Dª. MARÍA CELESTE CAMPO VÁZQUEZ, con D. N. I. 33332772-E

AUTORIZA:

A que su tesis doctoral con el título: ***"Tecnologías Middleware para el Desarrollo de Servicios en Entorno de Computación Ubicua"*** pueda ser utilizada para fines de investigación por parte de la Universidad Carlos III de Madrid.

Leganés, 7 de mayo de 2004

Fdo.: María Celeste Campo Vázquez

DEPARTAMENTO DE INGENIERÍA TELEMÁTICA

Escuela Politécnica Superior

Universidad Carlos III de Madrid

TESIS DOCTORAL

TECNOLOGÍAS MIDDLEWARE PARA EL DESARROLLO DE SERVICIOS EN ENTORNOS DE COMPUTACIÓN UBICUA

Autora: Mª **Celeste Campo Vázquez**
Ingeniero de Telecomunicación

Director: **Andrés Marín López**
Doctor Ingeniero de Telecomunicación

2004

Tribunal nombrado por el Mgfco. y Excmo. Sr. Rector de la Universidad Carlos III de Madrid, el día 25 de FEBRERO de 2004.

Presidente CARLOS DELGADO KLOOS

Vocal GUSTAVO ALONSO GARCÍA

Vocal TOMAS DE MIGUEL MORO

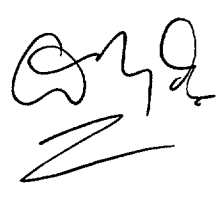
Vocal MIGUEL SORIANO IBÁÑEZ

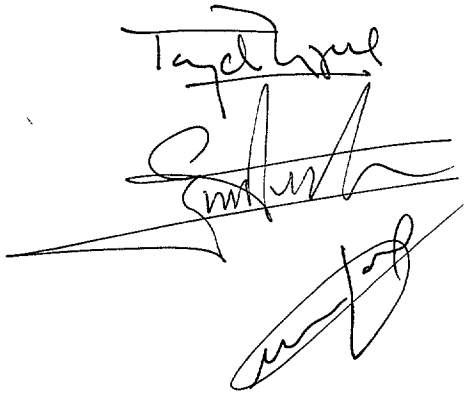
Secretario NATIVIDAD MARTINEZ MADRID

Realizado el acto de defensa y lectura de la Tesis el día 7 de Mayo de 2004 en Leganés.

Calificación: SOBRESALIENTE CUM LAUDE POR UNANIMIDAD

EL PRESIDENTE | EL SECRETARIO | LOS VOCALES





A mi abuelo Juan.

Any sufficiently advanced technology is indistinguishable from magic
Arthur C. Clarke

Agradecimientos

Son muchas las personas que han hecho posible esta tesis doctoral, y por ello quiero dejar constancia de mi agradecimiento a todos ellos en esta primera página de la memoria, que sin embargo es la última que escribo.

En primer lugar, quiero agradecer esta tesis doctoral a mi director Andrés Marín por su apoyo y consejos durante la realización de este trabajo y por sus ánimos en los momentos en los que más los necesité.

A mis compañeros del Departamento de Ingeniería Telemática: a Carlos Delgado Kloos por darme la oportunidad de formar parte de él; a mis compañeros de investigación, y sobre todo amigos: Andrés Marín, Carlos García Rubio, Florina Almenárez, Guillermo Díez-Andino, Mario Muñoz y Rosa M. García por escuchar y contribuir a las diferentes propuestas e ideas que finalmente dieron como resultado esta tesis doctoral. Quiero destacar de manera especial la ayuda de Carlos García Rubio en las simulaciones con MODSIM, y la de Florina Almenarez por nuestras productivas charlas sobre descubrimiento y seguridad que concluyeron con la definición de la versión segura de PDP.

Al comité técnico FIPA Ad-Hoc, de manera especial a Michael Berger por apoyar algunas de mis propuestas.

A la Cátedra Nokia Universidad Carlos III de Madrid por haber apoyado económicamente parte de los trabajos realizados en el marco de esta tesis doctoral.

A Arturo y a Nacho por ser los mejores compañeros de trabajo que he tenido y por todo lo que me enseñaron cuando empecé a trabajar aquí. Gracias también a M.Carmen, por ser sobre todo mi amiga y ahora compañera de despacho. Muchas gracias también a Goyo, Jesús, Luis, Liliana, Vicente, Rafael y Michael.

Un agradecimiento muy especial a Javier Martínez Pereira, por ser uno de mis mejores amigos y por haber tenido la suerte de conocerte.

A mis proyectistas durante estos años: a Guillermo, Rosa y Miguel por contribuir a que los agentes móviles en dispositivos limitados sean una realidad; a José Carlos, por su implementación del PDP en J2ME; a Jonatan, por descu-

brirme un nuevo camino en el que trabajar para hacerle la vida más facil a los invidentes y a Lucía, Sandra y Miguel Ángel por su trabajo con tarjetas inteligentes.

En el plano personal, quiero agradecer y dedicar esta tesis doctoral a mi familia, de manera especial a mis padres, Luis y Montse, porque si he llegado hasta aquí ha sido gracias a ellos y porque siempre me han apoyado y animado a seguir con mis estudios a pesar de que esto suponga estar lejos de casa. A mi hermana Mónica, por ser durante estos años uno de mis mejores apoyos. A mi abuela, por preocuparse siempre tanto por mi.

Tampoco me puedo olvidar de mis amigos: Ruth, Sonia, Souto y Sinuhé (y a las nuevas incorporaciones Javier y Eva) por seguir unidos a pesar de la distancia. A Rebeca, por animarme a empezar en esta universidad. A Rosa, por compartir conmigo piso y obras estos últimos tres años. A Martín, Mar, Merche, Jaime, Romina y a nuestra vecina Lolín por esos dos años tan diferentes en Maiquez 20. A Javier, Susana y Vincent por ser mis primeros amigos en Madrid. A M.Luz, Javier y M.Luz Congosto por seguir pendientes de mi después de mi paso por Teléfonica I+D.

Por último, mi agradecimiento a Carlos por su apoyo y comprensión durante la realización de esta tesis, por creer en mi y por darme la confianza que yo nunca he tenido en lo que hago y sobre todo, por todas las horas que esta tesis nos ha separado, espero poder compensártelas a partir de ahora.

Graciñas a todos.

Resumen

Weiser, en su artículo "The Computer for the 21st Century", describe entornos saturados de elementos con capacidades de cómputo y comunicación, totalmente integrados en nuestras vidas y que nos proporcionan información asociada a nuestras necesidades y al entorno en el que nos encontramos en cada momento, de forma transparente. A esta nueva era de la computación imaginada por Weiser se denomina computación ubicua, y en nuestros días podemos decir que comienza a ser una realidad. Esto es así, gracias fundamentalmente a los avances realizados en la microelectrónica, que permiten introducir capacidad de cómputo en un mayor número de dispositivos que se pueden embeber en el mundo físico que nos rodea, y a los avances en el desarrollo de protocolos inalámbricos, que dotan a estos dispositivos de capacidad de comunicación sin necesidad de cableados. Pero para que realmente la visión de Weiser se haga realidad, es necesario realizar un gran esfuerzo de investigación, centrado en aportar soluciones para que la tecnología software nos dé el soporte adecuado para desarrollar servicios en estos entornos.

En esta tesis doctoral realizamos contribuciones en el campo de la definición de tecnologías middleware para el desarrollo de servicios en entornos de computación ubicua. En primer lugar, abordamos el problema del descubrimiento de servicios, que permitirá que de forma automática un dispositivo descubra los servicios ofrecidos por otros dispositivos que le rodean. Aunque existen propuestas en este sentido, desde nuestro punto de vista no cubren todas las necesidades que imponen estos nuevos entornos de computación, por lo que hemos definido un nuevo mecanismo de descubrimiento: Pervasive Discovery Protocol (PDP). PDP es un protocolo de ámbito local, totalmente distribuido en el que tanto las peticiones como las respuestas se envían por multicast, cada dispositivo almacena en una caché local los anuncios recibidos, cuyo contenido comparte con los dispositivos que le rodean. PDP consigue minimizar el número de mensajes transmitidos por búsqueda, manteniendo tasas de descubrimiento de servicios altas, además permite que los dispositivos con mayor tiempo de disponibilidad transmitan un mayor número de respuestas, minimizando el consumo energético de los más limitados.

En segundo lugar, proponemos el uso de la tecnología de agentes móviles co-

mo middleware para el desarrollo de servicios en entornos ubicuos. Los agentes se caracterizan porque su comportamiento se orienta a alcanzar ciertas metas, por ser autónomos, por su capacidad de cooperar con otros sistemas y si poseen la característica de movilidad, por ser capaces de moverse a sistemas remotos para realizar sus tareas, y de esta forma minimizar el coste de las comunicaciones. Estas características se adaptan a las restricciones que impone la computación ubicua. En esta línea, contribuimos a la adaptación del estándar de agentes FIPA para su funcionamiento en estos entornos, y en concreto, nos centramos en el servicio de páginas amarillas, denominado Directory Facilitator (DF). Nuestra solución consiste en definir un nuevo agente, Service Discovery Agent (SDA), que utiliza una implementación subyacente del protocolo de servicios PDP para realizar búsquedas eficientes de servicios remotos, eliminando el mecanismo de federación de DFs definido por FIPA. Estas propuestas nos han llevado a participar de forma activa en el comité técnico FIPA Ad-Hoc.

Abstract

Weiser, in his paper "The Computer for the 21st Century", describes environments saturated with devices, with computing and communication capacities, fully integrated in our lives, and that give us information related with our needs and with our surroundings, everywhere, every time, and in a transparent way. This new age of computing, imagined by Weiser, is known as pervasive computing, and it is now beginning to become a reality thanks to the recent advances in microelectronics, that make possible to introduce computing power in smaller devices embedded in the physical world, and to the recent advances in wireless communications, that allow devices to communicate without wires. For pervasive computing to become a full reality, further research is necessary, specially in the field of developing software technology able to support the deployment of services in these environments.

In this Ph.D. dissertation, we contribute to the definition of middleware technologies for the development of services in pervasive computing environments. First, we broach the problem of service discovery, that allow devices to automatically discover the services offered by other devices in their surroundings. Although other proposals exist, we think none of them fulfils the needs of these new environments, so we have defined a new discovery mechanism: the Pervasive Discovery Protocol (PDP). PDP is a new protocol designed for local scopes, fully distributed, where requests and replys are both multicasted, and where each device stores in a local cache the advertisements listened so far and shares this information with the devices arround it. PDP manages to reduce the number of messages transmited per service request while obtaining high service discovery ratios, and besides it makes devices with greater availability time to answer first, so minimizing the battery drain of the more power-constrained ones.

Secondly, we propose the use of mobile agents technology as the middleware for the development of services in a pervasive computing environment. Agents are characterized by their autonomous and goal-oriented behaviour, their ability to cooperate with other agents and, if mobile, they are able to move to remote systems to carry out their task, so minimizing the communication cost. These

characteristics fit well into the restrictions that pervasive computing impose. We also aim to adapt the FIPA standard to these environments; specifically, we concentrate in the yellow pages service, the so called Directory Facilitator (DF). In our solution, we define a new agent, the Service Discovery Agent (SDA), that uses an underlaying implementation of our service discovery protocol, PDP, to make efficient searches of remote services, removing the mechanism of DFs federation defined by FIPA. These proposals have been presented and discussed in the FIPA Ad-Hoc Technical Commitee, after what we were invited to take part and are now an active member of it.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción y objetivos

Si nos fijamos a nuestro alrededor, la visión futurista que Mark Weiser describió en su artículo "The Computer for the 21st Century" [Weiser, 1991] comienza a ser una realidad. Weiser describe entornos saturados de elementos con capacidades de cómputo y comunicación, totalmente integrados en nuestras vidas y que nos proporcionan información asociada a nuestras necesidades y al entorno en el que nos encontramos en cada momento. En la actualidad, gracias a la evolución de la tecnología hardware, existen un mayor número de dispositivos: teléfonos móviles, PDAs, pagers, que nos acompañan en todo momento debido a su reducido tamaño. Estos dispositivos tienen capacidad de cómputo y además pueden comunicarse con otros elementos sin necesidad de conexiones físicas, gracias al desarrollo y evolución de los protocolos inalámbricos, tanto en redes celulares, GPRS [Andersson, 2001] y UMTS [Kaaranen et al., 2001], como en redes locales, WLAN [Dayem, 1997] y Bluetooth [Bray y Sturman, 2001]. Así, ya es más o menos habitual que estos dispositivos nos proporcionen nuevas aplicaciones además de las que propiamente tenían asociadas, por ejemplo, desde un móvil no sólo mantenemos una conversación telefónica, sino que también podemos ver la información del tiempo, localizar la farmacia más cercana o incluso, algo menos común, pero posible, programar la lavadora [Margherita2000.com, 2000].

La primera aproximación que la tecnología ha dado a la visión de Weiser es lo que se denomina *"anytime, anywhere"*, es decir, ha permitido a los usuarios, a través de sus nuevos dispositivos personales inalámbricos, acceder a las mismas aplicaciones que utilizan habitualmente en sus PCs, pero en cualquier momento y desde cualquier lugar. Así, los usuarios a través de ellos pueden navegar por Internet, WAP [Singhal, 2001] o i-Mode [Frengle, 2001], pueden consultar su correo electrónico, pueden sincronizar la información almacenada entre sus dispositivos, SyncML [Hansmann et al., 2002], para poder gestionar automáticamente

las diversas actualizaciones realizadas en diferentes momentos desde diferentes dispositivos.

Pero la visión de Weiser va mucho más allá, involucra introducir capacidad de cómputo y de comunicación en el mundo físico [Hansmann et al., 2001]: en sensores, que capturen información del entorno o del propio usuario, en actuadores que nos permitan ejecutar acciones de forma remota sobre elementos físicos (puertas, interruptores,...), en electrodomésticos del hogar (lavadoras, neveras,...) o de oficinas (impresoras, faxes, aire acondicionado, proyectores, ...) y en sistemas de transporte (coches, autobuses, ...). Además será preciso que estos dispositivos tengan la capacidad de operar sin infraestructura de red fija, lo que implica no sólo la necesidad de emplear protocolos inalámbricos, sino que también será necesario proporcionales autonomía, es decir, que puedan comunicarse con otros dispositivos directamente sin necesidad de dispositivos intermedios (típicamente PCs) para construir nuevos servicios.

En estos nuevos entornos, los dispositivos personales de los usuarios permitirán que el mundo físico se configure según sus necesidades y preferencias, así no sólo podrá utilizar estos dispositivos como un ordenador portátil sino como un mando a distancia que le permita controlar su entorno físico más próximo. Por ejemplo, podemos pensar que la PDA de un usuario al entrar en una sala le indique al aire acondicionado que la temperatura ideal es 25 grados y que sea el propio aire acondicionado el que interaccione con los sensores de temperatura para que, a partir de sus mediciones, pueda mantener la temperatura indicada.

Para que esta visión se haga realidad es necesario realizar un gran esfuerzo de investigación, centrado en aportar soluciones para que la tecnología software nos dé el soporte adecuado para desarrollar aplicaciones en lo que se denominan entornos de computación móvil y ubicua. Las nuevas restricciones y características que imponen estos entornos, nos lleva a cambiar los conceptos tradicionales que se han aplicado en entornos de redes fijas en las que los nodos eran computadores. Parece claro que no es eficiente migrar soluciones tradicionales a estos nuevos sistemas [Banavar et al., 2000], existe una mayor diversidad de dispositivos, con diferentes limitaciones hardware para ejecutar aplicaciones y en los que hay que tener en cuenta que las comunicaciones pueden ser costosas y las distancias mucho más significativas que en las redes tradicionales. Además, la mayoría de estos nuevos dispositivos no son multipropósito, sino que proporcionan un serie de servicios concretos, pero de manera espontánea, cuando entran en contacto unos con otros a través de algún protocolo de comunicación, es posible que se compongan estos servicios de forma inteligente, para ofrecer un nuevo servicio que satisfaga las expectativas del usuario de manera transparente al mismo, como quería Weiser.

1.1. Motivación de la tesis

Como hemos introducido en la sección anterior, los grandes avances que se han realizado en el campo de la microelectrónica, permitiendo embeber microprocesadores en cada vez más elementos del mundo físico, unido al desarrollo y esfuerzo de estandarización que se ha realizado en la definición de nuevos protocolos de red, hace posible que los dispositivos que pueden formar parte de un entorno de computación distribuida sean muchos y muy distintos, lo que introduce nuevos retos añadidos a los problemas tradicionales de este paradigma de la computación.

En la literatura [Satyanarayanan, 2001] se habla de la computación móvil y de la computación ubicua como evoluciones de la computación distribuida. Se comenzó a utilizar el término computación móvil a principios de los 90 cuando los dispositivos personales y las redes inalámbricas permitieron la movilidad de los usuarios, así a los problemas tradicionales se añadían los problemas de direccionamiento, de la calidad y capacidad cambiante de las redes de acceso debido al interfaz radio, de las limitaciones de recursos y baterías de los dispositivos finales, etc. Una gran mayoría de las soluciones que se presentaron a estos problemas, se centraban sobre todo en solventarlos de tal manera que las aplicaciones que se desarrollaban y los servicios que se le ofrecían a los usuarios finales, eran iguales que si estuvieran conectados a una red fija.

La computación móvil dio paso a lo que se denomina computación ubicua, cuando los dispositivos con capacidad de computación y comunicación no eran sólo dispositivos personales, sino casi cualquier dispositivo físico que rodea al usuario (los dispositivos del hogar, de la oficina, sensores,...). En este nuevo paradigma se mantienen los retos de la computación móvil, y además se añaden algunos nuevos: la definición de nuevos protocolos de red inalámbrica para la comunicación directa entre dispositivos; la necesidad de que los dispositivos descubran las capacidades de otros, para poder ofrecer servicios más inteligentes y complejos al usuario, sin necesidad de interaccionar continuamente con él; la necesidad de capturar las preferencias del usuario de manera que los servicios se adapten a ellas, etc.

