las funcionalidades que le devolvió el registro de servicios como respuesta a la ejecución de la primera consulta del Listado 8.3. Observamos como la aplicación necesita enviar un Input como parámetro de las funcionalidades para indicar el nivel de alerta (un subconcepto de Level en la ontología) por el que quiere filtrar los resultados.

Vemos como, utilizando estos mecanismos, se consigue desacoplar a las aplicaciones de los servicios que monitorizan el entorno de la persona dependiente y generan los eventos de alerta. Por un lado, los servicios se desarrollan de forma independiente a quien los vaya a utilizar, ya que las funcionalidades relacionadas con alertas únicamente se preocupan de publicar los eventos de alerta. Por otro lado, los clientes pueden subscribirse a alertas para cualquier servicio existente en el

Listado 8.3: Objetivos de búsqueda para funcionalidades de subscripción a alertas.

```
<import prefix="bsdmProfile" to="http://hi3project.com/broccoli/bsdm/profile"/>
<import prefix="alertOntology" to="http://hi3project.com/safehome/ontology"/>

<!-- Búsqueda para cualquier alerta y posibildad de filtrado por nivel -->

<instance of="bsdmProfile#requestedFunctionality"
URI="http://hi3project.com/safehome/client#requestAlertNotifications">
   <with property="input">
      <with property="name" value="Nivel de alerta"/>
      <with property="type" valueURI="alertOntology#Level"/>
   </with>
   <with property="output">
      <with property="name" value="Alerta de cualquier tipo"/>
      <with property="type" valueURI="alertOntology#Alert"/>
   </with>
</instance>

<!-- Búsqueda para ActivityAlert y seguridad de cifrado en la funcionalidad -->

<instance of="bsdmProfile#requestedFunctionality"
URI="http://hi3project.com/safehome/client#requestActivityAlertNotificationsSecure">
   <with property="output">
      <with property="name" value="Alertas de actividad"/>
      <with property="type" valueURI="alertOntology#ActivityAlert"/>
   </with>
   <with property="nonFunctionalProperty">
      <instance of="bsdmProfile/nonFunctionalProperties/security/AES"/>
   </with>
</instance>
```

entorno con independencia de cuál sea la tarea principal de dicho servicio.

## 8.3. Resumen

En este capítulo se ha abordado la conceptualización y desarrollo de dos sistemas de IAm en base a los conceptos y capacidades proporcionados por la arquitectura HI3. A través de ambos casos se ha tratado de ilustrar, en primer lugar, como los desarrolladores pueden construir nuevas aplicaciones altamente desacopladas del contexto en el que se van a ejecutar. Entendiendo que este contexto incluye al entorno físico, a los usuarios y a otros componentes funcionales que pueden estar o no disponibles en un escenario de ejecución concreto. En segundo lugar, se ha buscado ilustrar como esas aplicaciones, gracias a las capacidades proporcionadas por la arquitectura HI3, pueden adaptarse a cada contexto y colaborar de forma flexible con los componentes funcionales disponibles en el entorno.

Vimos ejemplos en los que la variedad de entornos físicos y computacionales que acompañan a las personas en su vida diaria dificultan la ubicuidad y adaptación de

Listado 8.4: Fragmento de código de la aplicación SafeHome para subscribirse a funcionalidades de alerta previamente descubiertas a través del registro.

```
final IParameterConverter parameterConverter = this.clientLoader.getParameterConverter();
Collection<IAnnotatedValue> inputs = new ArrayList<>();
inputs.add(parameterConverter.createAnnotatedValue(new Severe()));

for (IFunctionalitySearchResult searchResult : message.getSearchResults()) {
    searchResult.getServiceDescription().subscribeTo(
        searchResult.getServiceDescription(),
        inputs,
        new IAsyncMessageClient() {
            public void receiveMessage(IMessage msg) throws ModelException {
                if (msg instanceof FunctionalityResultMessage) {
                    IResult result =
                        ((FunctionalityResultMessage) msg).getResult().iterator().next();
                    if (result instanceof IOutputValue) {
                        Object outputToObject =
                            parameterConverter.outputToObject((IOutputValue) result);
                        if (outputToObject instanceof Alert) {
                            // Hacer algo con la Alert recibida
                        }
                    }
                }
            }});
}
```

los sistemas de IAm. También ilustramos como las propuestas de la arquitectura HI3 basadas en diferentes tipos de contenedores de servicios, con distintas capacidades pero capaces de colaborar entre ellos, facilitan esta tarea.

Asimismo, quedó patente la necesidad de soportar comunicaciones desacopladas entre los componentes funcionales de sistemas de IAm. Interacciones espontáneas basadas en eventos asíncronos y en patrones de comunicación uno-a-muchos son frecuentes y demandadas por este tipo de sistemas. Así, vimos diferentes casos de servicios que resuelven una tarea en su entorno y son capaces de producir resultados de forma puntual y en respuesta a cambios en el entorno, esperando que otros componentes funcionales a priori desconocidos muestren su interés este tipo de información. Se ilustró como se resuelven este tipo de interacciones con el modelo avanzado de interacción de la arquitectura HI3, en oposición a la habitual falta de soporte para las mismas en otras tecnologías de servicios semánticos existentes.

Igualmente, a través de diferentes ejemplos se ha podido comprobar como, utilizando el modelo de servicios BSDM y el lenguaje BSDL, resulta sencillo e intuitivo definir nuevos servicios semánticos, permitiendo que los desarrolladores se centren en la programación de la funcionalidad.

Por tanto, estos casos experimentales han permitido validar la adecuación de la arquitectura HI3 al proceso de desarrollo de sistemas de IAm partiendo de un núcleo de requisitos fundamentales y dejando la puerta abierta a su futura extensión.

# Capítulo 9

# Conclusiones y trabajo futuro

Para finalizar, en este capítulo se realiza un análisis de los resultados del trabajo realizado y de las conclusiones que pueden extraerse de ellos. Asimismo, se introducen las principales líneas de trabajo futuro que han surgido durante la realización de esta tesis y que permanecen abiertas a la investigación.

## Conclusiones

En el Capítulo 2 se presentó el objetivo general de este trabajo: proporcionar un soporte para la construcción de sistemas de Inteligencia Ambiental (IAm) partiendo de una aproximación uniforme e independiente de entornos concretos y de las características particulares de los usuarios. Posteriormente, vimos que para abordar este objetivo tan amplio es necesario enfrentarse a un campo multidisciplinar en el que los esfuerzos están muy dispersos y a menudo se adolece de una visión integrada y focalizada de los mismos. Así, a lo largo de esta tesis se han definido e implementado métodos para avanzar en este camino siguiendo una aproximación homogénea e integrada soportada por una arquitectura software para sistemas de Computación Ubicua e IAm.

De forma más concreta, siguiendo los objetivos de este trabajo, buscábamos focalizar de manera especial los esfuerzos en dos aspectos fundamentales:

- Facilitar que los desarrolladores puedan centrar sus esfuerzos en la implementación de funcionalidades de IAm de forma aislada a la problemática común a este tipo de entornos, y en especial con independencia de particularidades de tecnologías y middleware heterogéneos.
- Facilitar el desarrollo de aplicaciones ubicuas capaces de adaptarse a distin-

tos escenarios, y en particular a las características de los usuarios y de los entornos físicos.

Para dar respuesta a estos objetivos esta tesis ha producido como resultado principal la arquitectura software HI3 para entornos de Computación Ubicua e IAm. Se trata de una arquitectura de carácter general para IAm que proporciona una gran flexibilidad para su aplicación en diferentes ámbitos, gracias a una aproximación orientada a servicios en la que tanto las capacidades de la propia arquitectura como las funcionalidades de sistemas concretos son exportados como componentes homogéneos semánticamente descritos, con independencia de las particularidades de implementación y la plataforma de ejecución. La arquitectura HI3 proporciona además unos modelos conceptuales y unas guías de diseño que sirven como referencia para la construcción de un nuevo sistema de IAm. De forma más concreta, como parte de la arquitectura HI3 podemos considerar que se han obtenido los siguientes resultados:

- Un middleware orientado a servicios para entornos ubicuos que proporciona una infraestructura escalable y capaz de abordar la heterogeneidad de lenguajes y modelos de servicios, la heterogeneidad de protocolos de comunicación y la heterogeneidad de plataformas de implementación y ejecución. Por tanto, se consigue abstraer al desarrollador de dicha heterogeneidad y se posibilita la interoperabilidad con otros componentes propios y de terceros. Además, derivados de este middleware se han obtenido una serie de resultados relevantes adicionales:
  - Una implementación del middleware orientado a servicios que puede ser utilizada al margen de la arquitectura HI3, como una arquitectura SOA de carácter general.
  - Una implementación del framework de servicios semánticos Broccoli, que puede ser utilizada de forma independiente para la descripción semántica de las capacidades (funcionales, no-funcionales y de comunicación) de cualquier componente software.
  - Una ontología del modelo uniforme de servicios BSDM y un conjunto de especificaciones del middleware orientado a servicios que permite la realización de nuevas implementaciones en distintas plataformas y lenguajes.
- Una serie de frameworks y servicios integrados en la arquitectura HI3 que proporcionan soluciones a requisitos de IAm relacionados principalmente con la adaptación dinámica de las aplicaciones a los usuarios y a los entornos físicos.
- Una definición conceptual y una implementación completa de la arquitectura HI3 con capacidad para ser extendida con nuevos frameworks y servicios que proporcionen soluciones a requisitos comunes de sistemas de IAm.

Respecto al objetivo de concebir mecanismos que faciliten la implementación de componentes funcionales utilizando los paradigmas y las plataformas computacionales que los desarrolladores consideren más adecuados y a la vez soportando la interoperabilidad entre componentes y sistemas, vimos en el Capítulo 5 como la arquitectura HI3, y en particular el modelo de servicios BSDM, no restringen las tecnologías utilizadas para la implementación de funcionalidades. No obstante, el modelo BSDM sí admite la definición de modelos de implementación para tecnologías concretas que faciliten todavía más al desarrollador la implementación de funcionalidades de servicios. Así, en la implementación de referencia de la arquitectura HI3 realizada se incluye un modelo de implementación para el lenguaje Java que permite programar implementaciones de servicios libres de lógica de control de la interacción con los clientes del servicio. Asimismo, en el Capítulo 6 se ilustró como en la arquitectura HI3 es posible utilizar un paradigma completamente diferente para resolver la funcionalidad de un servicio, en este caso el paradigma de Sistemas Multi-Agente. Por otro lado, utilizando las capacidades de abstracción e integración de otras tecnologías proporcionadas por el lenguaje BSDL y el modelo BSDM es posible alcanzar un alto nivel de interoperabilidad entre componentes con independencia de las particularidades de implementación de los mismos.

En referencia al objetivo de facilitar el desarrollo de sistemas capaces de colaborar, para la resolución de sus tareas, con otros componentes funcionales a priori desconocidos, el modelo de servicios de la arquitectura HI3 abstrae la interacción con independencia de la lógica de control de la comunicación y de los protocolos concretos de comunicación utilizados. Además, el modelo de interacción propuesto abarca una riqueza de interacciones que incluye comunicaciones síncronas y asíncronas, comunicaciones de tipo petición/respuesta y comunicaciones de tipo publicación/subscripción. Esto, junto con las capacidades de composición dinámica de servicios y las capacidades de interoperabilidad permiten a los desarrolladores resolver la funcionalidad de sus aplicaciones en base a la interoperación espontánea con los componentes disponibles en cada entorno. Los entornos de IAm demandan este tipo de interoperación ya que serán poblados por componentes de múltiples desarrolladores cuya disponibilidad y capacidades pueden variar en el tiempo. Así, en los Capítulos 7 y 8 pudimos comprobar como un modelo de interacción rico como el de la arquitectura HI3 resulta fundamental para posibilitar el desarrollo de componentes autónomos y altamente desacoplados capaces de establecer dinámicamente colaboraciones con componentes de terceros a priori desconocidos.