En los siguientes apartados detallaremos más esta evolución, describiendo primero lo que denominamos nuevos actores, que son los nuevos dispositivos que han aparecido estos últimos años y que permiten la movilidad de los usuarios y la ubicuidad de la tecnología. Después describiremos algunos de los nuevos escenarios en los que estos dispositivos proporcionarán al usuario nuevos servicios. Por último, detallaremos los retos que imponen estos escenarios y las respuestas que debe proporcionar la tecnología para que sean una realidad.

1.1.1. Nuevos actores

Desde nuestra visión creemos que los nuevos dispositivos con capacidad de computación y comunicación que han aparecido estos últimos años, nunca serán como ordenadores personales, aunque los desarrollos en microlectrónica hagan viable igualar sus capacidades, ya que la mayoría de ellos están dedicados a una tarea concreta y muy determinada. El usuario a lo que aspira es a que las nuevas tecnologías permitan configurar esas tareas, adaptándolas a sus propias necesidades y que su interacción con ellas sea más fácil y transparente.

Con esta idea en mente, hemos clasificado estos dispositivos en varios grupos y hemos visto qué posibilidades de integración tendrán en un entorno distribuido.

- **Dispositivos personales**

  Englobamos en esta categoría a los nuevos dispositivos personales, como teléfonos móviles y agendas electrónicas, que, aunque en principio fueron diseñados para proporcionar un servicio concreto, telefonía inalámbrica o agenda, la realidad es que estos dispositivos cada vez proporcionan una mayor funcionalidad al usuario y además lo acompañan en todo momento.

  En estos últimos años existe una clara tendencia a incorporar a estos dispositivos capacidad de comunicación inalámbrica, no sólo sistemas de telefonía como GSM o GPRS para conectarse a Internet y obtener servicios tradicionales como navegación Web o correo electrónico, sino también de corto alcance como IrDA, Bluetooth o IEEE 802.11 para que puedan colaborar con los dispositivos más próximos, formando redes ad-hoc. El uso de protocolos inalámbricos facilita también la movilidad de estos dispositivos.

  La característica más importante que tienen estos dispositivos es que son personales, de hecho todos ellos ofrecen un conjunto de aplicaciones denominadas *Personal Information Management* (PIM) y por lo tanto, las aplicaciones que se ejecutan en ellos deben estar personalizadas a las propias preferencias del usuario. Para ello los servicios se configurarán basándose en ellas, empleándose perfiles obtenidos de forma dinámica y transparente (o no) a partir de la interacción que tenga el usuario con el dispositivo y con las aplicaciones que en él residen.

  Las capacidades hardware de estos dispositivos varían enormemente, además la mayoría de ellos tienen un sistema operativo distinto y propietario, por lo que el desarrollo de aplicaciones en un principio estuvo cerrado a los propios fabricantes. En la actualidad podemos decir que existen tres grandes alternativas en el mercado que además han permitido que el desarrollo de aplicaciones se extienda, estos sistemas operativos son: Windows

Mobile, PalmOS y Symbian, [Hansmann et al., 2001], los dos primeros centrados más en el mundo de las agendas personales y el último en el de los teléfonos móviles, aunque todos ellos comienzan a dar soporte a ambas funcionalidades. También existen varias iniciativas para introducir Linux en estos dispositivos y en este sentido, en el año 2003 ya han aparecido productos comerciales con una versión reducida de este sistema operativo.

En general, casi todos estos dispositivos tienen displays con buenas resoluciones que facilitan la visualización de información al usuario. La introducción de datos se realiza mediante teclados tradicionales o reducidos, como los de los teléfonos móviles, pero también se han definido nuevos métodos para facilitar esta tarea al usuario como el método Graffiti para agendas, el T9 para teléfonos móviles, o reconocedores de escritura y de voz. Funcionan todos ellos con baterías.

- **Dispositivos de función específica**

  Englobamos en esta categoría a lo que se denomina tradicionalmente electrónica de consumo, tanto del hogar (lavadoras, aire acondicionado, ...), de oficinas (impresoras, fax, ...) y de entretenimiento (televisión, equipos de audio, ...). Estos dispositivos tienen una función concreta, y el usuario no espera que realicen otro tipo de tareas más que aquella para la que han sido diseñados y comprados. Entonces la pregunta que se nos plantea es: cómo se pueden emplear las capacidades de computación y comunicación de estos dispositivos, posibles respuestas serían obtener un servicio mejor sin interacción directa y constante del usuario, o realizar funciones de mantenimiento que faciliten a los fabricantes dar un mejor producto.

  En general, estos dispositivos son estáticos y con conexión a la red mediante protocolos inalámbricos o cableados, empleando la red eléctrica, la red telefónica o cableados específicos. Sus características hardware varían de unos a otros y los sistemas operativos son propietarios y diseñados a medida, para que se realice correcta y eficientemente la función específica para la que se han construido. El desarrollo de aplicaciones está bastante cerrado, sobre todo porque no se ha tenido en cuenta la posibilidad que plantea el hecho de que estos dispositivos interaccionen con otros para realizar mejor su tarea, por ejemplo, la lavadora con los identificadores del tipo de ropa, el frigorífico con los identificadores de los productos, etc.

  Una característica común a casi todos ellos, es que su interfaz para interaccionar con el usuario está realizada a medida, con displays reducidos y teclas con funciones específicas. Al contrario que en los dispositivos personales, la mayoría de éstos no emplean baterías sino que se alimentan a través de la

red eléctrica.

- **Sensores y actuadores**

  Esta categoría engloba a un gran número de dispositivos que agrupamos en dos tipos, por una parte los sensores, que nos permiten realizar medidas y obtener datos del medio físico que nos rodea, ejemplos de este tipo de dispositivos son un sensor de temperatura o de presión, o una *smart label* [Finkenzeller, 2003] que nos proporciona la identificación de un objeto. El otro tipo es lo que denominamos actuadores, que nos permiten realizar una acción concreta sobre el medio físico, ejemplos de este tipo son un sistema de apertura de puertas, una bombilla, una alarma acústica, etc.

  Estos dispositivos son los más limitados en cuanto a capacidad de almacenamiento y procesamiento, pero son imprescindibles para proporcionarle a los usuarios servicios que tengan en cuenta el espacio físico que le rodea y que le permitan actuar directamente sobre él. Suelen ser dispositivos estáticos. En cuanto a los sensores, la tendencia actual es interconectarlos con protocolos inalámbricos, que faciliten colocarlos en cualquier lugar sin necesidad de cableado. Por este último motivo estos dispositivos suelen funcionar gracias a baterías, la optimización de su uso es más crítico en este caso debido a que cambiarlas o recargarlas no puede estar realizándose continuamente, por lo que el objetivo es conseguir que duren años. Algunos sensores también emplean mecanismos inductivos para su funcionamiento, por ejemplo las *smart labels* transmiten la identificación cuando un sistema lector se aproxima y le suministra la energía necesaria para transmitir esa información. En cuanto a los actuadores, es más habitual que estén conectados a la red eléctrica, y por lo tanto, a veces emplean ésta como medio de transmisión para comunicarse con otros dispositivos.

1.1.2. Nuevos escenarios

En esta sección describiremos algunos escenarios de ejemplo que se basan en la capacidad de movilidad de los dispositivos y en su ubicuidad. Estos escenarios pueden parecer futuristas pero los retos tecnológicos que plantean son abordables en poco tiempo, si damos respuestas a algunas de las necesidades que se plantean, sobre todo a nivel software, y que han sido parte de la motivación de esta tesis.

- **Personalización**

  María dentro de un par de meses comienza sus vacaciones, este año le gustaría ir a Francia siempre tuvo ganas de conocer ese país, sobre todo su

capital París. Antes de irse de vacaciones debe entregar un proyecto en su trabajo, y no tiene mucho tiempo para organizar su viaje, así que delega esta tarea al asistente personal que tiene instalado en su PDA. Este software realiza una búsqueda de billetes de avión y hoteles que mejor se adapten a las preferencias de María, además sabe que a María le gusta mucho la pintura impresionista y que las colas en los museos obligan a realizar reservas previas de las entradas, también sabe que recientemente un par de amigos de María han trasladado su residencia a una localidad cercana a París, por lo que probablemente sea un buen momento para ir a visitarlos.

Cuando el asistente detecta que María ya se encuentra en su casa, le informa mediante una alerta de algunas de las ofertas de avión y hotel que ha encontrado para saber cuál es la que más le satisface y si quiere realizar la reserva de alguna en concreto. El asistente también le informa de la posibilidad de reservar entradas en algunos museos, y le recuerda que es un buen momento para visitar a sus amigos, pero que debería ponerse en contacto con ellos para saber si estarán en esas fechas. A María le parece una gran idea e indica a su PDA que quiere realizar la llamada en ese momento, la PDA se pone en contacto con el teléfono y marca el número para que María puede hablar. A sus amigos les hace mucha ilusión la visita, así que le indica a la PDA que la incluya en el plan de viaje.

- **Hogar**

  Se aproxima la fecha del viaje de María, y el trabajo no le deja ningún tiempo libre. Pero su PDA sabe que es necesario informar a algunos dispositivos de la casa de que María va a pasar fuera unas cuantas semanas, así que lo anuncia al entorno. La nevera al recibir este anuncio realiza un chequeo de la comida que contiene, todos los productos poseen una *smart label* que los identifica y además almacena la fecha de caducidad, algunos de ellos es preciso consumirlos antes de su viaje, así que emplea su conexión a Internet para ofrecerle a María unas posibles cenas que le permitan consumirlos, la nevera no tiene display pero sabe que María siempre enciende la tele de la cocina antes de ponerse a cocinar así que le manda un mensaje a través de ella, y si María lo desea le enviará también algunos de sus sugerencias para la cena de hoy.

  La PDA también informa a la calefacción los días que María no va a estar en casa, para que se desconecte siempre que la temperatura no baje de 0 grados. La calefacción detecta que hay una serie de sensores de temperatura en las habitaciones y los configura para que le comuniquen cuando la temperatura baja de ese valor. Además, para que María no encuentre su casa muy destemplada, dos días antes de su regreso se activará como habitualmente

y de esta forma la casa estará a una temperatura adecuada sin realizar un gasto energético excesivo. A María le preocupa mucho que alguien note su ausencia en casa, así que ha configurado la televisión para que se encienda un rato todas las noches, además la televisión cuando se activa se pone en contacto con las luces para que estas se enciendan también y así parezca más real que hay alguien en casa. Además las persianas se suben y se bajan según los horarios habituales de María, con lo que es más difícil detectar su ausencia.

- **Aeropuerto**

  María llega por fin al aeropuerto, observa las pantallas y ve que su vuelo a París se ha retrasado una hora, de todas formas se dirige a la puerta de embarque, en el fondo se alegra, así podrá refinar un último apartado del documento del proyecto y enviárselo a su jefe, lo redacta en su PDA y le indica que necesita enviarlo, la PDA busca algún punto de acceso inalámbrico, detecta uno, pero para tener un buena conexión María debe aproximarse más, y así se lo indica mediante la visualización de un mensaje. María se acerca y consigue transmitirlo. María sólo ha enviado el apartado que estaba modificando y unas directivas de cómo introducirlo en el documento final y a quién mandárselo, cuando todo esto llegue al PC de su oficina, será allí donde se realizarán todas estas tareas.

  En ese momento, anuncian por los altavoces un cambio de puerta de embarque, María no conoce muy bien el aeropuerto pero su PDA le visualiza un mapa del mismo y le indica cuál es el camino óptimo para ir a la nueva puerta de embarque.

  Cuando llega a la puerta de embarque y espera para poder entrar en el avión recibe una alerta en su PDA que alguien en su entorno busca a un jugador para una partida de damas, María acepta y se le descarga el juego para poder comenzar la partida y así amenizar el tiempo de espera.

- **Turismo**

  Por fin, María llega a París, ha empezado sus vacaciones. En su PDA tiene sus planes de viaje, al llegar al aeropuerto se le descarga un mapa del mismo y así puede llegar antes a la salida. Cuando llega al hotel la recepcionista toma sus datos y le configura a María una tarjeta inteligente inalámbrica. Al entrar en el ascensor no necesita marcar el piso al que va, sino que la propia tarjeta se comunica con el ascensor para indicárselo. Del mismo modo cuando llega a la puerta de su habitación ésta se abre automáticamente en cuanto María se aproxima. Al entrar en la habitación se enciende la televisión automáticamente apareciendo un mensaje de bienvenida para

María y una serie de normas del hotel, como horarios del comedor, servicio de habitaciones, etc.

María deja el hotel para pasear por las calles de París, como es la primera vez que visita la ciudad su PDA le proporciona un mapa de la zona. Al aproximarse por las calles y a través de su teléfono móvil le llegan alertas sobre determinadas tiendas que existen en la zona, el teléfono móvil se las pasa a la PDA que selecciona aquellas que más le interesan a María y se las muestra.

A María también le gusta disfrutar "sin tecnología" de su viaje, así que deshabilita su PDA y decide callejear por París y fijarse ella misma en lo que le rodea. Pero cuando llega la hora de cenar, vuelve a habilitar su PDA para que le localice el restaurante de comida francesa más cercano en la zona y que tenga una mejor relación calidad/precio.

- **Automóvil**

  Después de un par de días en París, ha llegado el día de ir a visitar a sus amigos, María ha reservado para ello el alquiler de un coche. Cuando se sube en él, la PDA descubre que tiene sistema de navegación así que le indica cuál es la dirección a la que se dirige María, para que le ayude a seguir el camino correcto. Además ajusta la música al nivel adecuado, y el coche, como detecta que María no es francesa, le indica algunas de la normas de conducción que debe tener en cuenta en Francia, mediante los altavoces del coche y por supuesto, en su idioma.

1.1.3. Nuevos retos

En la sección anterior hemos visto algunos escenarios que ilustran situaciones ficticias pero a las que la tecnología está dando respuesta. En las distintas situaciones hemos visto cómo los diferentes dispositivos que hemos descrito en la sección 1.1.1 le facilitan a María poder irse de viaje, sin tener que estar pendiente de todo lo que esto implica. Es importante darse cuenta que María, como usuaria, lo que percibe es que el entorno que le rodea se adapta a sus preferencias, a María no le importa que su calefacción sea capaz de regularse cooperando con los sensores que tiene situados en su casa, lo que le importa es que su casa siempre tenga la temperatura que ella desea.

Con el ejemplo anterior lo que queremos ilustrar es que la tecnología debe centrarse en proporcionar servicios adaptados al usuario, y por lo tanto, el **modelo de aplicaciones en computación ubicua debe orientarse a proporcionar y componer servicios personalizados al usuario final**. Esta idea rompe con

las respuestas iniciales que se han dado al desarrollo de aplicaciones software en dispositivos móviles y ubicuos, en las que el principal objetivo es que una serie de programas software puedan ejecutarse en dispositivos limitados.

Con esta nueva forma de abordar el problema del desarrollo de servicios, hemos realizado un análisis de los componentes software que es necesario incluir en cada uno de los tipos de dispositivos que hemos considerado anteriormente.

- **Dispositivos personales**

  Son los dispositivos más completos y que además acompañan al usuario. En ellos deberán existir los siguientes componentes:

  - Anuncio de los servicios que ofrecen.
  - Descubrimiento de servicios alcanzables.
  - Gestión de perfiles y preferencias de usuario.
  - Personalización de servicios.
  - Composición de servicios.
  - Sincronización de datos almacenados entre diferentes dispositivos.
  - Seguridad.

- **Dispositivos de función específica**

  En este caso, los componentes deben facilitar y estar orientados a la realización de la tarea específica para la que han sido fabricados. Así serán necesarios los siguientes componentes:

  - Anuncio de los servicios que ofrecen.
  - Descubrimiento de servicios alcanzables que puedan facilitar la realización de su tarea.
  - Composición de servicios.
  - Configuración para que puedan adaptarse a otros sistemas, a los requisitos del usuario o al entorno en el que se encuentran.
  - Seguridad.

- **Sensores y actuadores**

  Su limitación y su propósito específico para proporcionar o realizar una tarea concreta, hace que los componentes necesarios en estos dispositivos se reduzcan a:

- Anuncio de los servicios que ofrecen.
- Configuración para que puedan adaptarse a otros sistemas, a los requisitos del usuario o al entorno en el que se encuentran.
- Seguridad.

Como se observa en la enumeración realizada, existen una serie de componentes comunes a todos los tipos de dispositivos, el planteamiento de esta tesis doctoral se va a centrar en proporcionar tecnologías middleware aplicables a todos los dispositivos, de tal manera que cada módulo compartido funcione del mismo modo en cada uno de ellos y por lo tanto, permita la interoperabilidad, sin limitar la flexibilidad a la hora del desarrollo de nuevos servicios. Así, estos módulos comunes son:

- Descubrimiento y/o anuncio de servicios.
- Configuración.
- Seguridad.

Los módulos dependientes del tipo de dispositivo son:

- Composición de servicios.
- Gestión de perfiles y preferencias de usuario.
- Personalización de servicios.
- Sincronización de datos almacenados.

Para definir esta serie de módulos que serán tecnologías básicas para la realización de un middleware que dará soporte al desarrollo de servicios en entornos de computación ubicua, es necesario tener en cuenta las restricciones que impone el entorno, las características hardware/software que tienen los nuevos dispositivos y las características de los diferentes protocolos de red que se emplean.

En cuanto a los entornos, la característica fundamental es que son muy dinámicos y cambiantes. Utilizando nuestro ejemplo, en su casa María puede tener todos sus electrodomésticos interconectados a una red, probablemente cableada y además los elementos que la forman son más o menos estables. En cambio en el aeropuerto o en las calles de París, los elementos con los que puede interaccionar María dependerán en buena medida de su posición, porque podrá alcanzarlos

cuando se aproxime a ellos con su PDA. Además, el tipo y características de estos dispositivos variará enormemente, algunos pertenecerán a otros usuarios, otros estarán embebidos en el entorno físico proporcionando servicios de terceros, etc. Por lo tanto, es necesario que las tecnologías middleware aporten soluciones que se adapten a ambas situaciones, y puedan adaptarse tanto a entornos dinámicos como estáticos.