Respecto al objetivo de reducir el coste de desarrollar aplicaciones capaces de operar en diferentes entornos físicos donde las personas realizan sus actividades, con independencia de las tecnologías de dispositivos y de comunicaciones necesarias para interaccionar con los mismos, la arquitectura HI3 está dotada de un conjunto integrado de soluciones capaces de proporcionar un acceso uniforme, distribuido y ubicuo a redes heterogéneas de dispositivos presentes en los entornos

de IAm. Vimos en la sección 7.2 como las aplicaciones de IAm podrán obtener información contextual del entorno o controlar el mismo utilizando un paradigma de operación del entorno descrito en un lenguaje neutral de ontologías y accesible a través de un servicio semántico HI3, que a su vez podrá abstraer distintos frameworks de IoT heterogéneos.

Para dar respuesta al objetivo de facilitar el desarrollo de aplicaciones capaces de operar con usuarios con diferentes características, necesidades y preferencias, vimos en la sección 7.3 como la arquitectura HI3 proporciona una serie de soluciones que facilitan el acceso ubicuo y uniforme a información del perfil de los usuarios proveniente de fuentes heterogéneas. Asimismo la arquitectura integra, abstrae y proporciona acceso ubicuo a soluciones que posibilitan la interacción natural y adaptada con los usuarios en función de sus características. Utilizando ambas capacidades, los desarrolladores evitan la enorme complejidad de tener que programar de antemano sus aplicaciones de forma predefinida para cada tipo de usuario que se pueden encontrar en el entorno.

El objetivo de soportar la ubicuidad y movilidad de los sistemas de IAm gracias a la adaptación dinámica de sus componentes a la situación sin necesidad de una programación específica para cada escenario, se consigue en base a una utilización conjunta de todos los resultados de esta tesis. Dando respuesta a este objetivo se facilita la creación de sistemas que exhiben un comportamiento adaptado en el momento y el lugar donde los usuarios lo requieren. Para ello, los desarrolladores pueden explotar las capacidades de la arquitectura HI3 relacionadas con el descubrimiento, interacción espontánea, interoperabilidad, adaptación al usuario y adaptación al entorno físico para construir sistemas capaces de resolver su funcionalidad en distintos contextos en base a la colaboración con los componentes funcionales más adecuados presentes en cada contexto de ejecución.

En lo referente al objetivo de posibilitar la construcción de sistemas de IAm escalables, capaces de soportar un número a menudo impredecible de elementos distribuidos (dispositivos, usuarios, redes de comunicación, etc.), podemos considerar varias capacidades de la arquitectura HI3. En primer lugar, la ausencia de elementos únicos y centralizados de los que dependa el funcionamiento de los componentes funcionales (servicios y aplicaciones) de un sistema de IAm. En este sentido, la arquitectura soporta la distribución y federación de múltiples registros de servicios. Asimismo, dado que se sigue la filosofía de abstraer el acceso a frameworks que resuelven problemas concretos de IAm a través de servicios, nada impide que se desplieguen diferentes instancias distribuidas de dichos servicios en función de las demandas de un entorno determinado. Además, como se ilustró en los casos de evaluación del Capítulo 8, también es posible desplegar distintas instancias de la arquitectura en plataformas computacionales con distintas capacidades gracias a la disponibilidad de diferentes contenedores Vineyard. Por último, la arquitectura HI3 carece de una estructura rígida y encorsetada que condicione notablemente la

organización y relaciones entre componentes de un sistema construido en base a la misma. Al contrario, gracias a todas las características mencionadas previamente se caracteriza por la flexibilidad en el despliegue de la misma según las necesidades de un entorno determinado.

Finalmente, un objetivo que consideramos clave para este trabajo es el relacionado con proporcionar acceso público a un conjunto de especificaciones y herramientas software de referencia que permitan a la comunidad científica tanto evaluar y utilizar los resultados como construir nuevas soluciones e implementaciones compatibles con los mismos. Así, desde sus orígenes se construyó la arquitectura HI3 con vocación de constituir una plataforma viva que pueda ser extendida a través de nuevos desarrollos y a través de la integración de middleware proveniente de otros trabajos. Por ello se partió del objetivo de abordar de forma integrada y uniforme una serie de requisitos fundamentales para construir sistemas de IAm, que progresivamente podrán ser ampliados en el futuro. En consecuencia, tanto las especificaciones necesarias para construir nuevas implementaciones de referencia de la arquitectura HI3 como el software correspondiente a la implementación de referencia del núcleo de la arquitectura HI3 generada en esta tesis se han publicado bajo una licencia software de código libre y se encuentran disponibles públicamente para su descarga y uso. En concreto, en https://github.com/GII/broccoli/ y en https://github.com/GII/vineyard/ se encuentran publicados respectivamente los frameworks Broccoli y Vineyard que constituyen el middleware orientado a servicios de la arquitectura HI3. Mientras que en https://github.com/GII/broccoli/specification/bsdmOntology.xml se encuentra publicada la ontología que describe los elementos del modelo de servicios BSDM. Asimismo, en el Apéndice A se incluyen las principales especificaciones de la arquitectura.

Por otro lado, en el Capítulo 3 se realizó un extenso análisis y una comparativa de arquitecturas software y middleware para la construcción de sistemas de IAm. Se concluyó que no existe ninguna aproximación que se haya erigido como una solución de referencia y que muchos de los requisitos que deben abordarse están relacionados con líneas de investigación que permanecen abiertas y que están lejos todavía de los ambiciosos objetivos de la IAm. Es más, en los últimos años los esfuerzos de investigación parecen centrarse en ámbitos más acotados como el IoT o la Computación Móvil. Como hemos visto, la arquitectura HI3 pone especial énfasis en la generalidad y escalabilidad de la solución así como la abstracción de la heterogeneidad en todos los aspectos, reconociendo que la proliferación de middleware heterogéneo y no interoperable simplemente está trasladando el problema a un nivel superior de abstracción. De acuerdo con la tabla comparativa entre arquitecturas presentada en la Sección 3.4.7, en la Figura 9.1 presentamos una nueva tabla en la que se ha añadido una nueva columna correspondiente a la arquitectura HI3. En este caso se ha añadido una nueva notación, que consiste en añadir un símbolo + al grado de cumplimiento de una característica para indicar que dentro

| | AMIGO | CHIL | LAICA | OXYGEN | PERSONA | SOPRANO | HI3 |
|---|---|---|---|---|---|---|---|
| Heterogeneidad | Media | Media | Baja | Baja | Media | Media | Alta |
| Escalabilidad | Alta | Media | Alta | Alta | Media | Baja | Alta |
| Sensibilidad al contexto | Media | Baja | No | Baja | Media | Media | Media |
| Interacción natural | Baja | Media | No | Media | Baja | No | Media+ |
| Movilidad | Baja | No | Baja | No | No | No | Baja+ |
| Invisibilidad | Media | Media | Baja | Media | Media | Media | Alta |
| Interoperación espontánea | Media | Baja | No | Baja | Baja | Media | Media+ |
| Interoperabilidad | Media | Baja | No | No | Baja | Baja | Media |
| Integridad y seguridad | Baja | Media | Media | Baja | Baja | Baja | Media |
| Privacidad y confianza | Media | Baja | No | Media | Baja | No | Baja |
| Inteligencia social | No | No | No | No | No | No | No |
| Facilidades desarrollador | Media | Media | Baja | Baja | Baja | Media | Media |
| Acceso y vigencia | Baja | Baja | Baja | Baja | Baja | Baja | Alta |
| Generalidad | Alta | Media | Baja | Media | Media | Baja | Alta |

Tabla 9.1: Comparación de la arquitectura HI3 frente a las principales arquitecturas existentes en el ámbito de la IAm y que fueron revisadas en el Capítulo 3.

de un mismo nivel una aproximación destaca sobre las demás.

En consecuencia con todo lo analizado previamente, podemos ver como la arquitectura HI3 se comporta razonablemente bien ante el conjunto de requisitos establecido y frente al resto de arquitecturas. No obstante, hay que tener en cuenta que la investigación en IAm se haya muy lejos de los objetivos establecidos en sus principios y que esto es aplicable a gran parte de las características tomadas en cuenta en la comparativa. Teniendo esto en cuenta, podemos ver que la arquitectura HI3 comparte con Amigo un enfoque de base ambicioso en cuanto a su generalidad, escalabilidad e interoperabilidad. En este último aspecto, a pesar de que ambas aproximaciones incluyen los mayores esfuerzos por proporcionar mecanismos de interoperabilidad con otras tecnologías obtienen una calificación Media debido a que con carácter general la interoperabilidad plena entre sistemas heterogéneos está lejos de ser alcanzada. Algo similar ocurre con la interoperación espontánea, ya que se trata de una característica que involucra a numerosos aspectos y que también está relacionada con la interoperabilidad. En el caso de la arquitectura

HI3, vimos como el modelo avanzado de interacción proporcionado favorece esta característica frente a otras aproximaciones. Tanto la arquitectura Amigo como la SOPRANO reciben una calificación próxima a HI3 en la tabla comparativa, sin embargo es pertinente comentar que la solución de HI3 es de carácter más general. En el caso de Amigo debido al alcance más limitado de su modelo de interacción (basado únicamente en protocolos RPC) y en el caso de SOPRANO debido a que la interoperación está limitada a componentes de la propia plataforma.

En cuanto a la característica de sensibilidad al contexto, la arquitectura HI3 y algunas otras aproximaciones incluyen mecanismos que soportan la obtención de información contextual de distinto tipo y facilitan su utilización para la adaptación de las aplicaciones al contexto. No obstante, en todos los casos, estas capacidades podrían ampliarse incluyendo mecanismos más avanzados para la gestión de este tipo de información, como veremos en la sección dedicada al trabajo futuro.

En el caso de la interacción natural, aunque en este trabajo no se incluye el desarrollo de dispositivos y modos concretos de interacción natural como ocurre en proyectos como CHIL y Oxygen, la arquitectura HI3 proporciona un conjunto de mecanismos que permiten a los desarrolladores construir aplicaciones capaces de adaptar dinámicamente la interacción con el usuario en función del escenario (como se explicó en la Sección 7.3).

Como hemos visto desde el comienzo de esta sección, la construcción de la arquitectura HI3 se ha centrado en primer lugar en satisfacer requisitos que de forma general soporten la construcción de sistemas en entornos dinámicos, parcialmente desconocidos de antemano, impredecibles y poblados con componentes funcionales y tecnologías provenientes de orígenes heterogéneos. El middleware orientado a servicios de la arquitectura HI3 aborda esta problemática aportando soluciones para la heterogeneidad, escalabilidad, movilidad, interacción espontánea, interoperabilidad, integridad y seguridad. En segundo lugar, en la construcción de la arquitectura HI3 se han focalizado los esfuerzos en dos elementos centrales de cualquier sistema de IAm como son los usuarios y los entornos físicos. Así, en el capítulo 7 vimos como a través de abordar la heterogeneidad de los entornos, utilizar información contextual y soportar la interacción natural adaptada, en conjunción con las capacidades del middleware orientado a servicios, la arquitectura HI3 soporta a los desarrolladores en la construcción de aplicaciones capaces de adaptarse a los usuarios y sus entornos de forma no intrusiva.

# Trabajo futuro

En este trabajo hemos diseñado, implementado y demostrado una solución integral para la construcción de sistemas de IAm, siguiendo una aproximación uniforme basada en la utilización de una arquitectura software adaptada a los principales requisitos de dichos sistemas. Dada la amplitud del problema y la inmadurez actual del área de la IAm, ya desde el principio del trabajo se planteó una simplificación en torno a un conjunto de requisitos que se consideraron fundamentales sobre otros. Vimos también, como incluso en aquellos aspectos que se abordaron con mayor profusión nos encontramos lejos de la ambiciosa visión de la IAm. Además, durante la realización del trabajo también hemos identificado otros aspectos dónde se requerirá una futura investigación y desarrollo.

## Mejorar el proceso de desarrollo desde el diseño

La diversidad y complejidad inherente a los entornos de IAm dificulta, desde los primeros pasos, el desarrollos de sistemas de IAm. La arquitectura HI3, desde sus modelos conceptuales hasta su middleware y herramientas proporcionan un importante grado de simplificación y abstracción en el desarrollo de este tipo de sistemas. No obstante es inevitable que parte de la complejidad del dominio del problema se traslade al diseño del sistema de IAm y a la utilización del middleware. Para disminuir todavía más esta complejidad es posible introducir en el diseño de sistemas de IAm técnicas relacionadas con la aproximación MDA (del inglés, Model Driven Architecture).

Existen algunas aproximaciones de tipo MDA para el diseño de sistemas basados en una arquitectura SOA. No obstante, será necesaria una investigación adicional para aplicar este tipo de técnicas al diseño de sistemas basados en conceptos y elementos del ámbito de la Computación Ubicua y la Inteligencia Ambiental. En concreto, para la arquitectura HI3 planteamos la definición de un lenguaje de modelado que incluya los conceptos software manejados por la arquitectura de modo que se facilite el diseño software de sistemas de IAm y su posterior implementación sobre elementos del middleware de la arquitectura. Una posible aproximación al problema sería la extensión de algún lenguaje de modelado software existente, como UML o SoaML, para considerar conceptos y elementos manejados por la arquitectura HI3.

Más allá de mejorar el proceso de diseño de estos estos sistemas, otro camino de investigación futura estaría relacionado con la creación de mecanismos que automaticen en parte la implementación a partir del diseño y la incorporación de estos mecanismos al proceso de desarrollo de sistemas de IAm. En particular, sobre la arquitectura HI3 parece factible incorporar cierto nivel de asistencia en la imple-

mentación de servicios individuales o incluso composiciones de servicios a partir de un modelo de diseño de los mismos.

## Ampliar las capacidades de composición dinámica de servicios

La investigación relacionada con mecanismos que posibiliten la composición dinámica de servicios es una de las más relevantes para la IAm en el ámbito de la Computación Orientada a Servicios. En base a la composición dinámica de funcionalidades, a priori desconocidas pero que están disponibles en un contexto determinado de ejecución, es posible adaptar el comportamiento de un sistema al escenario. En la arquitectura HI3 se han propuesto algunos mecanismos para realizar este tipo de composición utilizando las capacidades del middleware orientado a servicios desarrollado. Un línea interesante de investigación podría estar relacionada con la mejora de este tipo de mecanismos a través de la utilización de mayor información en el proceso de composición o de la incorporación de técnicas provenientes de otras áreas como la Inteligencia Artificial.

Así, sería interesante introducir en el proceso mayor información relacionada con el contexto de ejecución de la propia composición dinámica con el fin de adaptarla de manera más adecuada a la situación. Por ejemplo, utilizando información dinámica obtenible del entorno relacionada con la calidad de servicio.

Otra aproximación a investigar podría estar relacionada con la introducción de información difusa e información con incertidumbre tanto en las entradas como en los resultados de las funcionalidades de los servicios. Cuando hablamos de entornos complejos y abiertos de IAm, es fácil imaginar que los sistemas deberán manejar este tipo de información para adaptarse a los mismos.

## Mejorar la interoperabilidad entre sistemas heterogéneos

La interoperabilidad real y completa entre sistemas desarrollados aisladamente continúa estando a día de hoy lejos de ser una realidad. A pesar de los avances tanto en la interoperabilidad a nivel semántico como sintáctico, la inmensa diversidad de lenguajes, plataformas computacionales, protocolos y redes de comunicación, modelos de interacción, etc., hacen de esta característica uno de los retos más importantes de los sistemas computacionales en general. Hemos visto como la arquitectura HI3 incorpora diferentes mecanismos de interoperabilidad relacionados principalmente con los lenguajes de descripción de ontologías y servicios, los modelos de descripción semántica de servicios, los modos de interacción y los protocolos de comunicación entre servicios.

En este sentido, una línea de investigación interesante estaría relacionada con mejorar la interoperabilidad a nivel de las redes de comunicación. En un sistema de

IAm podremos encontrarnos elementos conectados a través de multitud de redes: Bluetooth, redes móviles, Internet, redes locales, redes personales, etc. Para lograr la invisibilidad e interoperabilidad de los sistemas es necesario tener en cuenta esta heterogeneidad y conseguir abstraer a los componentes funcionales de la misma. Además, es posible que dichos componentes deban adaptar autónomamente su comportamiento a las características de las redes a través de las que se están comunicando en cada momento. Esto es especialmente relevante en las redes móviles, en las que la disponibilidad de los servicios y dispositivos no está garantizada.

## Mejorar el soporte para la movilidad de componentes

La creación de sistemas ubicuos en entornos de IAm implica en última instancia que los componentes puedan trasladar su funcionalidad de un entorno a otro siguiendo a los usuarios. Este tipo de movilidad puede involucrar muchos aspectos: trasladar la implementación de un componente a otra plataforma, trasladar su estado, adaptar el componente a los recursos funcionales disponibles en la nueva plataforma o consideraciones de seguridad y privacidad. La implementación realizada de la arquitectura HI3 facilita la adaptación de un mismo componente funcional a distintos entornos y escenarios, pero no incluye mecanismos concretos para la movilidad física de dichos componentes y de sus estado de ejecución. En este sentido queda abierta la posibilidad de explorar distintas alternativas para mejorar las capacidades relacionadas con la movilidad. Una aproximación podría consistir en trasladar la implementación empaquetada de un servicio, a través de la comunicación entre instancias de contenedores Vineyard de la arquitectura HI3. Esta opción requiere que ambas instancias sean compatibles a nivel de implementación o bien que un servicio tenga empaquetadas diversas implementaciones. Otra aproximación podría consistir en trasladar información del estado de ejecución de un servicio de una plataforma a otra, esperando que en la nueva plataforma exista un servicio que ofrezca la misma funcionalidad y sea capaz de utilizar dicha información de estado.

# Apéndice A

# Especificaciones para un middleware HI3-compatible

En este apéndice se incluyen las especificaciones necesarias para realizar nuevas implementaciones de referencia del middleware orientado a servicios de la arquitectura HI3. Igualmente, estas especificaciones pueden ser utilizadas para la implementación de un servicio BSDM aislado (ejecutado al margen de una arquitectura HI3) o para la implementación de cualquier componente software que desee interaccionar con un servicio HI3 utilizando el modelo de servicios BSDM y el lenguaje BSDL. Así, a lo largo del apéndice se incluye la especificación del lenguaje BSDL, la especificación del modelo BSDM, la especificación para la implementación de nuevos groundings BSDM y la especificación del protocolo para la interacción con un registro de servicios HI3.

## A.1. Especificación de la sintaxis del lenguaje BSDL

En esta sección se describe formalmente la sintaxis del lenguaje BSDL utilizando la notación ABNF (del inglés, Augmented Backus–Naur Form), que es habitualmente utilizada para la descripción de sistemas y lenguajes computacionales.

A pesar de que en el lenguaje BSDL todavía no se han desarrollado las construcciones basadas en reglas, se incluyen los conceptos básicos de éstas en la especificación para mayor claridad. También se incluye una metadescripción inicial con equivalencia a conceptos manejados por lenguajes clásicos como RDF, que definen sus axiomas en base a triples en forma de Sujeto-Predicado-Objeto.

Listado A.1: Sintaxis del lenguaje BSDL en notación ABNF.

```
AXIOM = SUBJECT / PREDICATE / OBJECT

SUBJECT = SemanticAxiom / RuleBodyElement

PREDICATE = Property / PropertyValue / Rule

OBJECT = SemanticAxiom / RuleHead

SemanticAxiom = Concept / Instance

Rule = RuleHead RuleBody

RuleBody = *RuleBodyElement


Concept = *Import URI *Superconcept [Ontology_Ref] *Property OntologyLanguage
      NaturalLanguage

Instance = InstanceLiteral / InstanceURI / InstanceObject / InstanceList /
      InstanceLiteralList

InstanceList = *Instance

InstanceLiteral = URI Literal

InstanceLiteralList = *InstanceLiteral

InstanceURI = URI URIValue

URIValue = URI

InstanceObject = *Import Concept_Ref [URI] [Ontology_Ref] *PropertyValue

Ontology_Ref = URI

OntologyLanguage = URI

NaturalLanguage = URI

Superconcept = Concept / URI

Property = Concept_Ref Multiplicity Concept_Ref PropertyName

Multiplicity = Min Max

Min = Literal

Max = Literal

PropertyName = Literal
```

Import = "Prefix To Dialect Language"

Prefix = Literal

To = URI

Of = URI

Dialect = URI

Lang = URI

Concept_Ref = Concept / URI

PropertyValue = Instance_Ref PropertyName **{**Literal / Instance_Ref**}**

Instance_Ref = Instance / URI

Literal = "Simple datatype"

## A.2. Esquema del lenguaje BSDL

En esta sección se incluye la especificación del lenguaje BSDL en base a su esquema en XML (ver Listado A.2) y a los conceptos de base que proporciona el lenguaje (ver Listado A.3).

Listado A.2: Esquema XML del lenguaje BSDL.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">

<!-- ************ simple and reusable elements ************ -->

<xs:element name="literal">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:enumeration value="string"/>
         <xs:enumeration value="integar"/>
         <xs:enumeration value="decimal"/>
      </xs:restriction>
   </xs:simpleType>
</xs:element>

<xs:element name="multiplicity">
   <xs:complexType>
      <xs:attribute name="min" type="xs:string"/>
      <xs:attribute name="max" type="xs:string"/>
   </xs:complexType>
</xs:element>
```

```
<xs:element name="import">
   <xs:complexType>
      <xs:attribute name="prefix" type="xs:string" use="required"/>
      <xs:attribute name="to" type="xs:anyURI" use="required"/>

      <xs:attribute name="dialect" type="xs:anyURI"/>
      <xs:attribute name="lang" type="xs:anyURI"/>
   </xs:complexType>
</xs:element>

<!-- ************** complex elements: concept ************** -->

<xs:element name="concept">
   <xs:complexType>

      <xs:sequence>
         <xs:element ref="import" minOccurs="0" maxOccurs="unbounded"/>

         <xs:element name="subconceptOf" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
               <xs:choice>
                  <xs:element name="URI" type="xs:anyURI"/>
                  <xs:element ref="concept" />
               </xs:choice>
            </xs:complexType>
         </xs:element>

         <xs:element name="property" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>

      <xs:attribute name="URI" type="xs:anyURI" use="required"/>
      <xs:attribute name="subconceptOf" type="xs:anyURI"/>

      <xs:attribute name="ontology" type="xs:anyURI"/>

      <xs:attribute name="ontologyLanguage" type="xs:anyURI"/>
      <xs:attribute name="naturalLanguage" type="xs:string"/>

   </xs:complexType>
</xs:element>

<xs:element name="property">
   <xs:complexType mixed="true">

      <xs:sequence>
         <xs:element ref="concept" minOccurs="0"/>
         <xs:element ref="multiplicity" minOccurs="0" maxOccurs="1"/>
         <xs:element ref="property" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>