En cuanto a los dispositivos, la característica fundamental es su heterogeneidad y las limitaciones de algunos de ellos. En los escenarios en los que se mueve María tenemos desde dispositivos personales, como su PDA o su teléfono móvil, que poseen interfaces amigables para el usuario, que funcionan con baterías y emplean para comunicarse protocolos inalámbricos; hasta dispositivos fijos, con conexión a red fija y sin interfaz como su nevera, hasta los más limitados como son los sensores de temperatura. Por lo tanto, a la hora de definir tecnologías middleware que deben proporcionarse en todos ellos, debemos proponer soluciones simples y flexibles que sean factibles de implementar hasta en los más limitados dispositivos.

En cuanto a los protocolos de red, la característica fundamental es su heterogeneidad y sus diferentes prestaciones en cuanto ancho de banda, calidad de la transmisión y conectividad. En el hogar, los electrodomésticos de María pueden estar conectados empleando una red fija de alta velocidad, pero su PDA, su teléfono móvil, y los sensores situados en su hogar emplean protocolos inalámbricos, que se caracterizan por:

- calidad cambiante de las transmisiones, debido a los errores en la interfaz radio;
- conectividad no transitiva, debido al corto alcance de algunos protocolos y a la dificultad de imponer soluciones de encaminamiento de múltiples saltos;
- coste de comunicación, debido fundamentalmente al coste de baterías que supone la transmisión;

Por lo tanto, las soluciones que proporcionemos deben intentar minimizar el número y cantidad de transmisiones a realizar, de manera que las soluciones propuestas sean eficientes tanto en redes fijas como en redes móviles.

Los servicios proporcionados en entornos ubicuos deben estar orientados a realizar las tareas que desea el usuario, es necesario también que tengan las siguientes características:

- autonomía propia para poder alcanzar los objetivos que quiere el usuario, pero sin necesidad de interaccionar continuamente con él;

- capacidad de cooperar con otros sistemas para poder obtener información o ejecutar tareas que las limitaciones del dispositivo no les permiten realizar de forma local;
- movilidad para poder ejecutarse en los sistemas que tengan la información localmente, de forma que se minimice el número de transmisiones, se compensen las limitaciones de los protocolos inalámbricos y no se dependa de protocolos de encaminamiento multisalto.

Analizando estas características hemos considerado que el paradigma de computación que mejor se adapta a todas ellas es el paradigma de agentes móviles. Éste es el motivo que nos lleva a proponer la aplicación de la tecnología de agentes como tecnología middleware para el desarrollo de servicios en entornos de computación ubicua. Aunque por definición, esta tecnología se adapta a las características de estos entornos y ya existen muchas implementaciones, estudios y estándares al respecto, no se había involucrado a dispositivos limitados ni se habían tenido en cuenta las restricciones que imponen las redes inalámbricas y la posibilidad de formación de redes ad-hoc entre dispositivos limitados, por lo tanto, es necesario realizar un análisis detallado de todos los aspectos desarrollados hasta este momento para adaptarlos a los nuevos requisitos y retos que se nos plantean.

El análisis y definición de todos los módulos definidos en este apartado, teniendo en cuenta las restricciones enumeradas anteriormente, presenta importantes retos para la investigación e involucra un importante trabajo. Esta tesis abordará en primer lugar la definición de un mecanismo de **descubrimiento y anuncio de servicios**, y en segundo lugar la **adaptación de paradigma de agentes móviles** que facilitará la realización de los módulos de configuración y composición de servicios. En tareas como la gestión de perfiles y preferencias de usuario, y seguridad no se realizarán aportaciones, aunque el estudio realizado sobre los requisitos que imponen los entornos ubicuos serán útiles para realizar la definición de soluciones y propuestas para cubrir estos retos.

1.2. Objetivos

En esta tesis nos hemos planteado realizar contribuciones en el campo de la definición de tecnologías middleware, en las que se apoyarán los servicios y aplicaciones que se desarrollen en entornos de computación ubicua. Como hemos ido describiendo y justificando a lo largo de los apartados anteriores, los objetivos concretos que nos hemos planteado en la realización de esta tesis doctoral son los que enumeramos a continuación:

- Estudio y análisis de las tecnologías relacionadas con las objetivos abordados en esta tesis doctoral:

  - Estudio de los protocolos de red que permiten la movilidad de los usuarios y la ubicuidad de la tecnología.
  - Estudio de las tecnologías para el desarrollo de aplicaciones en dispositivos limitados.
  - Estudio de los mecanismos existentes de descubrimiento y anuncio de servicios.
  - Estudio del paradigma de agentes móviles y su implementación en dispositivos limitados.
  - Estudio de proyectos de investigación relacionados con la computación ubicua y que poseen retos comunes con los planteados en esta tesis doctoral.

- Contribución a la definición de un mecanismo para proporcionar descubrimiento y anuncio dinámico de servicios.

  El alcance de este objetivo lo hemos dividido en una serie de subobjetivos:

  - Estudio y análisis detallado de los protocolos de descubrimiento y anuncio dinámico de servicios.
  - Enumeración detallada y justificada de las necesidades y requisitos en los que se debe fundamentar la definición de nuestro nuevo protocolo.
  - Definición de un nuevo protocolo de descubrimiento y anuncio dinámico de servicios.
  - Análisis de las prestaciones del nuevo protocolo y comparativa con otros protocolos existentes, en diferentes escenarios.

- Contribución a la implantación del paradigma de agentes móviles como tecnología middleware para el desarrollo de servicios y aplicaciones en entornos ubicuos.

  El alcance de este objetivo lo hemos definido en una serie de subobjetivos:

  - Estudio y análisis del modelo de referencia FIPA, centrándonos en el servicio de páginas amarillas, denominado Directory Facilitator.
  - Análisis de las posibles soluciones para la adaptación del servicio de páginas amarillas definido en FIPA a entornos ubicuos.

- Definición de nuestra solución basada en el protocolo de descubrimiento de servicios propuesto y análisis de su compatibilidad con el modelo de referencia FIPA.
- Realización de contribuciones en los foros de discusión del comité técnico FIPA Ad-Hoc.
- Análisis de la viabilidad de implantación de nuestra propuesta basándonos en la tecnología Java 2 Micro Edition (J2ME).

A lo largo de este capítulo hemos ido reflejando la necesidad de abordar otras tecnologías middleware básicas para el desarrollo de servicios en entornos de computación ubicua, como la seguridad, y la composición y personalización de servicios según las preferencias del usuario, su localización y su entorno. Aunque en esta tesis doctoral no se van a abordar estos temas, las conclusiones obtenidas del estudio de las necesidades y requisitos en los entornos de computación ubicua pueden servir como base para realizar propuestas en esta temática.

Además, en las aportaciones concretas que realicemos tendremos en cuenta estos aspectos para que estos trabajos puedan apoyarse y extender al que nosotros realicemos. Así, en el protocolo de descubrimiento y anuncio de servicios somos conscientes de que se deberán introducir mecanismos de seguridad y además, deberá ser lo suficientemente flexible, para que pueda ser utilizado por cualquier descripción y definición de servicios que se realice a más alto nivel. En cuanto a la implantación del paradigma de agentes en entornos ubicuos, esta propuesta está motivada en gran parte, por la flexibilidad y facilidad que aporta este paradigma para desarrollar servicios personalizados, aunque es cierto que en este campo todavía existen muchos retos por alcanzar y a los que comienza a dar respuesta la inteligencia artificial.

1.3. Plan de trabajo

El plan de trabajo que se ha seguido en la realización de esta tesis doctoral se detalla a continuación:

- Estudiar las características de los entornos de computación ubicua.
- Analizar las necesidades y restricciones que imponen estos entornos para el desarrollo de aplicaciones.
- Contribuir a la adaptación del paradigma de agentes móviles para ofrecer y componer servicios en entornos de computación ubicua. Para ello se han planificado las siguientes subtareas:

- Estudiar y analizar el paradigma de agentes móviles, así como las plataformas y estándares existentes.
- Estudiar las especificaciones actuales de FIPA y adaptarlas para operar en estos entornos.

El estudio de la adaptación del elemento definido en la arquitectura FIPA como Directory Facilitator nos llevó a incluir nuevas tareas en el desarrollo de la tesis:

- Contribuir con la definición de un protocolo de descubrimiento y anuncio dinámico de servicios. Para ello se planificaron las siguientes subtareas:
  - Estudiar y analizar los protocolos de descubrimiento y anuncio dinámico de servicios propuestos en la literatura.
  - Definir un nuevo protocolo de descubrimiento y anuncio dinámico de servicios adaptado a las restricciones de entornos ubicuos.
  - Analizar las prestaciones del protocolo definido, comparándolo con otros protocolos existentes.
- Integrar la propuesta realizada, respecto al descubrimiento y anuncio de servicios, en la arquitectura FIPA.
- Realizar el análisis de resultados, la formalización y la redacción de la tesis, de las conclusiones y propuesta de trabajos futuros.

1.4. Estructura de la memoria

La memoria de esta tesis se estructura como sigue:

En el capítulo 2, se presenta el estado del arte en el ámbito en el que se desarrolla este trabajo. Por un parte se revisan las tecnologías de red utilizadas por los diferentes dispositivos existentes en los entornos ubicuos, y por otra la tecnología que ha permitido solventar el problema de heterogeneidad a la hora de desarrollar aplicaciones que se ejecutan en dispositivos limitados: Java 2 platform Micro Edition (J2ME). También se incluye un breve análisis introductorio a las tecnologías directamente relacionadas con las contribuciones realizadas en esta tesis doctoral, los mecanismos de descubrimiento de servicios y las plataformas de agentes móviles en dispositivos limitados. Finalizamos, con una descripción de algunos de los más destacados proyectos de investigación relacionados con

la computación ubicua y que pretenden abordar algunos retos similares a los tratados en esta tesis doctoral.

En el capítulo 3, se realiza un estudio detallado de los mecanismos de descubrimiento de servicios existentes en la literatura, se analiza el problema del descubrimiento de forma teórica y después se describen los protocolos definidos en el ámbito de Internet.

En el capítulo 4, se proporciona un diseño razonado de nuestra propuesta para el descubrimiento de servicios en entornos ubicuos, el protocolo Pervasive Discovery Protocol, PDP. El diseño realizado se apoya en estudios analíticos y de simulación para comprender las mejoras obtenidas con cada uno de las nuevos mecanismos que hemos introducido en el protocolo.

En el capítulo 5, se realiza un análisis de prestaciones del protocolo propuesto PDP respecto a sus parámetros fundamentales: tiempo de disponibilidad de los dispositivos, número de dispositivos y tamaño de la caché. También se presenta una comparativa de PDP con otros protocolos existentes en escenarios tanto homogéneos, en los que todos los dispositivos tienen las misma características de movilidad, como heterogéneos.

En el capítulo 6, se describe la propuesta para adaptar las especificaciones FIPA a entornos ad-hoc, centrándonos principalmente en la definición de un nuevo agente que proporciona un servicio de páginas amarillas adaptado a estos entornos, denominado Service Discovery Agent, SDA. Se analiza también la viabilidad de implementación de nuestra propuesta, en dispositivos reales mediante la tecnología J2ME.

Finalmente, en el capítulo 7 se resumen las principales conclusiones del trabajo realizado y se enumeran una serie de posibles líneas de continuación del mismo.

Como parte de la memoria hemos incluido una serie de apéndices para aclarar y completar algunas de las contribuciones de esta tesis doctoral. Así:

En el apéndice A, se describe la metodología de simulación empleada para realizar el diseño y el estudio de prestaciones del protocolo PDP.

En el apéndice B, se realiza una descripción detallada del protocolo para facilitar su implementación y garantizar la compatibilidad entre diferentes implementaciones.

En los apéndices C y D se incluyen las contribuciones originales realizadas al comité técnico FIPA Ad-Hoc.

1.5. Historia de la tesis

En esta sección, realizamos un breve resumen de la historia de esta tesis, y enumeramos las publicaciones realizadas durante estos últimos años y en las que se reflejan parte de sus resultados.

La temática en la que se comenzó esta tesis doctoral era la aplicación de la tecnología de agentes móviles a redes de telefonía móvil, como GSM. En [Campo et al., 2001b] abordábamos nuestro primer estudio del estado del arte en esta temática, y proponíamos aplicarla para reducir el coste, en cuanto a tráfico de señalización, que supone la gestión de localización de los usuarios móviles en una red celular. En [Campo et al., 2001a] describíamos un primer análisis de los retos que suponía introducir una plataforma de agentes móviles en dispositivos limitados, en concreto en teléfonos móviles, empleando tanto la tecnología J2ME como la tecnología Java Card.

Posteriormente, comenzamos a investigar también en la posibilidad de aplicar la tecnología de agentes a lo que se denomina computación ubicua, un tema que empezó a estar de actualidad gracias a la comercialización de nuevos dispositivos personales, tipo agendas digitales y teléfonos móviles, con mayores capacidades de procesamiento y almacenamiento, y al desarrollo de nuevos protocolos inalámbricos de corto alcance que permiten que estos dispositivos formen redes de forma espontánea, ofreciéndole al usuario nuevos servicios adaptados al entorno en el que se encuentra. Nuestras primeras propuestas en esta temática se recogen en [Campo, 2002a].

Uno de los objetivos que nos marcamos a la hora de realizar contribuciones en el campo de los agentes móviles, era basarnos en los estándares FIPA. Esto nos llevó a analizar los servicios básicos que debe proporcionar según FIPA una plataforma de agentes, en concreto nuestro trabajo se centró en el servicio de páginas amarillas, denominado Directory Facilitator. Las carencias encontradas al intentar proponer una solución eficiente a este problema, nos llevó a realizar un estudio más genérico del problema de descubrimiento de servicios en redes ad-hoc, y a proponer un nuevo protocolo de descubrimiento denominado Pervasive Discovery Protocol. La primera versión del protocolo, así como su aplicación en plataformas de agentes se propuso en los artículos [Campo, 2002c] y [Campo et al., 2002a].

Cuando el comité técnico FIPA Ad-Hoc solicitó a la comunidad propuestas para la adaptación de las especificaciones FIPA a entornos ad-hoc, enviamos un documento resumen de nuestro trabajo hasta ese momento, [Campo et al., 2002b]. A raíz de esta contribución, fuimos invitados a participar en la siguiente reunión del FIPA celebrada en Helsinki en Junio de 2002, para esta reunión realizamos un nuevo documento clarificando algunas detalles de nuestra propuesta

[Campo, 2002b].

Durante todo este tiempo hemos trabajado sobre la viabilidad de implementar nuestras propuestas en dispositivos reales basándonos en el lenguaje de programación J2ME. Los resultados de este trabajo se recogen en proyectos fin de carrera dirigidos por la autora de esta tesis [Díez-Andino, 2002], [Perea, 2003], así como en congresos nacionales e internacionales [Campo et al., 2003], [Díez-Andino et al., 2003c], [Díez-Andino et al., 2003a].

Por último, en [Almenárez y Campo, 2003] se propuso una versión segura del protocolo de descubrimiento de servicios, denominada Secure Pervasive Discovery Protocol, basándonos en propuestas sobre seguridad en computación ubicua desarrolladas en nuestro grupo de trabajo.

Capítulo 2

Estado del arte

En este capítulo realizamos un estudio sobre las diferentes tecnologías que es necesario tener en cuenta a la hora de realizar las contribuciones propuestas en esta tesis doctoral. En la sección 2.1, realizamos un estudio de las nuevas tecnologías de red, tanto las inalámbricas que emplean los dispositivos personales, como las que se emplean en entornos del hogar y que utilizan la red eléctrica o la red telefónica como medio de transmisión. En la sección 2.2, describimos la tecnología que ha permitido solventar el problema de heterogeneidad a la hora de desarrollar aplicaciones que se ejecuten en dispositivos limitados, que es Java 2 platform Micro Edition (J2ME). Este estudio nos permitirá analizar las necesidades y restricciones que imponen los entornos de computación ubicua, a la hora de definir tecnologías middleware para el desarrollo de servicios en este tipo de entornos.

Además, en este capítulo incluimos una descripción detallada de las tecnologías directamente relacionadas con las contribuciones que se realizan en esta tesis doctoral. En la sección 2.3, describimos los protocolos de descubrimiento y anuncio de servicios y tecnologías relacionadas, y concluimos con el análisis que nos ha llevado a la necesidad de realizar una contribución en este campo, a pesar de la posibilidad de aplicar las soluciones existentes. En la sección 2.4, realizamos un estudio de la tecnología de agentes, centrándonos en los agentes móviles y en las implementaciones de plataformas de agentes desarrolladas en Java, analizando la viabilidad de implantar esta tecnología sobre dispositivos limitados J2ME.

2.1. Nuevos protocolos de comunicación

En esta sección realizamos una introducción a los nuevos protocolos de comunicación que han aparecido estos últimos años, y que están permitiendo que la computación móvil y ubicua comience a ser una realidad. No va a ser objetivo de esta tesis proporcionar ninguna aportación en este campo, pero se considera fundamental como base para entender algunos de los requisitos que se deben imponer, a la hora de definir tecnologías middleware para el desarrollo de aplicaciones distribuidas en este tipo de entornos.

Se realiza un breve estudio de los protocolos más importantes en el ámbito de las redes de área extensa, sección 2.1.1, y área local, sección 2.1.2, cubriendo las más importantes tecnologías inalámbricas que han permitido la movilidad de los usuarios. En la sección 2.1.3 se cubren los protocolos de menos alcance, que se han denominado de área personal porque han sido diseñados para la comunicación entre dispositivos personales del usuario. Por último, en la sección 2.1.4 se cubren los nuevos protocolos que se están desarrollando para conectar en red los diferentes dispositivos localizados en un hogar.