      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="URI" type="xs:anyURI"/>
```

```
    </xs:complexType>
</xs:element>


<!-- ************ complex elements: instance ************ -->

<xs:element name="instance">
    <xs:complexType>

        <xs:sequence>
            <xs:element ref="import" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="with" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>

        <xs:attribute name="of" type="xs:string" use="required"/>
        <xs:attribute name="URI" type="xs:anyURI"/>

        <xs:attribute name="ontology" type="xs:anyURI"/>

    </xs:complexType>
</xs:element>

<xs:element name="with" >
    <xs:complexType mixed="true">

        <xs:choice minOccurs="0">
            <xs:element name="URI" type="xs:anyURI"/>
            <xs:element ref="concept" minOccurs="0" maxOccurs="1"/>
            <xs:element ref="instance" minOccurs="0" maxOccurs="1"/>
            <xs:element ref="literal" minOccurs="0" maxOccurs="1"/>
            <xs:element ref="with" minOccurs="0" maxOccurs="unbounded"/>
        </xs:choice>

        <xs:attribute name="property" type="xs:string" use="required"/>
        <xs:attribute name="valueURI" type="xs:string"/>
        <xs:attribute name="value" type="xs:string"/>
        <xs:attribute name="ofURI" type="xs:anyURI"/>

        <xs:attribute name="ontology" type="xs:anyURI"/>

    </xs:complexType>
</xs:element>


<!-- *********** complex elements: ontology ************ -->

<xs:element name="ontology">
    <xs:complexType>

        <xs:sequence>
            <xs:element ref="concept" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="instance" minOccurs="0" maxOccurs="unbounded"/>
```

```
        </xs:sequence>

        <xs:attribute name="URI" type="xs:anyURI" use="required"/>

        <xs:attribute name="ontologyLanguage" type="xs:anyURI"/>
        <xs:attribute name="naturalLanguage" type="xs:string"/>
        <xs:attribute name="versionNumber" type="xs:string"/>

    </xs:complexType>
</xs:element>

</xs:schema>
```

Listado A.3: Conceptos básicos del lenguaje BSDL.

```
<!-- ********** bsdlOntology ********** -->

<ontology URI="http://hi3project.com/broccoli/bsdl#bsdlOntology"
    ontologyLanguage="hi3project.com/broccoli/bsdl#ontology"
    versionNumber="0.5.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="bsdlSchema.xsd"/>

<!-- ********** object ********** -->

<concept URI="http://hi3project.com/broccoli/bsdl#object"
    ontology="http://hi3project.com/broccoli/bsdl#bsdlOntology"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="bsdlSchema.xsd">
</concept>

<!-- ********** semanticIdentifier ********** -->

<concept URI="http://hi3project.com/broccoli/bsdl#semanticIdentifier"
    subconceptOf="http://hi3project.com/broccoli/bsdl#object"
    ontology="http://hi3project.com/broccoli/bsdl#bsdlOntology"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="bsdlSchema.xsd">
</concept>

<!-- ********** literal ********** -->

<concept URI="http://hi3project.com/broccoli/bsdl#literal"
    subconceptOf="http://hi3project.com/broccoli/bsdl#object"
    ontology="http://hi3project.com/broccoli/bsdl#bsdlOntology"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="bsdlSchema.xsd">
</concept>
```

## A.3. Especificación del modelo de servicios BSDM

El modelo de descripción de servicios BSDM se especifica de forma completa a través de una ontología (bsdmOntology) en lenguaje BSDL, que se encuentra publicada en https://github.com/GII/broccoli/specification/bsdmOntology.xml. Además, en esta sección, en el Listado A.4, se incluye la especificación formal en notación ABNF correspondiente a dicho modelo.

Listado A.4: Especificación del modelo BSDM en notación ABNF.

```
IServiceDescription = IProfile 1*IServiceGrounding *IServiceImplementation

Profile = 1*ServiceFunctionality *NonFunctionalProperty ServiceType

ServiceGrounding = *FunctionalityGrounding

ServiceImplementation = 1*FunctionalityImplementation

ServiceFunctionality = AdvertisedFunctionality / AdvertisedMultiResultFunctionality /
    AdvertisedSubscription / ComposedFunctionality

NonFunctionalProperty = QoSProperty / SecurityProperty / ReferenceToNFP

ReferenceToNFP = SemanticIdentifier

ServiceType = SemanticIdentifier

ComposedFunctionality = *ServiceFunctionality

AdvertisedFunctionality = Functionality

AdvertisedMultiResultFunctionality = Functionality

AdvertisedSubscription = Functionality

Functionality = FunctionalityCategory *NonFunctionalProperty *Output *Input *Effect

Output = Parameter

Input = Parameter

Parameter = Name SemanticIdentifier

Name = Literal

ServiceGrounding = GroundingType OntologyLanguage 1*FunctionalityGrounding

GroundingType = Literal

OntologyLanguage = Literal
```

ServiceImplementation = 1∗FunctionalityImplementation

RequestedFunctionality = Functionality

FunctionalitySearchResult = ServiceFunctionality FunctionalitySearchEvaluation

## A.4. Especificación de un grounding basado en mensajería asíncrona para servicios BSDM

En esta sección se incluye la especificación de un grounding basado en mensajería asíncrona que soporta el modelo de interacción propuesto para los servicios HI3 descritos según el modelo BSDM.

El protocolo incluye dos tipos de mensajes: FunctionalityExecutionMessage y FunctionalityResultMessage.

El formato escogido para la serialización de los mensajes, en un formato multiplataforma soportado por los protocolos de comunicación más extendidos, es JSON.

En el Listado A.5 se especifican en notación ABNF, la estructura de los mensajes del protocolo.

Listado A.5: Especificación de los mensajes del grounding.

FunctionalityExecutionMessage =
  MessageType FunctionalityName **[**ClientName**]** **[**ConversationId**]** ParametersList

ParametersList = InstanceList

InstanceList = ∗Instance

Instance = "BSDL Instance element"

MessageType = Literal

FunctionalityName = Literal

ClientName = Literal

ConversationId = Literal

FunctionalityResultMessage =
  MessageType FunctionalityName **[**ClientName**]** **[**ConversationId**]** ResultsList

ResultsList = InstanceList

El protocolo sigue la siguiente dinámica:

1. Desde el grounding en el lado del cliente se envía un FunctionalityExecMessage a un servicio como mensaje de solicitud de ejecución de una funcionalidad.

2. El servicio puede responder al cliente con uno o varios FunctionalityResultMessage dependiendo del modo de interacción de la funcionalidad invocada, que será uno de los siguientes:

   *a*) Ejecución síncrona de una funcionalidad puntual que produce una única respuesta como resultado.

   *b*) Ejecución asíncrona de una funcionalidad puntual que produce una única respuesta como resultado.

   *c*) Ejecución asíncrona de una funcionalidad durable que produce un número indeterminado de respuestas en el tiempo como resultado.

   *d*) Suscripción a una funcionalidad que puede producir un número indeterminado de notificaciones como respuesta en un tiempo indeterminado.

La comunicación según este protocolo debe tener en cuenta las siguientes particularidades:

- El campo ClientName de los mensajes es opcional. Si se indica en un FunctionalityExecMessage, el mensaje FunctionalityResultMessage correspondiente debe contener el mismo valor para ClientName. El caso típico en el que puede no especificarse es el de una suscripción a una funcionalidad si no se desea filtrar las notificaciones por cliente, de modo que todos los clientes suscritos reciben copias de los mismos mensajes de notificación.
- El campo ConversationId es opcional. Si se indica en un FunctionalityExecMessage, el mensaje FunctionalityResultMessage correspondiente debe contener el mismo valor. Si no se indica y un mismo cliente envía distintos FunctionalityExecMessage al mismo servicio usando el mismo FunctionalityName, el cliente no dispondrá de ningún mecanismo para distinguir qué FunctionalityResultMessage corresponde a cada FunctionalityExecMessage enviado.
- Para el establecimiento de la comunicación, el cliente debe conocer un grounding definido por el servicio cuya funcionalidad desea invocar. En la descripción BSDM del grounding se especifica la URI de conexión como única información específica necesaria.

Las implementaciones actuales de referencia de contenedores de servicios de la arquitectura HI3 proporcionan las siguientes implementaciones de groundings concretos:

- Realización del grounding basado en mensajería asíncrona sobre los siguientes protocolos de comunicación existentes: OpenWire, Stomp y Stomp sobre Websockets.
- Una implementación del serializador de mensajes JSON. Los mensajes se serializan en texto plano en formato JSON, donde cada campo del mensaje es serializado como un valor en un hashmap. El primer campo de todos ha de ser el tipo del mensaje.

## A.5. Especificación del protocolo para la interacción con un registro de servicios

En esta sección se incluye la especificación del protocolo estándar definido por la arquitectura HI3 para la interacción con un registro de servicios, tanto a la hora de realizar búsquedas de funcionalidades sobre el mismo como a la hora de añadir nuevos descriptores de servicios.

Se definen dos tipos de mensajes para una interacción de búsqueda: SearchFunctionalityRequestMessage y SearchFunctionalityResultMessage.

Se define un único tipo de mensaje para añadir un nuevo descriptor a un registro de servicios: RegisterServiceRequestMessage.

En el Listado A.6 se especifican en notación ABNF, la estructura de los mensajes del protocolo.

Listado A.6: Especificación de los mensajes de interacción con un registro de servicios.

```
SearchFunctionalityRequestMessage = Message_Type ClientName [ConversationId]
    RequestedFunctionalityList

RequestedFunctionalityList = InstanceList

InstanceList = *Instance

Instance = "BSDL Instance element"

SearchFunctionalityResultMessage = Message_Type ClientName [ConversationId]
    ServicesData OntologiesData SearchResultList

ServicesData = DescriptorDataList

OntologiesData = DescriptorDataList

DescriptorDataList = *DescriptorData

DescriptorData = DescriptorName DescriptorContents
```

DescriptorName = literal

DescriptorContents = "literal that contains a whole ontology following BSDL syntax"

SearchResultsList = ∗FunctionalitySearchResult

RegisterServiceRequestMessage = Message–Type **[**ClientName**]** **[**ConversationId**]**
ServicesData OntologiesData

Una interacción de búsqueda remota en un servicio a través del protocolo sigue la siguiente dinámica:

1. Un cliente envía una solicitud SearchFunctionalityRequestMessage a un contenedor, que contiene un objetivo de búsqueda.

2. Si el contenedor que recibe el mensaje está federado con otros contenedores reenvía el mensaje SearchFunctionalityRequestMessage a los mismos. El proceso se repite recursivamente para todos los contenedores federados. Se admite la definición de un número máximo de saltos entre contenedores así como un tiempo máximo de espera por respuesta.

3. Cada contenedor consultado construye un mensaje de respuesta de tipo FunctionalitySearchResult que engloba todos los resultados devueltos por su registro y lo envía como respuesta.

4. El contenedor que recibió originalmente el mensaje SearchFunctionalityRequestMessage envía una respuesta al cliente, en un mensaje FunctionalitySearchResult, que agrupa los resultados recibidos de todos los contenedores federados.

La comunicación según este protocolo debe tener en cuenta las siguientes particularidades:

- El campo ConversationId es opcional. Pero si se emplea, el contenedor debe usar el valor recibido en un SearchFunctionalityRequestMessage para su FunctionalitySearchResultMessage correspondiente.
- El valor del campo ClientName recibido en un SearchFunctionalityRequestMessage debe ser usado para su FunctionalitySearchResultMessage correspondiente.
- Para cada SearchFunctionalityRequestMessage, se envía únicamente un mensaje FunctionalitySearchResultMessage de respuesta que agrupa todos los resultados.

- Si hay contenedores federados con un contenedor que ha recibido un SearchFunctionalityRequestMessage, se compone un SearchFunctionalityRequestMessage nuevo que copia todos los campos del mensaje recibido, con las siguientes excepciones: i) el ClientName debe tener el valor que identifica al contenedor; ii) el ConversationId debe componerse del ClientName y ConversationId del mensaje recibido.

Una interacción para añadir un nuevo descriptor de servicio a un registro remoto a través del protocolo tiene las siguientes particularidades:

- Un contenedor envía un mensaje RegisterServiceRequestMessage a otro contenedor y no necesita recibir una respuesta.
- Los campos ClientName y ConversationId son opcionales.
- Los campos ServicesData y OntologiesData deben contener respectivamente los descriptores del servicio y de las ontologías referenciadas por éste.

Los mensajes se serializan en texto plano en formato JSON. Cada campo del mensaje es serializado como un valor en un hashmap. El primer campo de todos ha de ser el tipo del mensaje.

Para el establecimiento de la conexión que permite federar a dos contenedores, éstos deben tener conocimiento de sus URI de conexión respectivas y de los protocolos concretos soportados. En las realizaciones concretas de contenedores Vineyard se soporta JMS-OpenWire, STOMP y STOMP sobre WebSockets.

# Referencias


[Aarts and Diederiks, 2007] Aarts, E. and Diederiks, E. (2007). Ambient lifestyle: From concept to experience. *BIS Publishers Amsterdam*.

[Aarts and Encarnaçao, 2006] Aarts, E. and Encarnaçao, J. (2006). True visions: The emergence of ambient intelligence. *Springer*.

[Aarts et al., 2001] Aarts, E., Harwig, R., and Schuumans, M. (2001). The invisible future, the seamless integration of technology into everyday life. *Ambient Intelligence*, pages 235–250.

[Aarts et al., 2007] Aarts, E., Markopoulos, P., and Ruyter, B. D. (2007). The persuasiveness of ambient intelligence. *Intelligence*, V:367–381.

[Aarts and Ruyter, 2009] Aarts, E. and Ruyter, B. D. (2009). New research perspectives on Ambient Intelligence. *Journal of Ambient Intelligence and Smart Environments*, 1(1):5–14.

[Aberer et al., 2007] Aberer, K., Hauswirth, M., and Salehi, A. (2007). Infrastructure for data processing in large-scale interconnected sensor networks. In *Proceedings - IEEE International Conference on Mobile Data Management*, pages 198–205.

[Abowd, 1999] Abowd, G. D. (1999). Classroom 2000: An experiment with the instrumentation of a living educational environment. *IBM Systems Journal, 38(4):508–530*.

[Abowd and Mynatt, 2004] Abowd, G. D. and Mynatt, E. D. (2004). Designing for the human experience in smart environments. *Smart Environments: Technology, Protocols, and Applications, pages 153–174*.

[ActiveMQ, 2015] ActiveMQ (2015). Apache ActiveMQ. URL: http://activemq.apache.org/.

[Adelstein, 2004] Adelstein, F. (2004). Fundamentals of mobile and pervasive computing. *Odyssey*.

[Adler and Davis, 2004] Adler, A. and Davis, R. (2004). Speech and sketching for multimodal design. *In Proceedings of the 9th International Conference on Intelligent User Interfaces, pages 214–216*.

[AIRE, 2003] AIRE (2003). MIT's AIRE Group projects. URL: http://aire.csail.mit.edu/.

[Antoniou and Van Harmelen, 2004] Antoniou, G. and Van Harmelen, F. (2004). OWL Web Ontology Language. *Ubiquity*, 2007(September):1–1.

[ApacheShiro, 2015] ApacheShiro (2015). Apache Shiro Java security framework. URL: http://shiro.apache.org/.

[Arsanjani et al., 2009] Arsanjani, A., Booch, G., Erl, T., Josuttis, N., Chappell, D., De Vadoss, J., Erl, T., Josuttis, N., Krafzig, D., Little, M., and et al. (2009). SOA Manifesto. URL: http://www.soa-manifesto.org/.

[Ashton, 2009] Ashton, K. (2009). That 'Internet of Things' Thing. *RFiD Journal*.

[Augustin et al., 2002] Augustin, I., Yamin, A., Barbosa, J., and Resin Geyer, C. (2002). *Towards Taxonomy for Mobile Applications with Adaptive Behavior*, pages 224–228. ACTA Press.

[Augusto, 2007] Augusto, J. C. (2007). Ambient intelligence: The confluence of ubiquitous/pervasive computing and artificial intelligence. In *Intelligent Computing Everywhere*, pages 213–234. Springer London.

[Augusto and Shapiro, 2007] Augusto, J. C. and Shapiro, D. (2007). *Advances in Ambient Intelligence*. IOS Press.

[Autili et al., 2014] Autili, M., Caporuscio, M., Issarny, V., and Berardinelli, L. (2014). Model-driven engineering of middleware-based ubiquitous services. *Software and Systems Modeling*, 13(2):481–511.

[Avilés-López and García-Macías, 2009] Avilés-López, E. and García-Macías, J. A. (2009). TinySOA: A service-oriented architecture for wireless sensor networks. *Service Oriented Computing and Applications*, 3(2):99–108.

[Baldauf and Dustdar, 2007] Baldauf, M. and Dustdar, S. (2007). A survey on context-aware systems. *International Journal of Ad Hoc*, 2(4).

[Banavar et al., 2000] Banavar, G., Beck, J., Gluzberg, E., Munson, J., Sussman, J., and Zukowski, D. (2000). *Challenges: an application model for pervasive computing*, pages 266–274. ACM.

[Baresi et al., 2012] Baresi, L., Georgantas, N., Hamann, K., Issarny, V., Lamersdorf, W., Metzger, A., and Pernici, B. (2012). Emerging research themes in services-oriented systems. In *Annual SRII Global Conference, SR*, pages 333–342.

[Bass et al., 2003] Bass, L., Clements, P., and Kazman, R. (2003). *Software Architecture in Practice*, volume 2nd of *Series in Software Engineering*. Addison-Wesley Professional.

[BeagleBoneBlack, 2015] BeagleBoneBlack (2015). BeagleBone Black platform. URL: http://beagleboard.org/black.

[Becha and Amyot, 2012] Becha, H. and Amyot, D. (2012). Non-functional properties in service oriented architecture - A consumer's perspective. *Journal of Software*, 7(3):575–587.

[Becker, 2008] Becker, M. (2008). Software Architecture Trends and Promising Technology for Ambient Assisted Living Systems. In *Assisted Living Systems-Models, Architectures and Engineering Approaches*, volume 7462.

[Ben Mokhtar et al., 2007] Ben Mokhtar, S., Georgantas, N., and Issarny, V. (2007). COCOA: COnversation-based Service COmposition in pervAsive Computing Environments with QoS Support. *Journal of Systems and Software*, 80(12):1941–1955.

[Ben Mokhtar et al., 2010] Ben Mokhtar, S., Raverdy, P.-G., Urbieta, A., and Cardoso, R. S. (2010). Interoperable Semantic and Syntactic Service Discovery for Ambient Computing Environments. *International Journal of Ambient Computing and Intelligence*, 2(4):13–32.

[Bencomo et al., 2013] Bencomo, N., Bennaceur, A., Grace, P., Blair, G., and Issarny, V. (2013). The role of models@run.time in supporting on-the-fly interoperability. *Computing*, 99(3):167–190.

[Bennaceur et al., 2010a] Bennaceur, A., Blair, G., Chauvel, F., Gang, H., Georgantas, N., Grace, P., Howar, F., Inverardi, P., Issarny, V., Paolucci, M., and Others (2010a). Towards an architecture for runtime interoperability. *Leveraging Applications of Formal Methods, Verification, and Validation*, pages 206–220.

[Bennaceur et al., 2010b] Bennaceur, A., Blair, G., Georgantas, N., Grace, P., and Inverardi, P. (2010b). Revisiting the Middleware Paradigm: On-the-fly Interoperability in Highly Complex Distributed Systems. Technical report.

[BerkeleyDB, 2015] BerkeleyDB (2015). Berkeley DB: embedded key-value database library. URL: http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/.

[Bermúdez Pestonit et al., 2014] Bermúdez Pestonit, P., Raño Jares, I., Mosquera Collazo, B., Paz Lopez, A., Varela Fernadez, G., Faiña Rodrigez-Vila, A., Duro Fernandez, R., and Lopez Peña, F. (2014). Sistema y método de tele-asistencia mediante televisor inteligente. P201431701. SCIO Innovation Technologies.

[Bernardin et al., 2009] Bernardin, K., Ekenel, H. K., and Stiefelhagen, R. (2009). Multimodal identity tracking in a smart room. *Personal and Ubiquitous Computing*, 13(1):25–31.

[Bobick et al., 1999] Bobick, A. F., Intille, S. S., Davis, J. W., Baird, F., Pinhanez, C. S., Campbell, L. W., Ivanov, Y. A., Schutte, A., and Wilson, A. (1999). The KidsRoom: A Perceptually-Based Interactive and Immersive Story Environment. *Presence Teleoperators Virtual Environments*, 8(4):369–393.

[Bonino and Corno, 2008] Bonino, D. and Corno, F. (2008). DogOnt - Ontology Modeling for Intelligent Domotic Environments. *The Semantic Web-ISWC 2008*, pages 790–803.

[Bonino and Corno, 2010] Bonino, D. and Corno, F. (2010). Rule-based intelligence for domotic environments. *Automation in Construction*, 19(2):183–196.

[Booth et al., 2004] Booth, D., Haas, H., Mccabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. (2004). Web Services Architecture.

[Bosse et al., 2013] Bosse, T., Cook, D. J., Neerincx, M., and Sadri, F., editors (2013). *Human Aspects in Ambient Intelligence: Contemporary Challenges and Solutions*, volume 8 of *Atlantis Ambient and Pervasive Intelligence*. Atlantis Press, Paris.