2.1.1. Redes de Área Extensa

La movilidad de los usuarios se ha conseguido gracias a la aparición de protocolos inalámbricos de área extensa, en concreto con la aparición de las redes de telefonía celular, que han tenido un gran impacto en estos últimos años por su aceptación por parte de un gran número de usuarios. Estas redes conocidas como redes de Segunda Generación (2G), como GSM, están dando paso a las denominadas redes de generación intermedia entre la segunda y tercera generación (2.5G), como GPRS, y a las de Tercera Generación (3G), como UMTS y actualmente en la literatura ya están apareciendo trabajos de investigación sobre lo que será la siguiente generación, la denominada Cuarta Generación (4G).

Debido al gran impacto que han tenido las redes de telefonía móvil de segunda generación varios organismos en los últimos años están estandarizando redes móviles de 3G. A nivel mundial el ITU está especificando las características comunes que tendrán estas redes para garantizar su interoperabilidad, es lo que se denomina sistema IMT-2000. UMTS es uno de estos sistemas, y será el sucesor en Europa del sistema GSM.

Una característica común de todas estas redes es que se apoyan fuertemente en la infraestructura de red: la cobertura está dada por estaciones base, los recursos de radio están gestionados desde una ubicación central, y los servicios están

Tipo	Tecnología	Banda de frecuencia	Tasa de datos
2G	GSM	900/1800/1900 MHz	9.6 kbps
2.5G	GPRS	900/1800/1900 MHz	$\leq$ 171 kbps
	EDGE	900/1800/1900 MHz	$\leq$ 384 kbps
3G	UMTS	2 GHz	$\leq$ 2 Mbps

Cuadro 2.1: Tecnologías de redes inalámbricas de área extensa.

integrados en el sistema.

En un principio estos sistemas sólo proporcionaban al usuario final un servicio de voz personalizado al que podía acceder en cualquier momento y en cualquier lugar. La demanda por parte de los usuarios de poder acceder con esta misma flexibilidad a los mismos servicios que tienen en redes fijas, inicialmente navegación web y correo electrónico, ha llevado a importantes retos tecnológicos que se han ido abordando a través de las distintas evoluciones de los sistemas de segunda generación.

En el Cuadro 2.1 resumimos las características de las diferentes tecnologías que describiremos brevemente a lo largo de este apartado.

GPRS

General Packet Radio Service (GPRS) [GSM 02.60, 1997] es una evolución del sistema de telefonía móvil GSM y es un estándar de transición al sistema de Tercera Generación UMTS. Este sistema ha sido estandarizado por el ETSI pero su implantación no ha tenido lugar hasta el año 2000.

GPRS [Bettstetter et al., 1999] básicamente añade conmutación de paquetes de datos a todos los niveles de la red GSM, optimizando la utilización de los canales radio para el tráfico a ráfagas (por ejemplo, el tráfico de Internet), y realizando un uso más eficaz de los recursos de la red, de manera que se pueden alcanzar mayores tasas de datos.

La arquitectura del sistema GPRS no utiliza las centrales de conmutación de GSM para el transporte de paquetes de datos, sino que los controladores de estaciones base de radio están directamente conectados a la red de datos a través de dos nuevos tipos de servidores, también denominados nodos *GPRS Support Node* (GSN), que son de dos tipos:

- *Serving GPRS Support Node* (SGSN), se encarga de la entrega de paque-

tes a y desde los terminales móviles, también realiza tareas de gestión de movilidad y gestión de sesión, y lleva a cabo funciones de autenticación y tarificación.

- *Gateway GPRS Support Node* (GGSN), actúa como un interfaz entre la red GPRS y la red de datos externa, y también lleva a cabo funciones de autenticación y tarificación.

En el transporte de voz se siguen utilizando los mecanismos tradicionales de GSM.

Desde el punto de vista de los usuarios, GPRS ofrece unos menores tiempos de acceso, conectividad permanente (*always on*) y una tasa de datos mayor que la proporcionada por GSM, teóricamente se pueden alcanzar tasas de hasta 171 kbps, aunque los valores reales se aproximan a los 40 kpbs. Además, la tarificación se realiza según el volumen de información transmitida y no por tiempo de conexión.

EDGE

El paso siguiente en la evolución de los sistemas de telefonía móvil fue el sistema *GSM Enhanced Data rates for Global Evolution* (GSM/EDGE) [3GPP TS 43.051, 2002] que consiste en utilizar una modulación más eficiente en el interfaz radio para poder obtener mayores tasas de transmisión, utilizando las mismas frecuencias en las que opera GSM, lo que supone una gran ventaja a las operadoras en el camino hacia la implantación de sistemas de tercera generación.

En el sistema EDGE [Muller et al., 2001] se define una nueva red de acceso, la *GSM/EDGE Radio Access Network* (GERAN), que ofrece diferentes posibilidades de modulación, que combinadas con diferentes tasas del codificador de canal convolucional utilizado, dan lugar a tasas de transferencia máximas de 384 kbps. Los principales objetivos en la definición de GERAN fueron poder proporcionar servicios de tercera generación a los usuarios, reutilizando la mayor parte de infraestructura existente y que esta red de acceso fuese capaz de interoperar tanto con la *core network* de GSM-GPRS, como con la de los futuros sistemas UMTS.

El ETSI comenzó a estandarizar el sistema EDGE, pero debido a su proximidad con los sistemas de tercera generación, se delegó este trabajo al 3GPP en verano de 2000. El estándar ha evolucionado de forma paralela a las especificaciones de UMTS, de hecho existe la Release 99, Release 4 y Release 5 del mismo.

UMTS

El sistema *Universal Mobile Telecommunications System* (UMTS) se engloba dentro de la iniciativa mundial de estandarización de sistemas de telefonía móvil de tercera generación IMT-2000. El objetivo común de estos sistemas es proporcionar al usuario final:

- Convergencia de servicios, que le permitirá acceder a los mismos servicios que le proporcionan las redes fijas, gracias a mayores tasas de transmisión en el interfaz radio (hasta 2Mbps).
- Movilidad personal, que permitirá acceder a los servicios de la red, independientemente del lugar en el que se encuentre o del terminal a través del que se accede a ellos.
- Portabilidad y movilidad de servicios, a través del concepto *Virtual Home Enviroment* (VHE) que le permitirá mantener la personalización de sus servicios, independientemente del lugar en el que se encuentre o del terminal a través del que accede a ellos.

UMTS es el estándar de facto para la evolución de los sistemas GSM, técnicamente introduce importantes cambios respectos a los sistemas anteriores tanto en la red de acceso, como en la *core network*. El 3GPP es el organismo encargado de su estandarización, y hasta el momento ha publicado las siguientes versiones:

- **Release 99** [3GPP TS 21.101, 2000] es un estándar firmemente establecido y será el que se utilice en el despliegue inicial de UMTS en todas las operadoras europeas. El objetivo de esta versión es aumentar las tasas de transmisión para poder proporcionar servicios multimedia a los usuarios, conservando parte de la infraestructura de red de GSM-GPRS/EDGE, para minimizar a las operadoras los costes iniciales de implantación. Así, esta versión se centra en definir un nuevo interfaz radio, denominado *UMTS Terrestrial Radio Access Network* (UTRAN).
- **Release 4** [3GPP TS 21.102, 2001] en esta versión del estándar, la voz se transporta ya sobre IP y aparecen separadas las funciones de control y conectividad para voz.
- **Release 5** [3GPP TS 21.103, 2002] ésta es la versión denominada como *all IP*. IP será la tecnología de transporte en la *core network* para todo tipo de datos, y posiblemente también se empleará en la red de acceso radio UTRAN. En esta versión se culmina la separación entre los planos de transporte y de control.

Tecnología	Banda de frecuencia	Tasa de datos	Alcance
IEEE 802.11b	2.4 GHz	11 Mbps	100 m
IEEE 802.11g	2.4 GHz	54 Mbps	100 m
IEEE 802.11a	5 GHz	54 Mbps	100 m
HIPERLAN/2	5 GHz	54 Mbps	150 m

Cuadro 2.2: Tecnologías inalámbricas de área local.

- **Release 6** [3GPP TS 21.103, 2003] ésta es la última versión en la que se está trabajando, y el 3GPP pretende finalizar la especificación en Marzo de 2004.

2.1.2. Redes de Área Local

En los últimos años también han aparecido una serie de estándares y normas para redes LAN inalámbricas, como IEEE 802.11 o HIPERLAN, que proporcionan movilidad a los usuarios en entornos más limitados, pero que les permite acceso a Internet, por lo que están siendo unos claros rivales tecnológicos de las redes celulares 3G. Su modo de funcionamiento habitual es centralizado a través de puntos de acceso de forma similar a las redes anteriores, aunque también tienen modos en los que los terminales pueden comunicarse directamente entre sí.

Cuando los terminales se comunican entre sí, se dice que los nodos forman una red ad-hoc. En [Frodigh et al., 2000] se define una red ad-hoc, como una red formada sin ninguna administración central, en la que los nodos usan una interfaz inalámbrica para comunicarse. Habitualmente estos nodos son móviles por los que estas redes son muy dinámicas, en ellas los dispositivos que las forman aparecen y desaparecen de forma espontánea, por lo que el funcionamiento de la red no puede depender de un nodo concreto.

Tradicionalmente las redes ad-hoc se han empleado en aplicaciones militares en las que una configuración de red descentralizada, es una ventaja operativa o incluso una necesidad. En los últimos años se han definido una serie de protocolos inalámbricos que se están implementando cada vez más habitualmente en los nuevos dispositivos personales, lo que ha permitido que estas redes ad-hoc se formen entre estos nuevos dispositivos y en nuevas situaciones y por lo tanto, que se empleen en un mayor número de escenarios.

En el Cuadro 2.2 resumimos las características de las diferentes tecnologías que describiremos brevemente a lo largo de este apartado.

IEEE 802.11

El estándar [IEEE Std. 802.11, 1999] ha sido definido con el objetivo de permitir a usuarios inalámbricos conectarse a redes de área local. Está diseñado para ser compatible con el resto de estándares LAN cableados de la familia 802. El estándar define un mismo nivel MAC que usa el algoritmo *Carrier-Sense Multiple-Access with Collision-Avoidance* (CSMA/CA) y varias posibilidades de transmisión a nivel físico: infrarrojos, y las más populares, de espectro ensanchado por secuencia directa (*Direct-Sequence Spread Spectrum*, DSSS) y de espectro ensanchado por salto en frecuencia (*Frequency-Hopping Spread Spectrum*, FHSS) ambas en la banda ISM[1] (2,4 GHz). Se consiguen de esta forma velocidades de 1 y 2 Mbps.

Posteriormente, apareció el [IEEE Std. 802.11a, 1999] que utiliza *Orthogonal Frequency-Division Multiplexing* (OFDM) en la banda de los 5 GHz y que alcanza tasas de hasta 54 Mbps. Para que la Unión Europea aceptase el uso de esta banda de frecuencia, el grupo tuvo que realizar modificaciones al estándar recogidas en el [IEEE Std. 802.11h, 2003], en la que se incluyen también mecanismos de control de potencia de transmisión.

Sobre las formas de transmisión del estándar original en la banda ISM, se han definido varias ampliaciones para alcanzar mayores tasas binarias. Aparecen así:

- [IEEE Std. 802.11b, 1999] que utiliza *Complementary Code Keying* (CCK) sobre DSSS y consigue tasas de 5,5 Mbps y 11 Mbps. Ésta es la versión del estándar más utilizada y también es conocida comercialmente como WiFi.
- [IEEE Std. 802.11g, 2003], también denominado 802.11b extendido, que alcanza tasas de hasta 54 Mbps. Este estándar es compatible con las redes 802.11b y 802.11 DSSS.

Otras iniciativas dentro del grupo, algunas todavía en fase de desarrollo, son:

- IEEE 802.11e que se ha creado para introducir calidad de servicio sobre las implementaciones IEEE 802.11b, a y g.
- IEEE 802.11i que se ha creado para mejorar los mecanismos de seguridad e introducir autenticación a nivel MAC.
- IEEE 802.11n que se ha creado con el objetivo de desarrollar un estándar que permita alcanzar tasas de hasta 100 Mbps.

[1] Industrial-Scientific-Medical

El IEEE 802.11 define dos modos de arquitectura de red, por una parte el modo ad-hoc, en el que las estaciones se comunican entre sí directamente, y por otra el modo centralizado, en el que las estaciones acceden a la red a través de uno o varios puntos de acceso.

HIPERLAN/2

La norma [HIPERLAN/2, 1999] ha sido definida por el ETSI BRAN[2] con el objetivo de dar acceso inalámbrico de alta velocidad a redes de área local. Opera en la banda sin licencia de 5 GHz y alcanza tasas de transmisión de hasta 54 Mbps. La norma especifica una red de acceso radio que se puede utilizar con una variedad de núcleos de red, gracias a una arquitectura flexible que define la capa física y de control de enlace independiente al núcleo de red con el que opere. Además, especifica un conjunto de capas de convergencia que facilitan el acceso a los distintos núcleos. Se han definido varias capas de convergencia, en concreto existen definiciones para interoperar con redes IP, redes ATM, redes celulares de tercera generación y redes IEEE 1394.

La capa MAC de HIPERLAN/2 está basada en la aproximación *Time Division Multiple Access, Time Division Duplex* (TDMA/TDD) y ha sido diseñado para dar soporte a calidad de servicio, esencial para aplicaciones multimedia y de tiempo real. A nivel físico emplea la misma técnica que su estándar competidor el IEEE 802.11a, OFDM. En [Doufexi et al., 2002] se puede encontrar un estudio comparativo de ambas tecnologías.

HIPERLAN/2 depende de una topología de red celular y soporta dos modos de operación básicos, el modo centralizado y el modo directo. En el modo de operación centralizado los terminales se comunican entre sí y acceden al núcleo de red a través de puntos de acceso, este modo se utiliza cuando es necesario cubrir una zona amplia y por lo tanto, existen varios puntos de acceso. En el modo de operación directo existe un único punto de acceso que controla la asignación de recursos radio a los diferentes terminales, en este modo los terminales pueden comunicarse directamente entre ellos, a este modo de operación no se le denomina ad-hoc porque el funcionamiento de la red depende de un elemento central, el punto de acceso, para funcionar.

[2]European Telecommunications Standards Institute-Broadband Radio Access Networks

Tecnología	Banda de frecuencia	Tasa de datos	Alcance
IrDA	Infrarrojos	115 kbps 1.152 Mbps 4 Mbps	2 m
Bluetooth	2.4 GHz	1 Mbps	10/100m
IEEE 802.15.3	2.4 GHz	55 Mbps	10 m
IEEE 802.15.4	868 MHz 915 MHz 2.4 GHz	20 kbps 40 kbps 250 kbps	10-20 m

Cuadro 2.3: Tecnologías de área personal.

2.1.3. Redes de Área Personal

Paralelamente a los protocolos anteriores, estos últimos años se han desarrollado otros protocolos de más corto alcance, con un modo de operación principalmente ad-hoc. Aparecen así IrDA y Bluetooth, como mecanismos para comunicar dispositivos personales del usuario entre sí, con el objetivo de compartir y sincronizar información entre ellos. Por ello a este tipo de redes se les ha denominado redes de área personal, aunque estos protocolos también se están utilizando en electrónica de consumo y su uso puede extenderse a áreas locales.

En el Cuadro 2.3 resumimos las características de las diferentes tecnologías que describiremos brevemente a lo largo de este apartado.

IrDA

Infrared Data Association[3] fue creada en 1993 con el objetivo de desarrollar una tecnología inalámbrica por infrarrojos, de corto alcance, fácil de usar, de bajo coste e interoperable. El éxito de su iniciativa lo demuestra el hecho de que casi todos los ordenadores portátiles, agendas digitales, teléfonos móviles, cámaras digitales... lo soportan actualmente. Se puede decir que hasta la aparición de Bluetooth, esta tecnología era el único protocolo inalámbrico empleado para la transferencia y sincronización de datos entre este tipo de dispositivos. En la actualidad, más que tecnologías competidoras son complementarias dependiendo de la aplicación en la que se vayan a utilizar [Suvak, 2000].

El núcleo principal de la plataforma IrDA [Williams, 2000], que se denomina

[3]http://www.irda.org/

IrDA-Data, abarca tres protocolos obligatorios y uno opcional, estandarizados en el año 1994:

- *Infrared physical layer* (IrPHY), que soporta conexiones entre dispositivos a una distancia de 1 a 2 metros, y que para implementaciones de bajo consumo se reduce a 20cm. Se proporcionan tasas de transmisión de 115 kbps, 1.152 Mbps y hasta un máximo de 4 Mbps.
- *Infrared Link Access Protocol* (IrLAP), que proporciona conexiones fiables entre dispositivos para la transmisión de datos y mecanismos de descubrimiento de dispositivos.
- *Infrared Link Management Protocol* (IrLMP), que proporciona multiplexación de canales sobre una conexión IrLAP, y descubrimiento de servicios a través de *Information Access Service* (IAS).
- *Tiny Transport Protocol* (TinyTP), que proporciona control de flujo sobre conexiones IrLMP. Siendo éste el protocolo opcional.

Para promover la interoperabilidad de las aplicaciones que emplean la plataforma, se han definido un conjunto de protocolos que proporcionan servicios de comunicaciones: IrCOMM que emula la comunicación a través de puerto serie y paralelo y el IrLAN para el acceso a LANs tipo 802. Así como otros protocolos que proporcionan servicios a nivel de aplicación: IrOBEX que permite el intercambio de objetos (semejante a HTTP), IrTRAN-P para el intercambio de imágenes, IrMC para intercambio de información entre teléfonos móviles e IrJetSend que describe como interactuar con el protocolo HP JetSend.