[Brdiczka et al., 2009] Brdiczka, O., Crowley, J. L., and Reignier, P. (2009). Learning situation models in a smart home. *IEEE transactions on systems man and cybernetics Part B Cybernetics a publication of the IEEE Systems Man and Cybernetics Society*, 39(1):56–63.

[Bronsted et al., 2010] Bronsted, J., Hansen, K. M., Ingstrup, M., Bronsted, J., Hansen, K. M., and Ingstrup, M. (2010). Service Composition Issues in Pervasive Computing. *IEEE Pervasive Computing*, 9(1):62–70.

[Cabri et al., 2008] Cabri, G., Mola, F. D., Ferrari, L., Leonardi, L., and R (2008). The LAICA Project: An ad-hoc middleware to support Ambient Intelligence. *Multiagent and Grid Systems*, 4:235–247.

[Cahill et al., 2003] Cahill, V., Gray, E., Seigneur, J. M., Jensen, C., Chen, Y., Shand, B., Dimmock, N., Twigg, A., Bacon, J., English, C., and et al. (2003). Using trust for secure collaboration in uncertain environments. *Ieee Pervasive Computing*, 2(3):52–61.

[Chakraborty et al., 2006] Chakraborty, D., Joshi, A., Yesha, Y., and Finin, T. (2006). Toward distributed service discovery in pervasive computing environments. *IEEE Transactions on Mobile Computing*, 5(2):97–112.

[Chan and Lyu, 2008] Chan, P. P. W. and Lyu, M. R. (2008). Dynamic Web Service Composition: A New Approach in Building Reliable Web Service. In *22nd International Conference on Advanced Information Networking and Applications aina 2008*, pages 20–25. Ieee.

[Chung and Leite, 2009] Chung, L. and Leite, J. D. P. (2009). On Non-Functional Requirements in Software Engineering. *Conceptual modeling: Foundations and ...*, pages 363–379.

[Coen et al., 1999] Coen, M., Phillips, B., Warshawsky, N., Weisman, L., Peters, S., and Finin, P. (1999). Meeting the Computational Needs of Intelligent Environments: The Metaglue System. In *Proceedings of MANSE99*, volume Dublin, Ir.

[Compton et al., 2012] Compton, M., Barnaghi, P., Bermudez, L., García-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K., Kelsey, W. D., Le Phuoc, D., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K., Passant, A., Sheth, A., and Taylor, K. (2012). The SSN ontology of the W3C semantic sensor network incubator group. *Journal of Web Semantics*, 17:25–32.

[Cook, 2009] Cook, D. (2009). Multi-agent smart environments. *Journal of Ambient Intelligence and Smart Environments*, 1(1):51–55.

[Corchado et al., 2008] Corchado, J. M., Bajo, J., and Abraham, A. (2008). GerAmi: Improving Healthcare Delivery in Geriatric Residences. *IEEE Intelligent Systems*, 23(2):19–25.

[Coronato and De Pietro, 2010] Coronato, A. and De Pietro, G. (2010). Formal Design of Ambient Intelligence Applications. *Computer*, 43(12):60–68.

[Coutaz et al., 2005] Coutaz, J., Crowley, J., and Dobson, S. (2005). Context is key. *Communications of the*, 48(3):49–53.

[Curry, 2004] Curry, E. (2004). Message-Oriented Middleware. *Middleware for Communications*, pages 1–28.

[Da Costa et al., 2008] Da Costa, C., Yamin, A., and Geyer, C. (2008). Toward a general software infrastructure for ubiquitous computing. *IEEE Pervasive Computing*.

[De et al., 2012] De, S., Elsaleh, T., Barnaghi, P., and Meissner, S. (2012). An internet of things platform for real-world and digital objects. *Scalable Computing*, 13(1):45–57.

[Dean and Ghemawat, 2008] Dean, J. and Ghemawat, S. (2008). MapReduce : Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):1–13.

[Dey, 2001] Dey, A. K. (2001). Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1):4–7.

[Di Nitto et al., 2008] Di Nitto, E., Ghezzi, C., Metzger, A., Papazoglou, M., and Pohl, K. (2008). A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering*, 15(3-4):313–341.

[Donoho et al., 2008] Donoho, A., Costa-requena, J., Mcgee, T., Messer, A., Fiddian-green, A., and Fuller, J. (2008). UPnP™ Device Architecture 1.1. URL: http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf.

[Dourish et al., 2004] Dourish, P., Grinter, R. E., Delgado De La Flor, J., and Joseph, M. (2004). Security in the wild: user strategies for managing security as an everyday, practical problem. *Personal and Ubiquitous Computing*, 8(6):391–401.

[Ducatel et al., 2003] Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., and Burgelman, J. (2003). Ambient intelligence: From vision to reality. *IST Advisory Group Draft Report to the European Commission*, pages 45–68.

[Ducatel et al., 2001] Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., and Burgelman, J.-C. (2001). *ISTAG Scenarios for Ambient Intelligence in 2010*. IPTS (JRC).

[Dutta, 2008] Dutta, B. (2008). Semantic Web Services: A Study of Existing Technologies, Tools and Projects. *DESIDOC Journal of Library & Information Technology*, 28(3):47–55.

[Euzenat and Shvaiko, 2013] Euzenat, J. and Shvaiko, P. (2013). *Ontology Matching*. Springer-Verlag, second edi edition.

[Ferraiolo et al., 2003] Ferraiolo, D. F., Kuhn, D. R., and Chandramouli, R. (2003). *Role-Based Access Control*, volume 2002.

[Finin et al., 1994] Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). KQML as an agent communication language. In *Proceedings of the third international conference on Information and knowledge management - CIKM '94*, pages 456–463. ACM Press.

[FIPA, 2015] FIPA (2015). Foundation for Intelligent Physical Agents. URL: http://www.fipa.org.

[FP6-IST Amigo, 2008a] FP6-IST Amigo (2008a). Amigo Project: Ambient intelligence for the networked home environment. URL: https://gforge.inria.fr/projects/amigo/.

[FP6-IST Amigo, 2008b] FP6-IST Amigo (2008b). Amigo Project: Final Report. URL: http://cordis.europa.eu/documents/documentlibrary/115484471EN6.pdf.

[FP6-IST PERSONA, 2010] FP6-IST PERSONA (2010). PERSONA: PERceptive Spaces prOmoting iNdependent Aging. URL: http://cordis.europa.eu/project/rcn/80532_en.html.

[FutureInternet, 2015] FutureInternet (2015). European Future Internet portal. URL: http://www.future-internet.eu.

[Gao et al., 2006] Gao, Z., Wang, L., Yang, X., and Wen, D. (2006). PCPGSD: An enhanced GSD service discovery protocol for MANETs. *Computer Communications*, 29(12):2433–2445.

[García-Sánchez et al., 2009] García-Sánchez, F., Valencia-García, R., Martínez-Béjar, R., and Fernández-Breis, J. T. (2009). An ontology, intelligent agent-based framework for the provision of semantic web services. *Expert Systems with Applications*, 36:3167–3187.

[Garlan et al., 2002] Garlan, D., Siewiorek, D. P., Smailagic, A., and Steenkiste, P. (2002). Project aura: toward distraction-free pervasive computing. *Ieee Pervasive Computing*, 1(2):22–31.

[Georgantas et al., 2010] Georgantas, N., Issarny, V., Mokhtar, S., Bromberg, Y., Bianco, S., Thomson, G., Raverdy, P., Urbieta, A., and Cardoso, R. (2010). Middleware Architecture for Ambient Intelligence in the Networked Home. *Handbook of Ambient Intelligence and Smart Environments*, pages 1139–1169.

[Georgantas et al., 2005] Georgantas, N., Mokhtar, S. B., Bromberg, Y., Issarny, V., Kalaoja, J., Kantarovitch, J., Gerodolle, A., and Mevissen, R. (2005). The Amigo Service Architecture for the Open Networked Home Environment. *5th Working IEEEIFIP Conference on Software Architecture WICSA05*, 2005:295–296.

[Girolami et al., 2008] Girolami, M., Lenzi, S., Furfari, F., and Chessa, S. (2008). SAIL: A Sensor Abstraction and Integration Layer for Context Awareness. *2008 34th Euromicro Conference Software Engineering and Advanced Applications*, pages 374–381.

[Goertzel and Wang, 2006] Goertzel, B. and Wang, P. (2006). Advances in Artificial General Intelligence : Concepts , Architectures and Algorithms Frontiers in Artificial Intelligence and Applications. *Knowledge and Information Systems*.

[Grace et al., 2003] Grace, P., Blair, G., and Samuel, S. (2003). Remmoc: A reflective middleware to support mobile client interoperability. In Meersman, R., Tari, Z., and Schmidt, D., editors, *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, volume 2888 of *Lecture Notes in Computer Science*, pages 1170–1187. Springer Berlin Heidelberg.

[Grace et al., 2011] Grace, P., Georgantas, N., Bennaceur, A., Blair, G., Chauvel, F., Issarny, V., Paolucci, M., Saadi, R., Souville, B., and Sykes, D. (2011). The connect architecture. *Formal Methods for Eternal Networked Software Systems*, pages 27–52.

[Guinard et al., 2011] Guinard, D., Trifa, V., Mattern, F., and Wilde, E. (2011). From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices. In *Architecting the Internet of Things*, pages 97–129.

[Guo et al., 2013] Guo, B., Zhang, D., Wang, Z., Yu, Z., and Zhou, X. (2013). Opportunistic IoT: Exploring the harmonious interaction between human and the internet of things. *Journal of Network and Computer Applications*, 36(6):1531–1539.

[Guttman et al., 1999] Guttman, E., Perkins, C., Veizades, J., and Day, M. (1999). RFC2608: Service Location Protocol, Version 2. *Internet RFCs*.

[Hellenschmidt and Kirste, 2004] Hellenschmidt, M. and Kirste, T. (2004). SODAPOP: a software infrastructure supporting self-organization in intelligent environments. In *2nd IEEE International Conference on Industrial Informatics 2004 INDIN 04 2004*, number 26, pages 479–486. 2nd IEEE International Conference on Industrial Informatics (INDIN '04), Ieee.

[Hohpe and Woolf, 2003] Hohpe, G. and Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*.

[Hornecker, 2005] Hornecker, E. (2005). *A design theme for tangible interaction: embodied facilitation*, page 23–43. Number September. Springer.

[IntelEdison, 2015] IntelEdison (2015). Plataforma Intel Edison. URL: http://www.intel.es/content/www/es/es/do-it-yourself/edison.html.

[Izso, 2009] Izso, L. (2009). Appropriate dynamic lighting as a possible basis for a smart ambient lighting 1 introduction : The concept of the aladin project. *Access*, pages 67–74. Web: http://www.ambient-lighting.eu/.

[JADE, 2015] JADE (2015). Java Agent DEvelopment Framework. URL: http://jade.tilab.com.

[Janse and Vink, 2008] Janse, M. and Vink, P. (2008). Amigo Architecture: Service Oriented Architecture for Intelligent Future In-Home Networks. *Constructing Ambient Intelligence*, pages 2–3.

[Janse et al., 2008] Janse, M., Vink, P., and Georgantas, N. (2008). Amigo Architecture : Service Oriented Architecture for Intelligent Future In-Home Networks. *Architecture*, pages 371–378.

[Jetty Project, 2015] Jetty Project (2015). Jetty web server. URL: http://www.eclipse.org/jetty/.

[JMS, 2015] JMS (2015). JMS: Java Message Service. URL: http://docs.oracle.com/javaee/6/tutorial/doc/bncdq.html.

[Jordan and Alves, 2007] Jordan, D. and Alves, A. (2007). Web Services Business Process Execution Language Version 2 . 0. *Language*, 11(April):1–264.

[JSON, 2015] JSON (2015). JSON: Java Script Object Notation. URL: http://json.org/.

[Karaenke et al., 2012] Karaenke, P., Schuele, M., Micsik, A., and Kipp, A. (2012). Inter-organizational interoperability through integration of multiagent, web service, and semantic web technologies. In *Lecture Notes in Business Information Processing*, volume 98 LNBIP, pages 55–75.

[Kassim et al., 2007] Kassim, A., Esfandiari, B., Majumdar, S., and Serghi, L. (2007). A Flexible Hybrid Architecture for Management of Distributed Web Service Registries. *Communications Networks and Services Research Conference*.

[Kavantzas, 2005] Kavantzas, N. (2005). Web Services Choreography Description Language Version 1.0. URL: http://www.w3.org/TR/ws-cdl-10/.

[Kindberg and Fox, 2002] Kindberg, T. and Fox, A. (2002). System software for ubiquitous computing. *Ieee Pervasive Computing*, 1(1):70–81.

[Klein et al., 2005] Klein, M., Ries, B. K., and Mussig, M. (2005). What is needed for semantic service descriptions? A proposal for suitable language constructs. *International Journal of Web and Grid Services*, 1(3/4):328.

[Klein et al., 2007] Klein, M., Schmidt, A., and Lauer, R. (2007). Ontology-Centred Design of an Ambient Middleware for Assisted Living: The Case of SOPRANO. *Context*.

[Kopecky et al., 2007] Kopecky, J., Vitvar, T., Bournez, C., and Farrell, J. (2007). SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Computing*, 11(6):60–67.

[Krumm et al., 2000] Krumm, J., Harris, S., Meyers, B., Brumitt, B., Hale, M., and Shafer, S. (2000). Multi-camera multi-person tracking for EasyLiving. *Proceedings Third IEEE International Workshop on Visual Surveillance*, 6:3–10.

[Küster et al., 2007] Küster, U., König-Ries, B., Stern, M., and Klein, M. (2007). DIANE: an integrated approach to automated service discovery, matchmaking and composition. *Distribution*, pages 1033–1042.

[Lane et al., 2008] Lane, N. D., Eisenman, S. B., Musolesi, M., Miluzzo, E., and Campbell, A. T. (2008). Urban Sensing: Opportunistic or Participatory? In *HotMobile '08, Proceedings of the 9th workshop on Mobile computing systems and applications, February 26-26, 2008, Napa Valley, CA, USA*, pages 11–16.

[Lanthaler et al., 2010] Lanthaler, M., Granitzer, M., and Gütl, C. (2010). Semantic Web Services: State of the Art. In Kommers, P., Issa, T., and Isaías, P., editors, *Proceedings of the IADIS International Conference on Internet Technologies Society ITS 2010*, pages 107–114. IADIS.

[Larson, 2007] Larson, C. (2007). In elder care, signing on becomes a way to drop by. *New York Times, February 4, 2007*.

[Lazovik et al., 2006] Lazovik, A., Aiello, M., and Papazoglou, M. (2006). Planning and monitoring the execution of web service requests. *International Journal on Digital Libraries*, 6(3):235–246.

[LevelDB, 2015] LevelDB (2015). LevelDB: light-weight, single-purpose library for persistence. URL: http://leveldb.org/.

[Leyton-brown, 2010] Leyton-brown, K. (2010). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*.

[LibreriaUniDA, 2015] LibreriaUniDA (2015). Libreria UniDA: un framework para el acceso uniforme a dispositivos. URL: https://github.com/GII/UniDA/.

[Lohmann, 2010] Lohmann, N. (2010). *Correctness of Services and Their Composition*. PhD thesis, Universität Rostock.

[López-Sanz et al., 2008] López-Sanz, M., Acuña, C. J., Cuesta, C. E., and Marcos, E. (2008). Modelling of Service-Oriented Architectures with UML. *Electronic Notes in Theoretical Computer Science*, 194(4):23–37.

[Lu et al., 2010] Lu, H., Lane, N. D., Eisenman, S. B., and Campbell, A. T. (2010). Bubble-sensing: Binding sensing tasks to the physical world. *Pervasive and Mobile Computing*, 6(1):58–71.

[Luo et al., 2006] Luo, J., Montrose, B., Kim, A., Khashnobish, A., and Kang, M. (2006). Adding OWL-S support to the existing UDDI infrastructure. In *Proceedings - ICWS 2006: 2006 IEEE International Conference on Web Services*, pages 153–160.

[Martin et al., 2007] Martin, D., Burstein, M., McDermott, D., McIlraith, S., Paolucci, M., Sycara, K., McGuinness, D. L., Sirin, E., and Srinivasan, N. (2007). Bringing Semantics to Web Services with OWL-S. *World Wide Web Internet And Web Information Systems*, 10(3):243–277.

[Mell and Grance, 2011] Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology. *Nist Special Publication*, 145:7.

[Mietz et al., 2013] Mietz, R., Groppe, S., Romer, K., and Pfisterer, D. (2013). Semantic Models for Scalable Search in the Internet of Things. *Journal of Sensor and Actuator Networks*, 2:172–195.

[Miorandi et al., 2012] Miorandi, D., Sicari, S., De Pellegrini, F., and Chlamtac, I. (2012). Internet of things: Vision, applications and research challenges.

[Mitton et al., 2012] Mitton, N., Papavassiliou, S., Puliafito, A., and Trivedi, K. S. (2012). Combining Cloud and sensors in a smart city environment.

[Mokhtar et al., 2006] Mokhtar, S., Georgantas, N., and Issarny, V. (2006). COCOA: Conversation-Based Service Composition for Pervasive Computing Environments. In *ACS/IEEE International Conference on Pervasive Services*, pages 29–38. Ieee.

[Motorola, 2003] Motorola (2003). SPI block guide v03.06. URL: http://www.ee.nmt.edu/ teare/ee308l/datasheets/S12SPIV3.pdf.

[MyUI, 2015] MyUI (2015). Mainstreaming Accessibility through Synergistic User Modelling and Adaptability. URL: http://www.myui.eu/.

[Nagarajan et al., 2006] Nagarajan, M., Verma, K., Sheth, A. P., Miller, J., and Lathem, J. (2006). Semantic Interoperability of Web Services - Challenges and Experiences. *Evaluation*, 0:373–382.

[Neuman and Ts'o, 1994] Neuman, B. C. and Ts'o, T. (1994). Kerberos: An Authentication Service for Computer Networks. *IEEE Communications Magazine*, 32(9):33–38.

[Niemelä and Latvakoski, 2004] Niemelä, E. and Latvakoski, J. (2004). Survey of requirements and solutions for ubiquitous software. *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia MUM 04*, (October):71–78.

[OASIS, 2015] OASIS (2015). OASIS Web Services Quality Model. URL: http://www.oasis-open.org/committees/wsqm.

[OSGi Alliance, 2015] OSGi Alliance (2015). Open services gateway initiative. URL: http://www.osgi.org.

[OWLS-MX, 2008] OWLS-MX (2008). OWLS-MX matchmaker. URL: http://projects.semwebcentral.org/projects/owls-mx/.

[OWLS-SLR, 2008] OWLS-SLR (2008). OWLS-SLR matchmaker. URL: http://lpis.csd.auth.gr/systems/OWLS-SLR/.

[OXYGEN, 2005] OXYGEN (2005). The Oxygen Project. URL: http://www.oxygen.lcs.mit.edu.

[Padovitz et al., 2008] Padovitz, A., Loke, S. W., and Zaslavsky, A. (2008). Multiple-Agent Perspectives in Reasoning About Situations for Context-Aware Pervasive Computing Systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 38(4):729–742.

[Pan et al., 2010] Pan, G., Xu, Y., Wu, Z., Yang, L., Lin, M., and Li, S. (2010). Task Follow-me: Towards Seamless Task Migration Across Smart Environments. *IEEE Intelligent Systems*.

[Papazoglou and Georgakopoulos, 2003] Papazoglou, M. P. and Georgakopoulos, D. (2003). Service-Oriented Computing. *Communications of the ACM*, 46(10):24–28.

[Papazoglou et al., 2008] Papazoglou, M. P., Traverso, P., Dustdar, S., and Leymann, F. (2008). Service-Oriented Computing: a Research Roadmap. *International Journal of Cooperative Information Systems*, 17(02):223.

[Paz Lopez et al., 2012] Paz Lopez, A., Varela Fernadez, G., Vazquez Rodriguez, S., Faiña Rodrigez-Vila, A., Becerra Permuy, J. A., Deibe Diaz, A., Duro Fernandez, R., and Lopez Peña, F. (2012). Terminal instrumentation node for a self-configuring and a secure building automation system. Patent numbers: ES2400893, EP2592810, US2014148952, JP2014514885. MyTech Ingeniería Aplicada S.L.

[Penserini et al., 2010] Penserini, L., Kuflik, T., and Busetta, P. (2010). Agent-based organizational structures for ambient intelligence scenarios. *Journal of Ambient Intelligence*, 2:409–433.

[Peters et al., 2003] Peters, S., Look, G., Quigley, K., and Shrobe, H. (2003). Hyperglue: Designing High-Level Agent Communication for Distributed Applications. *Originally submitted to*.

[Peters, 2006] Peters, S. L. (2006). *Hyperglue: An Infrastructure for Human-Centered Computing in Distributed, Pervasive, Intelligent Environments*. PhD thesis, Massachusetts Institute of Technology.

[PhoneGap, 2015] PhoneGap (2015). PhoneGap framework. URL: http://phonegap.com/.

[Pistore et al., 2005] Pistore, M., Traverso, P., Bertoli, P., and Marconi, A. (2005). Automated synthesis of executable web service compositions from BPEL4WS

processes. *Special interest tracks and posters of the 14th international conference on World Wide Web WWW 05*, page 1186.

[Pokorny, 2013] Pokorny, J. (2013). NoSQL databases: a step to database scalability in web environment. *International Journal of Web Information Systems*, 9(1):69–82.

[Preuveneers and Novais, 2012] Preuveneers, D. and Novais, P. (2012). A survey of software engineering best practices for the development of smart applications in Ambient Intelligence. *Journal of Ambient Intelligence and Smart Environments*, 4(3):149–162.

[Prud'hommeaux and Seaborne, 2008] Prud'hommeaux, E. and Seaborne, A. (2008). SPARQL Query Language for RDF.

[Qu et al., 2006] Qu, Y., Hu, W., and Cheng, G. (2006). Constructing virtual documents for ontology matching. In *Proceedings of the 15th international conference on World Wide Web - WWW '06*, page 23, New York, New York, USA. ACM Press.

[Quitadamo et al., 2007] Quitadamo, R., Zambonelli, F., and Cabri, G. (2007). The Service Ecosystem: Dynamic Self-Aggregation of Pervasive Communication Services. *First International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments (SEPCASE '07)*, pages 1–1.

[Rakotonirainy and Tay, 2004] Rakotonirainy, A. and Tay, R. (2004). In-vehicle ambient intelligent transport systems (I-VAITS): towards an integrated research. *Proceedings The 7th International IEEE Conference on Intelligent Transportation Systems IEEE Cat No04TH8749*, pages 648–651.

[Rana et al., 2010] Rana, R. K., Chou, C. T., Kanhere, S. S., Bulusu, N., and Hu, W. (2010). Ear-Phone : An End-to-End Participatory Urban Noise Mapping System. In *Proceedings of the International Conference on Information Processing in Sensor Networks IPSN*, pages 105–116.

[Ranganathan and Campbell, 2003] Ranganathan, A. and Campbell, R. (2003). A middleware for context-aware agents in ubiquitous computing environments. In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, pages 143–161. Springer-Verlag New York, Inc. New York, NY, USA.

[Raverdy et al., 2006] Raverdy, P. G., Issarny, V., Chibout, R., and De La Chapelle, A. (2006). A multi-protocol approach to service discovery and access in pervasive environments. In *2006 3rd Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, MobiQuitous*.

[RDF, 2015] RDF (2015). Resource Description Framework. URL: http://www.w3.org/RDF/.

[Redis, 2015] Redis (2015). Redis key-value cache and store. URL: http://redis.io/.