Los principales inconvenientes de IrDA son su rango corto de cobertura, la necesidad de visión directa entre dispositivos, y su modo de operación punto a punto. Estos inconvenientes están siendo solventados parcialmente por lo que se denomina Advanced Infrared (AIR) [IBM Research Zurich, 1999].

Bluetooth

El protocolo inalámbrico [Bluetooth v1.1, 2001] surge como una iniciativa de Ericsson en 1994 con el propósito de sustituir los cables entre los dispositivos electrónicos, tales como teléfonos, PDAs, ordenadores portátiles, cámaras digitales, impresoras,... usando un chip de radio de bajo coste. En 1998, se unen a esta iniciativa IBM, Toshiba, Nokia e Intel formándose el *Bluetooth SIG*[4] para promover y estandarizar esta tecnología.

[4]http://www.bluetooth.com/

El protocolo Bluetooth trabaja en la banda ISM. La especificación define un canal de comunicación de máximo 720kbps en modo asimétrico o 432,6 kbps en modo simétrico con un rango óptimo de 10m (opcionalmente 100m).

Dos o más unidades Bluetooth que comparten el mismo canal forman una *piconet*. Dentro de una *piconet*, una unidad Bluetooth puede representar uno de dos papeles: maestro o esclavo. Cada *piconet* debe tener un maestro y puede tener hasta siete esclavos activos. Dos o más *piconets* pueden estar interconectadas formando lo que se denomina *scatternet*. Una unidad Bluetooth puede ser simultáneamente un miembro esclavo de múltiples *piconets*, pero solamente maestra en una de ellas. En el protocolo Bluetooth no hay transmisión directa entre esclavos, todas las comunicaciones deben pasar por el maestro, que organiza de acuerdo a un determinado esquema, las comunicaciones en la *piconet*.

El *IEEE 802.15 Working Group*, que surge en 1999 con el objetivo de definir una serie de estándares y normas para redes personales inalámbricas (*Wireless Personal Area Networks*, WPAN), decidió utilizar la especificación Bluetooth v1.1 como base de su estándar [IEEE 802.15.1, 2002] y así en él se estandariza la capa física y MAC de Bluetooth. Este grupo también está desarrollando nuevos estándares para WPAN:

- El *IEEE 802.15 WPAN Task Group 3* se encarga de la definición de estándares para soportar mayores tasas de datos, por encima de los 20 Mbps, con el objetivo de dar soporte a aplicaciones multimedia y procesado de imagen [Karaoğz, 2001].

  A finales de 2003 se ha publicado la primera versión de estándar [IEEE Std 802.15.3, 2003].

- El *IEEE 802.15 WPAN Task Group 4* se encarga de la definición de estándares para soportar menores tasas de datos, por debajo de los 200 kbps pero que optimiza el uso de las baterías, con el objetivo de ser utilizado en redes de sensores, etiquetas inteligentes, controles remotos [Callaway et al., 2002].

  A finales de 2003 se ha publicado la primera versión de estándar [IEEE Std 802.15.4, 2003].

En Noviembre de 2003, el Bluetooth SIG ha anunciado la finalización de la nueva versión del estándar Bluetooth, la versión 1.2. La nuevas funcionalidades incluidas en esta versión, totalmente complatible con la anterior, son las siguientes:

- Introducción de *Adaptative Frequency Hopping* (AFH) para reducir las interferencias provocadas por otros protocolos inalámbricos que emplean la

Tecnología	Banda de frecuencia	Tasa de datos	Alcance	Medio físico
HomeRF	2.4 GHz	10 Mbps	50 m	Inalámbrico
HomePNA	4–10 MHz	10 Mbps	–	Línea telefónica
HomePlug	4.3–20.9 MHz	14 Mbps	–	Red eléctrica

Cuadro 2.4: Tecnologías del hogar.

banda ISM, como IEEE 802.11b y IEEE 802.11g. De esta forma, se facilita la utilización simultánea de ambos protocolos en dispositivos móviles.

- Mejora de la calidad de las transmisiones de voz, sobre todo en ambientes ruidosos, introduciendo métodos de detección de errores.
- Mejora la velocidad de establecimiento de conexiones.

Se espera que los productos comerciales que implementen esta nueva versión salgan al mercado a finales de 2004.

2.1.4. Redes del Hogar

El entorno del hogar ha sido uno de los primeros escenarios en los que se ha querido implantar el concepto de computación ubicua [Dutta-Roy, 1999]. El hogar plantea retos tecnológicos particulares, el más importante es el de conseguir conectar entre sí los habituales electrodomésticos del hogar sin un coste adicional elevado a los usuarios. Así se han desarrollado una serie de protocolos para utilizar la red telefónica y la red eléctrica como medios de transmisión, también se ha empleado tecnología inalámbrica, y existen algunas iniciativas más invasivas como el uso del cable.

En el Cuadro 2.4 resumimos las características de las diferentes tecnologías que describiremos brevemente a lo largo de este apartado.

HomeRF

El *Home RF Working Group*[5] es un consorcio de las principales compañías fabricantes de PCs y electrónica de consumo y de telecomunicación que surge con el objetivo de permitir la interoperabilidad entre redes de voz y de datos en

[5]http://www.homerf.org

entornos del hogar, a través de un protocolo inalámbrico. El principal trabajo del grupo en estos últimos años ha sido la especificación de este protocolo, denominado *Shared Wireless Access Protocol* (SWAP), que auna características del estándar *Digital Enhanced Cordless Telephone* (DECT) para la transmisión de voz y de tecnologías WLAN para la transmisión de datos, pero siempre pensando en su uso para el desarrollo de aplicaciones en entornos del hogar.

SWAP [Negus et al., 2000] emplea a nivel físico la técnica FHSS en la banda ISM. A nivel MAC emplea por una parte TDMA para servicios interactivos y de tiempo real, y por otra, CSMA/CD para datos asíncronos, permitiendo alcanzar mayores tasas de transmisión, hasta 1,6 Mbps.

SWAP puede operar de dos formas, como red ad-hoc o como red centralizada. En el modo ad-hoc sólo se soportan comunicaciones de datos, todos los nodos son iguales y las funciones de control de red se distribuyen entre todos los nodos. En el modo centralizado, utilizado para conexiones de tiempo real, se precisa un punto de conexión para coordinar el sistema, el punto de conexión actúa como un gateway a la red telefónica conmutada (PSTN) y puede estar conectado a un PC para dar soporte también a servicios de datos. En SWAP el punto de conexión puede emplearse también para realizar gestión de potencia de los dispositivos estableciendo periodos de *wakeup* y *polling*.

Una red SWAP soporta hasta 127 nodos. Estos nodos puede ser de cuatro tipos diferentes:

- Puntos de conexión.
- Dispositivos de voz o datos síncronos, también denominados *I-nodes*.
- Dispositivos de datos asíncronos, también denominados *A-nodes*.
- Dispositivos mixtos de voz y datos.

El protocolo SWAP en su versión 2.0 ha añadido mejoras al estándar inicial, permitiendo tasas de datos de hasta 10Mbps y soporte al *roamming* entre varios puntos de conexión.

En Enero de 2003 el grupo se disolvió.

HomePNA

Home Phone Line Networking Alliance[6] fue creada en 1998 por los principales fabricantes de ordenadores y semiconductores para seleccionar, promover y estan-

[6] http://www.homepna.org

darizar tecnologías de red en el hogar empleando la red telefónica tradicional.

La utilización de la red telefónica para la transmisión de datos presenta importantes retos debido a los problemas que plantea de impedancias, reflexiones y atenuaciones. HomePNA 1.0, basado en la propuesta de Tut Systems, solventó estos problemas, empleando el rango de 5.5–9.5 MHz y basándose en el estándar IEEE 802.3, pero encapsulando tramas Ethernet en paquetes mayores, de esta forma se obtienen tasas de datos de hasta 1 Mbps, que permiten la compartición de recursos entre PCs (ficheros, acceso a Internet, periféricos), que eran los requisitos iniciales marcados.

Esta tasa de datos era insuficiente para dar soporte a tráfico multimedia y de entretenimiento, que eran las nuevas aplicaciones que demandaban los usuarios en su hogar, así el grupo estandarizó la versión 2.0, propuesta conjuntamente por Lucent Tecnologies y Broadcom Corporation, en la que se alcanzan tasas de hasta 10 Mbps y se introduce soporte a calidad de servicio. HomePNA 2.0 [Frank y Hollaway, 2000] emplea el rango de 4–10MHz utilizando a nivel físico *Frequency Diversity Quadrature Amplitude Modulation* (FDQAM) que aumenta la robusted de las transmisiones introduciendo redundancia a QAM, a nivel MAC se sigue empleando CSMA/CD, como en las redes IEEE 802.3, pero se introducen ocho niveles de prioridad y un nuevo algoritmo para mejorar la resolución de colisiones denominado *Distributed Fair Priority Queuing* (DFPQ), de esta forma se garantiza un ancho de banda a las aplicaciones y se reduce la latencia que presenta el nivel MAC original de IEEE 802.3.

La alianza ha finalizado recientemente la nueva versión del estándar HomePNA 3.0, en la que se alcanzan tasas de hasta 128 Mbps con extensiones adicionales que permiten hasta los 240 Mbps. La capa física empleada en esta versión está basada en la de la versión 2.0, haciéndolas totalmente compatibles e interoperables.

Hay que destacar que HomePNA fue concebido para la conexión en red de PCs en un hogar y posteriormente para interconectar también a la red televisiones, videos y dispositivos de entretenimiento, no ha sido diseñado para interconectar otro tipo de electrodomésticos de consumo.

HomePlug

El uso de la red eléctrica como medio de transmisión de datos nos permite interconectar en red casi todos los aparatos del hogar, ya que la mayoría de ellos están conectados a ella para obtener la energía necesaria para funcionar. Debido a la variabilidad en frecuencia de la señal eléctrica, éste es un medio complicado para emplear como transmisor de datos, sobre todo si se quieren conseguir unas

tasas elevadas.

En los últimos años han aparecido una serie de iniciativas, la mayoría propietarias, que emplean la red eléctrica para formar redes de control de aparatos eléctricos, y por lo tanto no es necesario alcanzar altas tasas de transmisión. Las tecnologías que más han destacado son:

- **X-10** [X10, 1997], fue diseñado en Escocia entre los años 1976 y 1978 con el objetivo de transmitir datos por las líneas de baja tensión a muy baja velocidad (60 bps en EEUU y 50 bps en Europa) y costes muy bajos. Existen tres tipos de dispositivos X-10: los que sólo pueden transmitir órdenes, los que sólo pueden recibirlas y los que pueden enviar/recibir éstas. Los transmisores pueden direccionar hasta 256 receptores. Se puede decir que a día de hoy es el protocolo más usado en aplicaciones domóticas.

- **LonWorks** [Echelon, 1999], fue presentada por Echelon en 1992 para implementar redes de control distribuidas y automatización. Es una arquitectura descentralizada, extremo a extremo, que permite distribuir la inteligencia entre los sensores y los actuadores instalados en la vivienda y que cubre desde el nivel físico al nivel de aplicación de la mayoría de los proyectos de redes de control. LonTalk es el protocolo que proporciona los servicios de transporte y routing extremo a extremo. Debido al coste de la tecnología se ha implantado más en entornos empresariales que en el hogar.

- **CEBus** [Wacks, 2002], fue presentada por *Electronics Industry Association* (EIA) Americana en 1992 como un estándar abierto para su uso en redes de control. CEBus usa una modulación en espectro ensanchado obteniendo una velocidad media de transmisión de 7500 bps y a nivel MAC es similar a IEEE 802.3. Como parte de la especificación se ha definido también CAL, un lenguaje a nivel aplicación para la comunicación entre dispositivos. En Europa existe una iniciativa equivalente denominada *European Home System* (EHS).

En Marzo de 2000 se forma *HomePlug Powerline Alliance*[7], una asociación de las principales compañías del sector del hogar que se unen para crear un estándar de redes de alta velocidad utilizando la red eléctrica, para su uso en el hogar. El objetivo es permitir que los consumidores que empleen esta tecnología puedan compartir periféricos, conexiones a Internet, distribución de audio/video, así como otras aplicaciones de entretenimiento [Gardner et al., 2000].

[7]http://www.homeplug.org

El grupo decidió apoyar la tecnología PowerPacket de Intellon como base para la primera versión del estándar. PowerPacket utiliza OFDM en el rango de frecuencia entre 4.3 MHz–20.9MHz y emplea CSMA/CD a nivel MAC alcanzando una tasa de datos nominal de 14 Mbps. Para dar soporte a tráfico de voz y multimedia se introduce un protocolo multinivel que permite introducir prioridades a determinados paquetes de datos. Intellon considera que empleando su técnica de modulación se podrán alcanzar tasas de datos de hasta 100 Mbps. PowerPacket ha sido diseñado para no interferir con los protocolos de control que emplean la red eléctrica, como los descritos anteriormente.

HomeCNA

Home Cable Network Alliance[8] es una agrupación de empresas, formada en 2001, que pretende promover la utilización de cable coaxial para su utilización como medio de transmisión dentro del hogar. El trabajo del grupo hasta la actualidad ha sido proponer el rango de frecuencias que emplearán las diferentes aplicaciones que utilizan el coaxial como medio de transmisión, para de esta forma fomentar la utilización de este soporte en redes del hogar.

El grupo tiene como objetivo definir un estándar común teniendo en cuenta algunos de los estándares claves que ya existen en el mercado, entre estos destacan:

- La norma [IEEE Std 1394-1995, 1996], comercialmente conocida como FireWire, define un interfaz digital de bajo coste que integra productos de entretenimiento, comunicaciones y computación en una misma red multimedia, que opera sobre cables de fibra o cobre. Permite comunicaciones peer-to-peer a velocidades de 100, 200 o 400 Mbps y da soporte a transmisiones de datos síncronos y asíncronos. Para desarrollar aplicaciones interoperables sobre redes IEEE 1394 se han definido una arquitectura denominada Home Audio/Video Interoperability (HAVi) [Lea et al., 2000]. La empresa Broadband Home Inc, perteneciente a HomeCNA, propone una extensión propietaria de la tecnología IEEE 1394 sobre cable coaxial, como estándar para la interconexión de productos de audio y video en el hogar.

- La especificación [Cable Home 1.0, 2002] es una iniciativa de CableLabs, asociación de operadores de televisión por cable de EEUU. Su objetivo fundamental es definir una arquitectura que permita garantizar servicios con una determinada calidad en redes del hogar basadas en cable. Para ello se define un modelo físico y un modelo lógico de red. En [Edens, 2001] se encuentra una descripción detallada de ambos modelos.

[8] http://www.homecna.org

2.1.5. Resumen

En esta sección hemos repasado, en primer lugar, el estado actual de las redes inalámbricas de área extensa, el gran éxito que han tenido las redes de segunda generación ha provocado que se estén realizando importantes esfuerzos en la estandarización y despliegue de las nuevas redes de tercera generación. La principal ventaja que le ofrecen al usuario es el acceso a servicios tradicionales de voz y datos, pero en cualquier momento y desde cualquier lugar. Los usuarios acceden a estas redes mediante teléfonos móviles y el servicio principal es el de voz, los servicios de datos son empleados de manera más marginal, aunque se espera que gracias a servicios multimedia este tráfico supere al de voz.

En cuanto a las redes de área local inalámbricas, hemos revisado el estado actual de las redes IEEE 802.11 y de HIPERLAN, aunque claramente son las primeras las que más se han extendido en el mercado en los últimos años. En general, estos protocolos son empleados en organizaciones para el acceso inalámbrico a través de ordenadores portátiles a redes de área local. En la actualidad, algunos operadores de telefonía están ofreciendo este tipo de acceso inalámbrico en espacios públicos como aeropuertos y cada vez más dispositivos personales, como PDAs de gama alta, introducen esta tecnología, por lo que se están convirtiendo en claros competidores de los sistemas de telefonía móvil de tercera generación para los servicios de datos.

Respecto a los protocolos denominados de área personal, IrDA y Bluetooth, la tendencia actual es que los dispositivos personales proporcionen ambos y que dependiendo de la aplicación concreta o las preferencias del propio usuario se emplee uno de ellos.

El entorno del hogar es uno de los primeros en los que se ha aplicado el concepto de computación ubicua y por ello, hemos realizado una revisión de los protocolos que se han definido para estos entornos. Hemos visto que son en su mayoría cableados con la idea de reutilizar el cableado existente en el hogar (línea eléctrica y telefónica) siendo así su implantación menos costosa e invasiva, aunque a pesar de ello su introducción en los hogares no ha sido muy significativa. El fuerte despliegue de protocolos inalámbricos de área local y personal, hace pensar que muchos de ellos se implantarán en el entorno del hogar desplazando a algunas de estas soluciones, manteniéndose sólo los protocolos cableados para redes de control.

Después del estudio realizado, parece claro que la tendencia es que los dispositivos que se embeban en el mundo físico empleen protocolos inalámbricos para comunicarse con los elementos que les rodean, fundamentalmente por los importantes avances que se ha realizado en esta tecnología y porque su implantación

es poco invasiva al no precisar cableado y cada vez menos costosa debido a su popularización. El uso de estos protocolos permite de manera flexible la creación de entornos de computación ubicua, pero impone ciertas restricciones (calidad cambiante, conectividad no transitiva, coste de las comunicaciones, ...) que es necesario tener en cuenta a la hora de proporcionar soluciones eficientes a alto nivel, además se debe tener en cuenta que cada más dispositivos soportan varios de estos protocolos.

El escenario de aplicación en el que nos centramos en esta tesis doctoral involucra fundamentalmente a dispositivos personales, que emplean los usuarios para interaccionar con otros dispositivos personales o de función específica existentes en el entorno próximo que les rodea. Por lo tanto, los protocolos empleados son principalmente los de área local y personal en modos de operación distribuido, de forma que los dispositivos pueden formar una red de forma espontánea (red ad-hoc). El hecho de que un dispositivo pueda tener conectividad global, gracias a protocolos de área extensa, o que pueda controlar otros dispositivos, gracias a protocolos de control tipo X-10, será desde nuestro punto de vista un servicio específico proporcionado por ese dispositivo y que puede ofrecer a los demás dispositivos que se encuentran a su alrededor.

2.2. Dispositivos

Los avances que ha realizado la microlectrónica en los últimos años en tres aspectos fundamentales:

- la miniaturización, consiguiendo embeber un mayor número de transistores en una pieza dada de silicio, [Moore, 1965],
- la evolución de las tecnologías de memoria, que permiten mayor capacidad en chips más pequeños,
- el desarrollo de baterías más pequeñas que duran más y que son menos pesadas,

nos abre la posibilidad de que cada vez más dispositivos tengan capacidad de procesamiento y por lo tanto, que se puedan integrar en el mundo de la computación distribuida para ofrecerles a los usuarios nuevos servicios hasta ahora no proporcionados.

Las limitaciones de estos dispositivos hacen que una gran mayoría de ellos no sean dispositivos multipropósito y que el desarrollo de aplicaciones se tenga que

realizar a bajo nivel y en lenguajes propietarios, lo que unido a sus propias limitaciones, dificulta su integración en las soluciones tradicionales de computación distribuida. Gracias a la aparición de [Java 2 Micro Edition, 1999] se ha abierto un importante camino para poder homogeneizar los desarrollos en los diversos tipos de dispositivos. En esta sección realizaremos una introducción a esta nueva tecnología.

2.2.1. J2ME: Java 2 Platform, Micro Edition

En 1999, Sun Microsystems anuncia la aparición de Java 2 Platform, Micro Edition (J2ME) con el propósito de permitir que aplicaciones Java se ejecuten en dispositivos con potencia de procesamiento limitada, como teléfonos móviles, *pagers*, *palm pilots*, *set-top boxes*, y otros. Una solución que responde a la amplia difusión que están teniendo estos dispositivos en los últimos años y a la demanda de usuarios y proveedores de servicios de recibir/ofrecer nuevas aplicaciones para aumentar las funcionalidades que aportan estos pequeños dispositivos. J2ME conserva, siempre que las limitaciones de los dispositivos lo permiten, compatibilidad con la versión J2SE, de manera que se eliminan las clases o paquetes que no es posible migrar y se definen unos nuevos adaptados a la funcionalidad concreta de un rango de dispositivos.

J2ME en la actualidad abarca dos categorías de dispositivos. Por un parte los que se denominan fijos, que poseen conexiones a la red fija y tienen una capacidad de almacenamiento del orden de 2 a 16 megabytes. Por ejemplo, *set-top boxes*, Internet TV y sistemas de navegación de automóvil. Y por otra parte, los dispositivos denominados móviles, que tienen capacidades de almacenamiento limitadas del orden de los 128 kilobytes, con microprocesadores del 16 ó 32 bit RISC/CISC y que se comunican a través de conexiones inalámbricas. Dentro de esta categoría están los teléfonos móviles, *palm pilots* y *pagers*.

Una de las principales ventajas de J2ME es que es una arquitectura modular, Figura 2.1, que se adapta a las limitaciones de los diferentes dispositivos en los que se quiere integrar. La arquitecura J2ME define tres capas:

- **Máquina virtual**. En la actualidad J2ME soporta dos máquinas virtuales: la Java Virtual Machine que se emplea en ediciones J2SE y en J2EE para los dispositivos con procesadores de 32 bit, y la [KVM, 2000] para arquitecturas de 16/32 bits pero con capacidades de almacenamiento limitado.

- **Configuraciones**. Definen una serie de bibliotecas Java que están disponibles para un conjunto de dispositivos, con similares capacidades de proce-

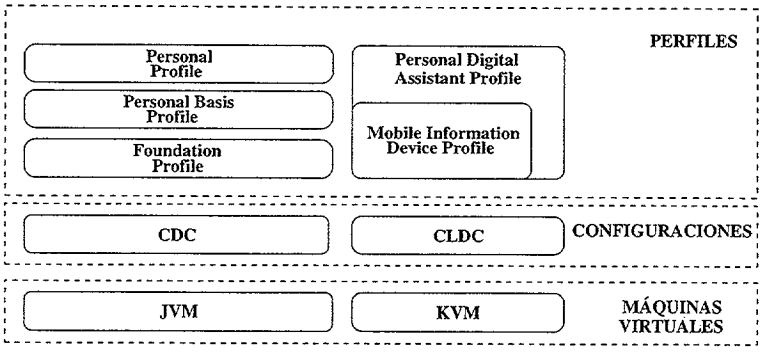


Figura 2.1: Arquitectura J2ME.

samiento y memoria. J2ME soporta varias configuraciones, en la actualidad existen dos estandarizadas:

- **Connected, Limited Device Configuration** (CLDC) [CLDC, 2000], que engloba en general a dispositivos personales móviles.
- **Connected Device Configuration** (CDC) [CDC, 2002], que engloba en general a dispositivos fijos. Por motivos de compatibilidad es un superconjunto de CLDC.

Ambas configuraciones tiene clases comunes con la J2SE, que permite la compatibilidad, pero poseen además clases específicas para los tipos de dispositivos para los que se definieron (ver Figura 2.2).

- **Perfiles**. Definen un conjunto de APIs que pueden emplearse para desarrollar aplicaciones para una familia particular de dispositivos. El principal objetivo en la definición de un perfil es garantizar la interoperabilidad de las aplicaciones entre un conjunto de dispositivos que soportan el mismo perfil. Un mismo dispositivo puede soportar diferentes perfiles y los perfiles se desarrollan para una determinada configuración. Ver Cuadro 2.5.

  Sobre CLDC se han estandarizado:

  - **Mobile Information Device Profile** (MIDP) [MIDP, 2000] para teléfonos móviles, *pagers* y PDAs de bajo coste.
  - **Personal Digital Assistant Profile** (PDAP) [PDAP, 2003], para asistentes personales. La diferencia añadida respecto al anterior es la parte gráfica que es menos limitada.

  Sobre el CDC se han estandarizado:

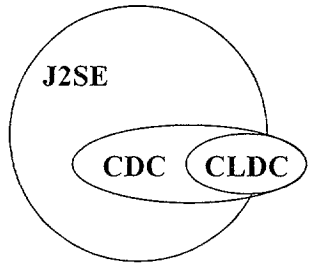


Figura 2.2: Relación entre CLDC, CDC y J2SE.

- **Foundation Profile** (FP), [FP, 2002] para dispositivos embebidos con CPU de 32 bit y sin interfaz de usuario. Este perfil es la base del *Personal Basis Profile.*
- **Personal Basis Profile** (PBP), [PBP, 2002] para dispositivos embebidos con interfaz gráfica limitada. Está construido sobre el *Foundation Profile* y es un subconjunto del *Personal Profile.* Proporciona un subconjunto de AWT, soporte a JavaBeans y un nuevo modelo de aplicaciones: Xlets.
- **Personal Profile** (PP), [PP, 2002] para dispositivos embebidos pero con una interfaz gráfica mucho más rica que el anterior, este perfil está pensado para *set-top box* y televisión interactiva. Está construido sobre los anteriores e introduce más clases de AWT y soporte a applets.

Ligadas a la arquitectura J2ME, existen una serie de APIs que se han ido definiendo para cubrir funcionalidades específicas de determinados tipos de dispositivos. La mayoría de ellas pueden emplearse como extensión a cualquier perfil J2ME porque se han desarrollado sobre la parte común de CLDC y CDC. A continuación, describimos brevemente algunos de ellos, en [Ortiz, 2002] se puede encontrar una tutorial más detallado:

- [Java APIs Bluetooth, 2002] que permite el uso de conexiones Bluetooth y da soporte al protocolo *Object Exchange* (OBEX).
- [Wireless Messaging API, 2003] que proporciona un API para el envío y recepción de *Short Messaging Service* (SMS).
- [Mobile Media API, 2003] que da soporte a procesado multimedia.

Dentro de *Java Community Proccess*[9] (JCP) también se están definiendo paquetes opcionales unidos a determinadas configuraciones, como son soporte a

[9]Comunidad encargada de la especificación de nuevos APIs y tecnologías Java

aplicaciones seguras (*Security and Trust Services API*), a aplicaciones basadas en localización (*Location API*) y a aplicaciones basadas en el protocolo SIP (Session Initiation Protocol) (*SIP API*).

2.2.2. Resumen

Después del estudio realizado en esta sección, vemos que existe una clara relación entre la clasificación de dispositivos propuesta en la sección 1.1 con la arquitectura J2ME. Podemos decir que nuestros dispositivos personales se corresponden con la configuración CLDC y los dispositivos de función específica con la configuración CDC. En principio, J2ME no da soporte a los dispositivos más limitados, que nosotros hemos denominado sensores y actuadores, podríamos pensar que en estos dispositivos es difícil llegar a introducir una máquina virtual Java, pero teniendo en cuenta que existe la especificación Java Card parece que, aunque no existe ninguna iniciativa en curso, es viable su realización en el futuro.

Java Card [Chen, 2000] es una iniciativa anterior a J2ME, y consiste en introducir la tecnología Java en una tarjeta inteligente. Las tarjetas inteligentes suelen tener CPUs de 8 bits, RAM de tan sólo 512 bits y memoria no volátil, EEPROM, de 24KB. El API que proporciona Java Card es limitado y muy adaptado a la funcionalidad tradicional de las tarjetas inteligentes que es la seguridad. Del mismo modo creemos que se podrán desarrollar iniciativas semejantes para *smart labels*, sensores de temperatura, actuadores,...

La existencia de J2ME y el gran apoyo que esta iniciativa está teniendo entre los fabricantes de dispositivos, facilita la posibilidad de realizar desarrollos portables a una gran variedad de dispositivos de distintos tipos. En esta tesis doctoral se empleará esta tecnología para analizar la viabilidad de implementación en dispositivos actuales de las tecnologías middleware definidas.

2.3. Protocolos de descubrimiento de servicios

En entornos de computación ubicua es imprescindible proporcionar un mecanismo de descubrimiento y anuncio de servicios, debido al dinamismo de los entornos y al mayor número y diversidad de dispositivos que aparecen y que pueden ofrecer y demandar servicios. Existen varias soluciones planteadas para el descubrimiento y anuncio de servicios, algunas de las propuestas están ligadas a un protocolo de red, otras se definen sobre IP, y otras forman parte de la arquitectura de un *framework* de desarrollo de servicios en entornos dinámicos, por lo que su utilización está muy ligada a la forma en la que se desarrollan y se

Configuración	Perfil	Dispositivo	Requisitos
CLDC	MIDP	Teléfonos móviles, pagers, PDAs de bajo coste	256KB ROM 128KB RAM
	PDAP	PDAs que gestionan información personal y que se sincronizan con PC u otras PDA	1MB (ROM + RAM)
CDC	FP	Dispositivos embebidos sin interfaz de usuario.	1MB ROM 512KB RAM
	PBP	Dispositivos con interfaz de usuario sencilla, electrónica de consumo del hogar lavadores, neveras ...y de oficina, faxes, impresoras ..., dispositivos de automóvil. Construido sobre el FP.	2MB ROM 1MB RAM
	PP	Dispositivos con una interfaz gráfica más compleja, como *set-top boxes*, televisiones, PDAs de alto coste. Construido sobre el PBP	2.5MB ROM 1MB RAM

Cuadro 2.5: Configuraciones y perfiles J2ME.

Funcionalidad	SDP	SLP	SSDP	DNS-SD	Salutation
Descubrimiento de servicios	√	√	√	√	√
Anuncio de servicios			√		√
Registro de servicios	√	√	√		√
Interoperabilidad	√	√		√	√
Seguridad		√		√	

Cuadro 2.6: Funcionalidad de los protocolos de descubrimiento de servicios.

proporcionan estos servicios.

En esta sección realizaremos una introducción a los más importantes protocolos de descubrimiento de servicios para ver sus principales características y su estado de desarrollo actual. En concreto, vamos a describir SDP de Bluetooth, los protocolos definidos en Internet y Salutation. En la Cuadro 2.6 podemos ver una comparativa de las distintas funcionalidades que ofrecen cada uno de ellos. También hemos incluido en este apartado la descripción de cómo se realiza el descubrimiento de servicios en algunos entornos de desarrollo de servicios para entornos dinámicos, como Jini, OSGi y JXTA.

2.3.1. Bluetooth Service Discovery Protocol

El SIG Bluetooth además del protocolo de comunicación propiamente dicho ha definido una serie de protocolos a nivel aplicación que facilitan el desarrollo de aplicaciones Bluetooth. Entre ellos se encuentra el *Service Discovery Protocol* [SDP, 2001]. Este protocolo le permite a un dispositivo Bluetooth conocer cuáles son los servicios que proporcionan los dispositivos con lo que tiene conectividad. Este protocolo sigue un esquema clásico cliente/servidor en el que la descripción de los servicios se almacena como un *service record* en el dispositivo que actúa como servidor SDP, que, normalmente se corresponde con el que ejerce de maestro en la *piconet* Bluetooth. SDP soporta búsquedas por clases de servicios, y por atributos de servicios, la distinción entre clases y atributos no está bien definida por lo que es necesario que los fabricantes se pongan de acuerdo en estas definiciones para que este protocolo sea interoperable a nivel práctico.

SDP sólo permite el descubrimiento de servicios y no entra en cómo se accede a ellos, se considera que se debe definir en los protocolos de más alto nivel, que hacen uso de SDP. Sin embargo, dentro de SDP se define un atributo común a todos los servicios, `ProtocolDescriptionList`, en el que se define la lista de protocolos con los que se puede acceder al servicio.

En Bluetooth a nivel SDP no se realiza ninguna consideración particular sobre seguridad.

2.3.2. Protocolos de Internet

En 1993 el IETF forma el grupo *Service Location Protocol Working Group* (SRVLOC) con el objetivo de definir un protocolo de descubrimiento automático de servicios en redes IP. Su principal resultado ha sido la definición del *Service Location Protocol* (SLP). Dentro de este mismo grupo se presentó el protocolo *Simple Service Discovery Protocol* (SSDP) que forma parte de la iniciativa UPnP y que se propuso como una simplificación a SLP para redes de dispositivos limitados aunque su definición no pasó del estado de draft. En 1999 el IETF forma el grupo *Zero Configuration Networking* (ZeroConf) con el objetivo de definir protocolos con configuración nula que faciliten la formación de redes IP sin infraestructura y sin administración, entre las propuestas del grupo se encuentra la definición de un protocolo de descubrimiento de servicios basado en DNS.

En este apartado describimos brevemente estos resultados aunque realizaremos un estudio más detallado de estos protocolos en el capítulo 3.

Service Location Protocol

Service Location Protocol (SLP) [Guttman et al., 1999b] es el principal resultado del SRVLOC del IETF. Su objetivo es la definición de un protocolo de descubrimiento automático de servicios en redes IP, descentralizado y extensible. Emplea URLs para la descripción de los servicios, y basándose en ellas los usuarios (aplicaciones) pueden conocer qué servicios existen en la red y acceder a ellos.

La infraestructura de SLP consiste en tres tipos de agentes: *User Agents* (UAs), que son los que realizan el descubrimiento de servicios para satisfacer las necesidades que demandan las aplicaciones de los usuarios finales, *Service Agents* (SAs), que son los responsables de anunciar las características y localización de los servicios y *Directory Agents* (DAs), que son los responsables de almacenar información sobre los servicios que se están anunciando en la red. SLP tiene dos modos de funcionamiento, con DAs, en cuyo caso los SAs registran en ellos los servicios que ofrecen y los UAs buscan en ellos los servicios que precisan; o sin DAs, en cuyo caso los UAs envían por multicast peticiones de servicios, a las que los SAs que ofrecen el servicio responden mediante mensajes unicast. En el RFC se comenta que la existencia de DAs mejora sustancialmente las prestaciones del protocolo.

SLP define varios mecanismos para descubrir DAs, el primero de ellos es el modo pasivo, en el que los SAs y los UAs escuchan los mensajes multicast enviados por los DAs en los que anuncian su existencia periódicamente; y el segundo de ellos es el modo activo, en el que los SAs y UAs envían un mensaje multicast o utilizan DHCP para descubrir los DAs existentes, si existe algún DA, los UAs y SAs utilizan comunicación unicast con ese DA para buscar o registrar servicios, respectivamente.

El protocolo sólo proporciona un mecanismo de búsqueda de servicios y en ningún momento aborda cómo los clientes hacen uso de los servicios. Introduce algunos mecanismos de seguridad, sobre todo para garantizar que no se propague información falsa sobre la localización de servicios.

Simple Service Discovery Protocol

Universal Plug-and-Play (UPnP) es una arquitectura propuesta para el descubrimiento, anuncio y uso de servicios en entornos dinámicos centrada en estandarizar protocolos de comunicación basados en XML, para la interacción entre los diferentes elementos de la arquitectura. El *UPnP Forum*[10] es promotor de esta iniciativa, que está liderada por Microsoft.

Simple Service Discovery Protocol (SSDP) [Goland et al., 1999] es el protocolo que se ha definido dentro de UPnP para el descubrimiento dinámico de servicios. SSDP puede operar con o sin un elemento central, denominado *Service Directory* en la red. SSDP está construido sobre HTTP, empleando tanto unicast, HTTPU[11], como multicast, HTTPMU[12]. Cuando un servicio quiere unirse a la red, primero envía un mensaje de anuncio con el fin de notificar al resto de los dispositivos su presencia, si está presente en la red un *Service Directory*, éste registra el servicio anunciado. También existe la opción de enviar un mensaje unicast directamente al *Service Directory*, para que éste registre el servicio. Los mensajes de anuncio contienen una URI que identifica el servicio anunciado y una URL a un archivo XML que proporciona una descripción de dicho servicio. Cuando un cliente quiere descubrir un servicio, puede tanto contactar con el servicio directamente a través de la URL que ha obtenido en alguno de los anuncios que ha escuchado anteriormente, o puede enviar activamente un mensaje de búsqueda de servicio por multicast. En el caso de descubrir un servicio a través de un mensaje de búsqueda, la respuesta puede ser proporcionada por el propio servicio o por un *Service Directory*.