[Reichle et al., 2008] Reichle, R., Wagner, M., Khan, M., Geihs, K., Lorenzo, J., Valla, M., Fra, C., Paspallis, N., and Papadopoulos, G. (2008). A comprehensive context modeling framework for pervasive computing systems. In *Distributed applications and interoperable systems*, pages 281–295. Springer.

[Ricci et al., 2007] Ricci, A., Buda, C., and Zaghini, N. (2007). An agent-oriented programming model for SOA & web services. In *IEEE International Conference on Industrial Informatics (INDIN)*, volume 2, pages 1059–1064.

[Richardson and Ruby, 2007] Richardson, L. and Ruby, S. (2007). *RESTful Web Services*, volume 12.

[RMI, 2015] RMI (2015). Java Remote Method Invocation (RMI). URL: http://download.oracle.com/javase/tutorial/rmi/index.html.

[Robinson et al., 2005] Robinson, P., Vogt, H., and Wagealla, W. (2005). *Some research challenges in pervasive computing*, volume 780, pages 1–16. Springer Verlag.

[Roman et al., 2006] Roman, D., de Bruijn, J., Mocan, A., Lausen, H., Domingue, J., Bussler, C., and Fensel, D. (2006). Www: Wsmo, wsml, and wsmx in a nutshell. In Mizoguchi, R., Shi, Z., and Giunchiglia, F., editors, *The Semantic Web – ASWC 2006*, volume 4185 of *Lecture Notes in Computer Science*, pages 516–522. Springer Berlin Heidelberg.

[Rosenthal and Stanford, 2000] Rosenthal, L. and Stanford, V. (2000). Nist smart space: pervasive computing initiative. *Proceedings of the 9th IEEE international workshops on enabling technologies: infrastructure for collaborative enterprises WETICE*, pages 06–11.

[Rui et al., 2009] Rui, C., Yi-bin, H., Zhang-qin, H., and Jian, H. (2009). Modeling the Ambient Intelligence Application System: Concept, Software, Data, and Network. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 39(3):299–314.

[Saha and Mukherjee, 2003] Saha, D. and Mukherjee, A. (2003). Pervasive computing: A paradigm for the 21st century. *Computer*, 36(3):25–31.

[Saif et al., 2003] Saif, U., Pham, H., Paluska, J., and Waterman, J. (2003). A case for goal-oriented programming semantics. pages 1–8.

[Schmidt and Schmidt, 2006] Schmidt, A. and Schmidt, A. (2006). Ontology-based user context management: The challenges of imperfection and time-dependence. *Lecture Notes in Computer Science*, 4275:995.

[Schmidt et al., 2011] Schmidt, L., Mitton, N., Simplot-Ryl, D., Dagher, R., and Quilez, R. (2011). DHT-based distributed ALE engine in RFID middleware. In *2011 IEEE International Conference on RFID-Technologies and Applications, RFID-TA 2011*, pages 319–326.

[Schmidt et al., 2005] Schmidt, M.-T., Hutchison, B., Lambros, P., and Phippen, R. (2005). The Enterprise Service Bus: Making service-oriented architecture real.

[Secker et al., 2003] Secker, J., Hill, R., Villeneau, L., and Parkman, S. (2003). Promoting independence: but promoting what and how?

[Seibel, 2005] Seibel, P. (2005). Beyond exception handling: Conditions and restarts. In *Practical Common Lisp*, pages 233–243. Apress.

[Shafiq et al., 2006] Shafiq, M. O., Ding, Y., and Fensel, D. (2006). Bridging Multi Agent Systems and Web Services: towards interoperability between Software Agents and Semantic Web Services. In *10th IEEE International Enterprise Distributed Object Computing Conference*, pages 85–96. IEEE.

[Shi et al., 2003] Shi, Y., Xie, W., Xu, G., and Shi, R. (2003). The smart classroom: Merging technologies for seamless tele-education. *IEEE Pervasive Computing, April-June:47–55*.

[Silva et al., 2008] Silva, E., Pires, F., and Sinderen, M. V. (2008). Dynamic Service Composition: Why, Where and How. *Communication*.

[SimpleDB, 2015] SimpleDB (2015). Amazon SimpleDB. URL: http://aws.amazon.com/es/simpledb/.

[Sivashanmugam et al., 2003] Sivashanmugam, K., Sivashanmugam, K., Verma, K., Verma, K., Sheth, A., Sheth, A., Miller, J., and Miller, J. (2003). Adding Semantics to Web Services Standards. In *Proceedings of the International Conference on Web Services*, pages 395–401.

[Sixsmith et al., 2009] Sixsmith, A. J., Müller, S., Klein, M., Bierhoff, I., Berlo, A. V., Delaney, S., Avatangelou, E., and Arnaudov, V. (2009). SOPRANO: An ambient assisted living system for supporting older people at home. *Gerontechnology*, 8(3):186–186.

[SOAP, 2007] SOAP (2007). SOAP Version 1.2 (W3C Recommendation. URL: http://www.w3.org/TR/2007/REC-soap12-part0-20070427/.

[Soldatos et al., 2006] Soldatos, J., Dimakis, N., Stamatis, K., and Polymenakos, L. (2006). A breadboard architecture for pervasive context-aware services in smart spaces: middleware components and prototype applications. *Personal and Ubiquitous Computing*, 11(3):193–212.

[Soldatos et al., 2007] Soldatos, J., Pandis, I., Stamatis, K., Polymenakos, L., and Crowley, J. (2007). Agent based middleware infrastructure for autonomous context-aware ubiquitous computing services. *Computer Communications*, 30(3):577–591.

[Stanford, 2005] Stanford, V. (2005). Infrastructure for distributed and embedded systems and interfaces. *In Proceedings of the Embedded Systems, Ambient Intelligence and Smart Surroundings Conference*.

[Stoettinger, 2004] Stoettinger, M. (2004). *Context-Awareness in industrial environments*. PhD thesis, FH Hagenberg.

[Streitz et al., 2007] Streitz, N. A., Kameas, A., and Mavrommati, I. (2007). The disappearing computer. *Lecture Notes in Computer Science, vol 4500, Springer*.

[Studer et al., 2007] Studer, R., Grimm, S., and Abecker, A. (2007). *Semantic Web Services, Concepts, Technologies, and Applications*.

[SWRL, 2004] SWRL (2004). SWRL: A Semantic Web Rule Language Combining OWL and RuleML. URL: http://www.w3.org/Submission/SWRL/.

[Tapia et al., 2009a] Tapia, D. I., Abraham, A., Corchado, J. M., and Alonso, R. S. (2009a). Agents and ambient intelligence: case studies. *Journal of Ambient Intelligence and Humanized Computing*, 1(2):85–93.

[Tapia et al., 2009b] Tapia, D. I., Rodríguez, S., Bajo, J., and Corchado, J. M. (2009b). FUSION@, A SOA-based multi-agent architecture. *Advances in Soft Computing*, 50:99–107.

[Tapia et al., 2004] Tapia, E. M., Intille, S. S., and Larson, K. (2004). Activity Recognition in the Home Using Simple and Ubiquitous Sensors. *Pervasive Computing*, 3001:158–175.

[Teixeira et al., 2013] Teixeira, T., Hachem, S., Issarny, V., and Nikolaos Georgantas (2013). Service oriented middleware for the Internet of Things. In *Proceedings of the 4th European Conference on Towards a Service-Based Internet*, volume 6994, pages 220–229.

[Thomson et al., 2008] Thomson, G., Sacchetti, D., Bromberg, Y. D., Parra, J., Georgantas, N., and Issarny, V. (2008). Amigo Interoperability Framework: Dynamically Integrating Heterogeneous Devices and Services. *Constructing Ambient Intelligence*, 11:421–425.

[TIAF, 2015] TIAF (2015). Dandelion TIAF: Threefold Interaction Abstraction Framework. URL: https://github.com/GII/Dandelion/.

[Tokuda et al., 2006] Tokuda, H., Edwards, W., and Ramachandran, U. (2006). A Bridging Framework for Universal Interoperability in Pervasive Systems. *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, pages 3–3.

[UDDI, 2006] UDDI (2006). Introduction to UDDI: Important features and functional concepts. URL: http://uddi.xml.org/files/uddi-tech-wp.pdf.

[UsiXML, 2015] UsiXML (2015). USer Interface eXtended Markup Language. URL: http://www.usixml.org/.

[Varela, 2015] Varela, G. (2015). Autonomous adaptation of user interfaces during application mobility processes in ambient intelligence scenarios. Master's thesis, Universidade da Coruña, Ferrol, Spain.

[Varela et al., 2012] Varela, G., Paz-Lopez, A., Becerra, J. A., and Duro, R. J. (2012). Dandelion: Decoupled distributed user interfaces in the hi3 ambient intelligence platform. In Bravo, J., López-de Ipiña, D., and Moya, F., editors, *Ubiquitous Computing and Ambient Intelligence*, volume 7656 of *Lecture Notes in Computer Science*, pages 161–164. Springer Berlin Heidelberg.

[Varela et al., 2011] Varela, G., Paz-Lopez, A., Becerra, J. A., Vazquez-Rodriguez, S., and Duro, R. J. (2011). UniDA: Uniform device access framework for human interaction environments. *Sensors*, 11(10):9361–9392.

[W3C, 2004] W3C (2004). OWL-S: Semantic Markup for Web Services. URL: http://www.w3.org/Submission/OWL-S/.

[W3C, 2007] W3C (2007). Semantic Annotations for WSDL and XML Schema (SAWSDL). URL: http://www.w3.org/TR/sawsdl/.

[W3C, 2015a] W3C (2015a). W3C Semantic Web initiative. URL: http://www.w3.org/2001/sw/.

[W3C, 2015b] W3C (2015b). W3C Web Services. URL: http://www.w3.org/2002/ws/.

[W3C, 2015c] W3C (2015c). Web Ontology Language (OWL). URL: http://www.w3.org/2004/OWL/.

[Waibel et al., 2004] Waibel, A., Steusloff, H., and Waibel, A. (2004). *Computers in the Human Interaction Loop*. Springer.

[Want and Pering, 2005] Want, R. and Pering, T. (2005). *System challenges for ubiquitous and pervasive computing*, pages 9–14. ACM Press.

[WebSockets, 2015] WebSockets (2015). WebSockets protocol. URL: http://www.websocket.org/aboutwebsocket.html.

[Weiser, 1991] Weiser, M. (1991). The Computer in the 21st Century. *Scientific American*, 265(3):94–104.

[Weiser, 1993] Weiser, M. (1993). Hot topics: Ubiquitous computing. *IEEE Computer 26(10):71–72*.

[Wolf et al., 2008] Wolf, P., Schmidt, A., and Klein, M. (2008). SOPRANO - An extensible, open AAL platform for elderly people based on semantical contracts. In *Proceedings of 3rd Workshop on Artificial Intelligence Techniques for Ambient Intelligence AITAmI'08 18th European Conference on Artificial Intelligence ECAI 08*, number Ecai 08. Citeseer.

[Wooldridge, 2002] Wooldridge, M. (2002). Intelligent agents: The key concepts. *MultiAgent Systems and Applications II*, 2322:3–43.

[Wright et al., 2008] Wright, D., Gutwirth, S., Friedewald, M., De Hert, P., Gutwirth, S., Schreurs, W., Moscibroda, A., Friedewald, M., Lindner, R., Wright, D., and et al. (2008). *Safeguards in a World of Ambient Intelligence*, volume 1. Springer.

[WS-Discovery, 2009] WS-Discovery (2009). Web Services Dynamic Discovery (WS-Discovery) Version 1.1. URL: http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.html.

[Wu et al., 2011] Wu, G., Talwar, S., Johnsson, K., Himayat, N., and Johnson, K. D. (2011). M2M: From mobile to embedded internet. *IEEE Communications Magazine*, 49(4):36–43.

[XACML, 2013] XACML (2013). eXtensible Access Control Markup Language (XACML). URL: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

[Yu and Su, 2009] Yu, L. and Su, S. (2009). Adopting context awareness in service composition. In *Proceedings of the First AsiaPacific Symposium on Internetware*, Internetware '09, pages 1–10. ACM.

[Zelkha, 1998] Zelkha, E. (1998). The future of information appliances and consumer devices. *Palo Alto Ventures, Palo Alto, California, (unpublished document)*.

[Zhou and Riekki, 2010] Zhou, J. and Riekki, J. (2010). Context-Aware Pervasive Service Composition. *2010 International Conference on Intelligent Systems Modelling and Simulation*, pages 437–442.

# Índice de figuras

# Índice de listados de código