[10] http://www.upnp.org
[11] Variante de HTTP sobre UDP
[12] Variante de HTTP que permite mensajes multicast sobre UDP

SSDP se presentó como un draft al SVRLOC del IETF, con el objetivo de realizar una simplificación de SLP, y que de esta forma se pudiera implementar en un mayor número de dispositivos. En la actualidad este draft está obsoleto.

En UPnP no se aborda la seguridad a nivel SSDP.

DNS-SD: DNS Service Discovery

DNS Service Discovery (DNS-SD) [Cheshire, 2003a] surge con el objetivo de proporcionar soporte al descubrimiento de servicios a través de DNS, de tal manera que sin realizar ningún cambio a DNS, se puedan realizar consultas que permitan al usuario obtener una lista de instancias de un tipo determinado de servicio. En la actualidad se encuentra en estado de *internet draft.*

DNS-SD puede operar sobre DNS, por lo tanto, con una arquitectura centralizada basada en una estructura jerárquica de servidores; o sobre las modificaciones de DNS para la resolución de nombres en redes sin infraestructura, como Multicast DNS o LLMNR, con una arquitectura totalmente distribuida. Estas propuestas se encuentran en definición dentro de varios grupos del IETF.

DNS-SD emplea la sintaxis y semántica de los registros de recurso SRV de DNS para el descubrimiento de servicios, pero añadiendo un nivel de indirección, para que a través de consultas DNS el usuario pueda obtener instancias de tipos de servicios de distintas características. El proceso de descubrimiento se divide en dos pasos, primero se obtienen las instancias de los servicios y posteriormente, para cada instancia se obtienen los datos necesarios para acceder a ella.

La seguridad de DNS-SD se basa en DNSSEC para garantizar la autenticidad de la información proporcionada por los dispositivos que forman la red.

2.3.3. Salutation

Salutation es una arquitectura para el descubrimiento y anuncio de servicios desarrollada por el *Salutation Consortium*[13], agrupación de compañías y centros académicos, que surge con el objetivo de resolver el problema del descubrimiento y acceso a servicios entre un amplio conjunto de dispositivos y equipos en un entorno cambiante, independientemente del protocolo de transporte concreto que se esté utilizando.

Salutation [Miller y Pascoe, 2000] es un estándar abierto independiente de sistemas operativos, protocolos de comunicaciones y plataformas hardware. La

[13]http://www.salutation.org

arquitectura Salutation define tres componentes:

- *Functional Units* que desde el punto de vista de un cliente definen un servicio. Para alguno de los servicios más habituales, como impresoras, faxes o almacenamiento de documentos, el consorcio está definiendo estas unidades de forma que se garantice la interoperabilidad.

- *Salutation Managers* (SLMs) que permiten a los clientes descubrir y comunicarse con los diferentes servicios proporcionados en la red. El proceso de descubrimiento de servicios puede realizarse a través de múltiples SLMs. Un SLM puede descubrir otros SLMs remotos y determinar los servicios que están registrados allí. De esta manera el descubrimiento de servicios se realiza de una manera mucho más rápida.

- *Transport Managers* (TMs) que permiten aislar al SLM del protocolo de transporte que se está empleando para acceder a él. De esta forma, para dar soporte a un nuevo protocolo de transporte sólo es necesario implementar un nuevo TM, sin modificar la implementación del SLM.

Salutation no sólo define mecanismos de descubrimiento, sino que en la especificación también se describen mecanismos para el acceso a los servicios. Este soporte se proporciona a través del SLM que tiene tres modos de operación: *native personality*, en la que sólo se emplea para descubrir servicios y para establecer la comunicación entre los clientes y el servidor; *emulated personality*, que además de establecer la conexión sirve como pasarela entre el protocolo que emplea el cliente y el que emplea el servidor cuando son diferentes; y *salutation personality*, en el que además de establecer la conexión obliga a que la comunicación entre cliente y servidor se realice en un formato determinado.

En cuanto a seguridad, Salutation soporta autenticación de usuario mediante esquemas de identificador y password. Opcionalmente, un servicio puede obtener información registrada de los sistemas que están accediendo a él.

Para dispositivos limitados se ha definido una simplificación del protocolo denominada Salutation-Lite [Pascoe, 1999], que se centra fundamentalmente en dar soporte al descubrimiento dinámico de servicios. En esta versión no sólo se ha tenido en cuenta la limitación de los dispositivos, sino también el limitado ancho de banda que proporcionan los protocolos de comunicación que suelen emplear, como son IrDA o Bluetooth.

2.3.4. Descubrimiento de servicios en Jini

[Jini, 1999] es una arquitectura distribuida orientada a servicios desarrollada por Sun Microsystems, su principal objetivo es convertir a la red en una herramienta flexible y fácilmente administrable en la que los clientes puedan encontrar servicios de un modo robusto y flexible. Jini se apoya fuertemente en Java, de tal forma que en Jini es necesario que todos los dispositivos tengan una máquina virtual Java, o un dispositivo, que sí que la tenga y actúe en su nombre.

En la arquitectura Jini se definen tres componentes: los servicios, los clientes y los servicios de directorio, que se denominan *Jini Lookup Services* (JLS). Un servicio en Jini está representado por un objeto Java y se define como una entidad software que proporciona algún tipo de cálculo o control sobre un dispositivo. Un cliente es aquel que hace uso de un servicio, y por lo tanto un servicio puede ser a su vez cliente de otro.

El JLS es obligatorio dentro de la arquitectura de Jini, y los clientes y los servicios siempre se descubren a través de él y nunca de forma directa. Para registrar o buscar un servicio primero es necesario localizar algún JLS, para ello se han definido tres tipos de protocolos: *unicast discovery protocol*, empleado cuando ya se conoce un JLS, *multicast request protocol* empleado para buscar un JLS y *multicast announcement protocol* empleado por los JLS para anunciar su disponibilidad. Para soportar el dinamismo de la red y que los servicios registrados en el JLS no se queden obsoletos, Jini implementa un mecanismo de *leasing* que obliga a que los servicios actualicen su registro periódicamente para mantener su entrada en un JLS y de esta forma poder ser localizados por los clientes.

En Jini se define no sólo un mecanismo de descubrimiento de servicios, sino también un mecanismo de uso: un cliente le solicita a un JLS la búsqueda de un servicio proporcionándole un identificador del servicio, un interfaz Java o un conjunto de atributos, el JLS responde a esta búsqueda con un conjunto de objetos *proxies* que permitirán al cliente acceder a implementaciones de dicho servicio. El mecanismo de comunicación entre clientes y servicios se basa en *Java Remote Method Invocation* (RMI).

En cuanto a seguridad, Jini depende del modelo de seguridad de Java.

2.3.5. Descubrimiento de servicios en OSGi

Open Services Gateway Initiative (OSGi[14]) fue creada en Marzo de 1999 con el objetivo de definir una especificación software abierta que permita diseñar y

[14]http://www.osgi.org

construir plataformas compatibles que sean capaces de proporcionar múltiples servicios en redes locales en general, aunque su uso actual está muy centrado en redes del hogar. Aunque OSGi define su propia arquitectura se ha tenido en mente su compatibilidad con otras iniciativas semejantes como son Jini o UPnP.

La arquitectura de OSGi [Marples y Kriens, 2001] tiene dos elementos fundamentales, por un parte una plataforma de ejecución de aplicaciones basada en Java y por otra, una serie de paquetes (*bundles*) que proporcionan una determinada funcionalidad a otros paquetes o directamente al usuario final y que se ejecutan sobre la plataforma. Estos elementos residen siempre en un elemento central que se denomina plataforma de servicio OSGi situada en la red local y conectada al proveedor de servicios a través de una pasarela en la red del operador. Este elemento además será el encargado de permitir la interacción entre dispositivos o redes de dispositivos que empleen distintas tecnologías para comunicarse.

En OSGi un servicio está proporcionado por un *bundle* que se ejecuta en la plataforma OSGi. En la plataforma existe el *OSGi service registry* que actúa como un servicio de directorios en el que los *bundles* se registran y a través del cual pueden localizar a otros *bundles* para componer nuevos servicios.

En la especificación de OSGi se han definido una serie de APIs básicas para el desarrollo de servicios, como el de logging, servidor HTTP y el que se denomina *Device Access Specification* (DAS) que permite el descubrimiento y anuncio dinámico de dispositivos y de los servicios ofrecidos por éstos. En DAS se definen los *Network Bundles* que son los encargados de descubrir nuevos dispositivos y servicios empleando el protocolo de descubrimiento correspondiente en esa red, una vez obtenida esta información, deben obtener del proveedor de servicios el *Device Bundle* correspondiente al dispositivo concreto, que se instalará en la plataforma y se registrará en *OSGi service registry* y de esta forma podrá ser descubierto por otros *bundles*.

Mediante este mecanismo OSGi permite la integración de cualquier protocolo de descubrimiento y anuncio de servicios dentro de su plataforma. En [Dobrev et al., 2002] se muestran dos ejemplos detallados de integración de Jini–OSGi y de UPnP–OSGi.

En cuanto a la seguridad, OSGi define un servicio específico en la plataforma que gestiona los permisos asociados con cada uno de los *bundles* existentes en la plataforma. De manera opcional, puede emplearse mecanismos de seguridad basados en el paquete Java `java.security`.

2.3.6. Descubrimiento de servicios en JXTA

El proyecto JXTA[15] es una plataforma de computación distribuida *peer-to-peer* (P2P) concebida y promovida inicialmente por Sun Microsystems a principios de 2001, pero en la que en la actualidad participan un gran número de centros de investigación académicos e industriales. JXTA proporciona una serie de tecnologías middleware por encima de las cuales se pueden construir servicios y aplicaciones. El objetivo principal del proyecto es crear un soporte software para sistemas P2P que permita la interoperabilidad entre diferentes implementaciones de un mismo servicio, la independencia de la plataforma, del lenguaje y entorno de programación en el que se desarrolla el servicio y la ubicuidad de los servicios, permitiendo incluir en el sistema desde los grandes servidores hasta los dispositivos más limitados como PDAs o electrónica de consumo.

Para alcanzar este objetivo JXTA [Gong, 2001] ha definido una serie de protocolos básicos, entre ellos se encuentra el *Peer Discovery Protocol* que estandariza el formato de mensajes que permite a un *peer*, o elemento del sistema, encontrar a otros *peers*, servicios, etc. Este protocolo emplea a su vez el *Peer Resolver Protocol* que es un protocolo genérico que permite enviar y recibir peticiones de búsquedas. A su vez este protocolo emplea el *Rendezvous Protocol* que está basado en que existe un *peer* especial en el sistema que guarda información sobre los *peers*, servicios, etc. que conoce, de tal forma que puede proporcionar esta información a otro *peer* que establezca una comunicación con él.

El *Peer Discovery Protocol* es el protocolo de descubrimiento por defecto que debe implementarse obligatoriamente en todos los sistemas JXTA. Además en el prototipo de implementación de la versión 1.0 se han desarrollado otros mecanismos opcionales para el descubrimiento, que son los siguientes:

- *LAN-based*, en el que el descubrimiento se realiza mediante mensajes broadcast.
- *Invitation*, en el que un *peer* proporciona información del sistema a otro *peer* sin solicitud previa de éste.
- *Cascade*, en el que los *peers* se van proporcionando información sobre los otros *peers* que conoce, previa autorización de estos.

En cuanto a la seguridad, la tecnología JXTA soporta un conjunto básico de algoritmos criptográficos utilizando una versión modificada del paquete

[15] http://www.jxta.org

`java.security`. Los servicios proporcionados son integridad y privacidad de mensajes, autenticación y protección de denegación de servicio.

Dentro del proyecto JXTA, existe la iniciativa JXME para introducir la tecnología JXTA en dispositivos limitados J2ME, la implementación se basa en que estos dispositivos tienen un JXTA *relay* asociado, normalmente un PC próximo, que actúa en su nombre y que le permite integrarse en la red P2P JXTA.

2.3.7. Pasarelas entre protocolos

A día de hoy la implantación de estos protocolos no ha sido muy amplia, por lo que ninguno de ellos se ha situado como un estándar de facto para el descubrimiento dinámico de servicios. Esto ha llevado a una necesidad mucho más acuciante de conseguir que estos protocolos interoperen entre sí. Un ejemplo de ello, es la especificación OSGi en la que ya se ha tenido en cuenta esta heterogeneidad para integrar los diferentes protocolos existentes dentro de la plataforma. Otras de las propuestas que se han realizado en este sentido han sido:

- **Jini–SLP** [Guttman y Kempf, 1999], en el que se ha creado un UA–SLP especial que registra los SA–SLP con capacidad Jini.
- **Salutation–SDP Bluetooth** [Miller y Pascoe, 1999] , en el se consideran dos aproximaciones, la primera mapea los APIs de Salutation a los de SDP–Bluetooth para implementar Salutation sobre SDP; la segunda utiliza un *Transport Manager* Bluetooth y reemplaza SDP por Salutation.

2.3.8. Resumen

En esta sección hemos realizado un breve estudio de las diversas propuestas existentes para el descubrimiento de servicios de forma dinámica. En primer lugar hemos visto SDP, un protocolo de descubrimiento ligado al protocolo de red Bluetooth, después hemos analizado las diferentes propuestas realizadas en el IETF: SLP, SSDP y la reciente alternativa basada en DNS, DNS-SD y posteriormente, la iniciativa Salutation. Estas soluciones son genéricas e independientes de cómo se han desarrollado los servicios, pero el problema del descubrimiento también se ha cubierto en propuestas más genéricas en las que se propone un middleware para el desarrollo de servicios en entornos de computación distribuida, y en las que, como parte de la arquitectura, se ha definido un mecanismo de descubrimiento de servicios. Así, hemos visto como se aborda el descubrimiento de servicios en Jini, OSGi y JXTA.

Aunque las soluciones anteriores han tenido en cuenta el dinamismo del entorno provocado por la movilidad de los dispositivos que ofrecen y demandan servicios, la mayoría no han considerado las limitaciones de los nuevos dispositivos, ni la posibilidad de que las redes ad-hoc que se forman entre ellos de forma espontánea, no incluyan ningún elemento fijo sin limitaciones, tipo PC. Así, en la mayoría de ellas aparece un servidor central en el que los que ofrecen servicios se registran y al que los clientes preguntan cuando quieren encontrar un determinado servicio. En entornos de computación ubicua las redes son muy dinámicas y los dispositivos tienen capacidades de almacenamiento limitadas, por lo que es muy costoso que uno de ellos afronte el papel de servidor.

Las soluciones distribuidas, que eliminan el repositorio centralizado de servicios, emplean dos mecanismos básicos para que clientes y servidores se descubran:

- Método *push*: en el que los servicios se anuncian periódicamente y los clientes escuchan y almacenan estos anuncios para seleccionar posteriormente el servicio que les interesa.

  Este mecanismo se emplea en SLP, SSDP, DNS-SD y JXTA.

- Métodos *pull*: en el que los clientes realizan peticiones y los servicios se descubran bajo demanda.

  Este mecanismo se emplea en SSDP y JXTA.

Un factor importante a tener en cuenta en estas soluciones en las que, o los anuncios de los servicios, o las peticiones, se envían a todos los dispositivos que están en el entorno, es que puede suponer un gran gasto de las baterías de los dispositivos debido al número de transmisiones realizadas. Recordemos además, que en entornos tan dinámicos los servicios estarán disponibles un tiempo limitado, por lo que tenemos el compromiso de implementar mecanismos que permitan detectar lo antes posible tanto la disponibilidad como indisponibilidad de esos servicios.

En el capítulo 3 realizaremos un estudio más detallado del problema de descubrimiento de servicios de forma dinámica y justificaremos la necesidad de definir un nuevo mecanismo de descubrimiento que se adapte mejor a los requisitos impuestos por los entornos de computación ubicua.

2.4. Agentes

La tecnología de agentes ha tenido un especial impacto en los últimos años gracias a su aplicación en la computación distribuida. El modelo previo más

ampliamente extendido es el modelo cliente/servidor, en el que un cliente que se ejecuta en un entorno envía un conjunto de datos a un servidor, y espera a que éste le envíe los datos de respuesta de la operación realizada, antes de enviar nuevos datos. Cada mensaje intercambiado en la red implica una petición de un servicio y una respuesta a esa petición. La comunicación establecida precisa de una conexión permanente y esto provoca el consumo de un gran número de recursos.

La introducción de la tecnología de agentes permite realizar las mismas operaciones pero con la ventaja de que sean asíncronas y que además no precisemos conexiones permanentes para la ejecución de tareas, puesto que el agente que migra hacia otro sistema además de llevar los datos necesarios para realizar la operación, conserva la información de estado del proceso. Este concepto se explicará en detalle a lo largo de esta sección.

A pesar de estos beneficios, debemos tener en mente que la tecnología de agentes no sustituye a otros paradigmas de la computación distribuida, una aplicación que se desarrolla con tecnología de agentes puede desarrollarse con las tecnologías convencionales. Lo que debemos analizar es si, para una aplicación determinada, con la tecnología de agentes se obtienen mejores prestaciones, y en ese caso aplicarla. Existen algunos campos de aplicación en los que se ha demostrado que el uso de agentes es mucho más beneficioso: en concreto se han realizado estudios en el caso de acceso a bases de datos [Papastravrou et al., 1999], en tareas de gestión de red [Baldi y Picco, 1998] [Glitho y Magedanz, 2002] y en recolección de datos en redes de sensores [Qi et al., 2001].

Existen dos conceptos fundamentales en los sistemas de agentes móviles, que son el de agente y el de plataforma de agente.

2.4.1. Agentes móviles

Un agente [Nwana, 1996] se define como una entidad software que actúa en nombre de otra entidad, por ejemplo, de una persona o de otro agente, que es autónomo y se comporta según los objetivos que debe alcanzar y que reacciona ante eventos externos y puede comunicarse y colaborar con otros agentes.

Un agente móvil es un agente que puede migrar entre dos nodos de una red. Junto a este tipo de agentes, surgieron también los agentes inteligentes, basados en aplicar conceptos de inteligencia artificial al paradigma de agentes. Un agente móvil puede ser inteligente y a la inversa, pero ambas características son independientes.

Para aclarar la definición de agente móvil es necesario indicar qué es lo que

entendemos por migración de un agente entre dos nodos. Una entidad software en ejecución está compuesta por el código, que nos proporciona la descripción estática de su comportamiento, y su estado, que nos proporciona su descripción en un momento determinado de ejecución. Migrar un agente consiste en enviar su código y el estado de ejecución al host remoto, de forma que después de su migración en el host remoto se retome su ejecución en el mismo punto en el que se encontraba antes de migrar. El agente una vez finalizadas sus tareas en el nodo remoto puede enviar un mensaje para devolver el resultado de su ejecución o volver a migrar al nodo inicial.

2.4.2. Plataformas de agentes

Los agentes móviles requieren un entorno especial para su ejecución. Este entorno es lo que se denomina plataforma. La plataforma además de aportar la capacidad de ejecución propiamente dicha, debe proporcionar una serie de servicios básicos:

- Comunicaciones: que facilite la comunicación de los agentes con el mundo exterior, principalmente, que le permitan interactuar con otros agentes en tareas cooperativas.

- Nombrado: que permite nombrar a los agentes de manera que puedan ser identificados de forma unívoca, y también nombrar a las plataformas a las que migran los agentes.

- Descubrimiento y localización: que facilite a los agentes descubrir otros agentes y plataformas con los que puede interactuar, o a las que puede migrar, para alcanzar los objetivos que tiene encomendados.

- Movilidad: que facilite la migración de agentes entre plataformas remotas.

- Seguridad: que garantice por una parte la protección de los agentes ante ataques del host en el que se ejecuta y por otra, la protección del host ante los ataques de los agentes que se ejecuten en él.

Existen diversas plataformas de agentes móviles entre las que cabe destacar Aglets de IBM [Lange y Oshima, 1998], Voyager de ObjectSpace [Glass, 1999], Grasshopper de IKV [Grasshopper, 1998] y JADE de la Universidad de Parma [Bellifemine et al., 1999].

2.4.3. Beneficios de los agentes móviles

Tradicionalmente, los agentes móviles se han contemplado como una alternativa viable en la que la transmisión del código necesario para realizar una tarea resultaba más económico (en términos de ancho de banda necesario) que la orientación cliente/servidor. En este sentido en [Glitho y Pierre, 2002] se describe una comparativa de sincronizaciones de agendas entre máquinas remotas en tres casos: el primero basado en cliente/servidor, el segundo basado en optimizar el servidor para la operación de sincronización, y el tercero basado en agentes móviles. Las conclusiones del trabajo fueron que, en términos de datos transmitidos, la alternativa de cliente/servidor crecía linealmente con el número de participantes, mientras que las otras permanecían casi constantes y a un nivel muy inferior. Hay que hacer notar que el escenario utilizaba sólo dos servidores que almacenaban las agendas. La segunda conclusión era que el coste de desarrollar el servidor optimizado fue bastante grande, esfuerzo que no fue necesario en el caso de agentes móviles.

Además del menor consumo de recursos de comunicación y del menor coste de desarrollo de aplicaciones concretas, el paradigma de agentes y sistemas multiagentes plantea otros beneficios adicionales, como son la cooperación entre agentes y el diseño de sistemas dirigidos al usuario, en los que el agente personal del usuario cobra una gran importancia. Como última ventaja citaremos la gran diversidad de dispositivos a disposición del usuario, que los agentes móviles pueden integrar para facilitar sus tareas y dar una visión de servicios ubicuos, en lugar de servicios diversos prestados por distinto software corriendo en distintos dispositivos.

2.4.4. Estándar FIPA

Foundation for Intelligent Physical Agents[16] (FIPA) comenzó sus actividades en 1995 con el objetivo de estandarizar aspectos relacionados con la tecnología de agentes y sistemas multiagente. En la actualidad se puede considerar el estándar más ampliamente reconocido y extendido internacionalmente, y se ha convertido en un referente a seguir a la hora de realizar desarrollos basados en agentes.

Las especificaciones FIPA se han agrupado en tres tipos: *component*, que se encargan de estandarizar todas las tecnologías básicas relacionadas con agentes, *informative* que describen posibles soluciones en aplicaciones realizadas con agentes en un determinado dominio y *profiles* que son conjuntos de especificaciones de tipo *component* que permiten validar cuándo una implementación es conforme

[16] http://www.fipa.org

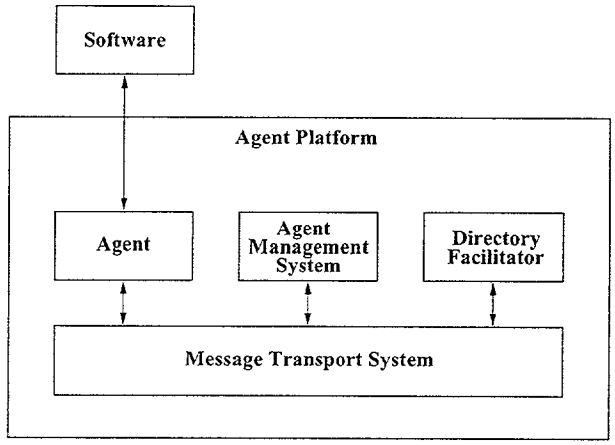


Figura 2.3: Modelo de referencia FIPA.

al estándar.

En *FIPA Agent Management Specification* [FIPA00023, 2002] se describe el modelo de referencia FIPA de plataforma de agentes y se describe la funcionalidad de cada uno de sus componentes, ver Figura 2.3. Éstos componentes son:

- **Agent Management System** (AMS): que gestiona el ciclo de vida de los agentes, los recursos locales, y los canales de comunicación y proporciona un servicio de páginas blancas, que permite localizar agentes por su nombre.
- **Directory Facilitator** (DF): que proporciona un servicio de páginas amarillas, que permite localizar agentes por sus capacidades y no por su nombre.
- **Message Transport System** (MTS): que gestiona el envío de mensajes entre agentes de la misma plataforma o de plataformas distintas, y permite la migración de agentes.

Lenguajes de programación

Una parte importante del trabajo realizado en el desarrollo de plataformas de agentes ha sido proporcionar o adaptar lenguajes de programación para implementar estas plataformas [Cugola et al., 1997]. Se han utilizado lenguajes de propósito más general como pueden ser Java o adaptaciones de otros, como las versiones realizadas sobre Tcl: Safe-Tcl [Borenstein, 1994], Agent Tcl [Kotz et al., 1999], TACOMA [Johansen et al., 1999] y en algunos casos, se han definido lenguajes específicos para este propósito: Telescript [White, 1995], MO [Tschudin, 1994] y Tycoon [Matthes et al., 1994].

La característica principal que deben proporcionar estos lenguajes es permitir la movilidad de una unidad de ejecución, que consiste en una parte de código, que nos proporciona la descripción estática del comportamiento de un programa, y el estado de la unidad de ejecución. El estado contiene por una parte, lo que se denomina espacio de datos, que son todos los recursos accesibles desde todas las rutinas activas y por otra, lo que se denomina estado de ejecución, que es información de control, como el valor del contador del programa y el estado de la pila, que nos permite retomar la ejecución después de la migración. Cada uno de estos componentes pueden moverse independientemente.

Los diferentes lenguajes han aportado soluciones diferentes para la movilidad de estos componentes, agrupándose en dos clases, por una parte los lenguajes que soportan *strong mobility* y los que soportan *weak mobility*:

- Movilidad fuerte (*strong mobility*): se migra el código y estado, abarcando tanto el espacio de datos como el estado de ejecución.
- Movilidad débil (*weak mobility*): se migra el código y el espacio de datos del estado del código, pero no el estado de ejecución.

Aunque existen lenguajes que proporcionan *strong mobility*, como son Telescript, TACOMA y Ara [Peine y Stolpmann, 1999], la mayoría de los lenguajes soportan *weak mobility*, entre ellos Java, que a pesar de esta limitación ha sido el lenguaje en el que más plataformas de agentes se han desarrollado, debido a las siguientes ventajas aportadas por el lenguaje:

- **Independencia de la plataforma**: una de las principales ventajas que introdujo Java, fue poder desarrollar software independiente del procesador. La clave consistió en desarrollar un código neutro que pudiera ser ejecutado sobre una máquina virtual, denominada *Java Virtual Machine*, que es la que interpreta este código neutro convirtiéndolo a código particular de cada CPU o plataforma. Esto nos permite, que cualquier agente desarrollado en Java pueda ejecutarse en un host remoto que tenga una máquina virtual Java.
- **Ejecución segura**: Java fue concebido para su utilización en redes como Internet, por lo que la seguridad ha adquirido una importancia vital en su definición. La arquitectura de seguridad de Java hace que sea relativamente sencillo salvaguardar a un host de un agente malicioso, porque no se permite el acceso directo a los recursos del host, es un lenguaje fuertemente tipado y no existen punteros.

- **Carga dinámica de clases**: este mecanismo permite a la máquina virtual cargar clases en tiempo de ejecución, e incluso se puede habilitar su descarga a través de la red. Esto facilita la migración de agentes de un host a otro, ya que sólo es necesario que migre la clase *root* del agente, las demás clases que se precisen en ejecución, y que no existan en el host destino, serán descargadas dinámicamente, bien desde el host origen o bien desde un repositorio de clases habilitado en la plataforma de agentes para proporcionar este servicio.

- **Programación multihilo**: los agentes por definición son autónomos, es decir, un agente se ejecuta independientemente de los otros agentes que residen en el mismo lugar. El comportamiento autónomo de los agentes se consigue permitiendo que cada agente se ejecute en su propio proceso ligero, también llamado hilo de ejecución. Java permite la programación multihilo y además proporciona un conjunto de primitivas de sincronización que nos permite la interacción entre los agentes.

- **Serialización de objetos**: el lenguaje Java proporciona mecanismos de serialización que permiten representar el estado de un objeto en forma serializada con el suficiente detalle para que este objeto se pueda reconstruir posteriormente.

- **Reflexión**: El código Java permite obtener información sobre los campos, métodos, y constructores de la clases cargadas. Esta información puede emplearse para crear objetos de una clase que se descubre en tiempo de ejecución. Esto permite dotar de cierta inteligencia a los agentes ya que pueden obtener información de si mismos y de otros agentes.

A pesar de todas estas ventajas, el desarrollo de plataformas de agentes con Java tiene ciertas limitaciones, una de las más importante es la que comentamos anteriormente, que sólo soporta *weak mobility*, pero además:

- No proporciona un soporte adecuado para el control de recursos, ya que en Java no se tiene control sobre los recursos consumidos, tiempo de procesador y memoria consumida, por un objeto.

- No proporciona control de referencias, todos los métodos públicos de un objeto Java son accesibles desde cualquier otro objeto que tenga una referencia a él. Esto presenta un problema ya que los agentes no tienen control sobre qué agentes acceden a sus métodos.

2.4.5. Plataformas de agentes y J2ME

Con la aparición de versiones de Java para dispositivos limitados, J2ME, y considerando que el lenguaje de programación en el que más plataformas de agentes móviles se han desarrollado es Java, las posibilidades de desarrollar plataformas de agentes en estos dispositivos se presenta alcanzable.

Para determinar las restricciones que nos impone J2ME a la hora de desarrollar una plataforma de agentes en dispositivos limitados, nos centramos en las limitaciones que posee la configuración CLDC por ser la más limitada y por ser un subconjunto de CDC, con lo que garantizamos que una solución en CLDC también funcionará en CDC. Por lo tanto, las principales limitaciones que nos impone CLDC son:

- Limitaciones del lenguaje Java:
  - No soporta tipos de datos de coma flotante (`float` o `double`).
  - No soporta la finalización de instancias de clases. El método `Object.finalize()` no existe.
  - Limitaciones en el manejo de errores. La mayoría de las subclases de `java.lang.Error` no están soportadas y en general, la gestión de errores es particular de la implementación realizada para el dispositivo concreto.
- Limitaciones de la máquina virtual:
  - No soporta Java Native Interface (JNI), debido a:
    - Modelo de seguridad limitado del CLDC.
    - Limitaciones de memoria necesarias para el soporte completo de JNI.
  - No soporta cargadores de clase definidos por el usuario, debido a restricciones de seguridad.
  - No soporta reflexión, y por lo tanto, ni serialización de objetos, ni soporte a RMI, ni otras características avanzadas de Java (JVM Debugging Interface, JVM Profile Interface).
  - No soporta grupos de `threads` ni `daemon threads`, las operaciones de arranque y parada de threads sólo se pueden aplicar individualmente.

La mayoría de estas limitaciones se deben a las propias limitaciones de procesamiento y memoria de los dispositivos y a razones de seguridad motivadas porque J2ME/CLDC no soporta el modelo completo de seguridad de J2SE. El modelo soportado por J2ME/CLDC es de tipo "sandbox", es decir las aplicaciones se ejecutan sobre un entorno limitado en el que la aplicación sólo puede acceder a las APIs definidas por el CLDC y por los perfiles proporcionados, o a clases específicas del dispositivo. El modelo de seguridad en la última versión de MIDP se ha completado y no es tan restrictivo, de manera que el usuario tiene cierta libertad a la hora definir sus propias restricciones de seguridad sobre las aplicaciones que se están ejecutando, por ejemplo, si está permitido o no que establezcan comunicaciones con sistemas remotos, que empleen la cámara, etc.

En el futuro se pretenden suplir algunas de estas limitaciones, entre ellas la sincronización de threads y el cargador dinámico de clases.

Si recordamos las características que convertían a Java en un buen lenguaje para desarrollar plataformas de agentes, sección 2.4.4, y las comparamos con las restricciones de J2ME, vemos que gran parte de éstas han desaparecido, o se han visto limitadas por motivos de seguridad, en concreto, aquellas que nos permitían implementar movilidad de objetos (carga dinámica de clases, serialización y reflexión). Veremos a lo largo de esta tesis doctoral que estas limitaciones pueden superarse de manera que el desarrollo de una plataforma de agentes sobre J2ME es abordable.

2.4.6. Resumen

La investigación en torno al paradigma de agentes móviles se ha centrado fundamentalmente en el desarrollo de plataformas de agentes, y en los retos que plantea en sí la movilidad de código. Han sido pocas las investigaciones que se han centrado en demostrar la validez del desarrollo de servicios empleando agentes móviles y la mejora que supone su utilización respecto al paradigma clásico cliente/servidor. Además, su implantación en productos comerciales se ha visto frenada fundamentalmente debido a los problemas de seguridad que plantea la movilidad de código, aunque en la actualidad existen varias líneas de investigación abiertas para solucionar este problema.

Los agentes se caracterizan por estar orientados a realizar tareas, por ser autónomos, por su capacidad de cooperar con otros sistemas y si poseen la característica de movilidad, por ser capaces de moverse a los sistemas remotos para realizar sus tareas y de esta forma minimizar el coste de las transmisiones. Estas características se adaptan a las limitaciones indicadas anteriormente y por las que se caracterizan los entornos ubicuos: entornos cambiantes, coste de las

comunicaciones y limitaciones de los dispositivos.

Los desarrollos llevados a cabo a nivel agentes se realizaron para redes como Internet, las plataformas se ejecutaban en PCs sin limitaciones en cuanto a su capacidad de proceso y comunicación, y aunque en su definición se consideraba que se adaptaban a entornos cambiantes, con conectividad intermitente y calidad cambiante, la realidad es que estos estados eran tratados más como excepciones a las que el sistema sabía adaptarse, que como las características habituales del entorno en el que se ejecutaban. Por ello, no es viable migrar los desarrollos realizados hasta la actualidad a los sistemas limitados que operan en entornos ubicuos, sino que es necesario volver a diseñarlos.

En esta tesis doctoral se propone emplear el paradigma de agentes móviles como tecnología middleware para el desarrollo de servicios en entornos ubicuos, para ello es necesario realizar un diseño de plataforma de agentes que se adapte a las limitaciones que presentan estos entornos. En el diseño tomaremos como punto de partida el estándar FIPA e intentaremos, con el mínimo impacto en la especificación, adaptarlo a los nuevos requisitos. Veremos en el capítulo 6 que nuestra contribución en esta línea se ha centrado en la adaptación del servicio de páginas amarillas definido en FIPA, el DF, para que opere en entornos ad-hoc y que nos permita descubrir los servicios ofrecidos por agentes remotos de una forma eficiente. Como tecnología para apoyar la viabilidad de implantación de nuestro diseño, utilizaremos la versión de Java para dispositivos limitados, J2ME.

2.5. Trabajos relacionados

La investigación en computación móvil y ubicua a nivel de aplicación es bastante reciente, ya que los principales retos hasta el momento estaban fundamentalmente en el campo de la microlectrónica y de los nuevos protocolos de red, principalmente inalámbricos. Además de proyectos precursores como ParcTab de Xerox Park [Schilit et al., 1993], en estos últimos años han aparecido los primeros resultados de algunos otros. En este apartado describimos brevemente aquellos proyectos que están relacionados directamente con la tesis doctoral propuesta, y cuyo resultados han de ser tenidos en cuenta para contrastarlos con nuestras contribuciones. En el Cuadro 2.7 se presenta de forma resumida los objetivos y retos que se abordan en cada uno de los proyectos.

Recientemente, también han aparecido algunas plataformas de agentes que se ejecutan en dispositivos limitados basados en J2ME, aunque ninguna de ellas tiene como objetivo operar en entornos ubicuos, si no en entornos móviles em-

Proyecto	Visión/Objetivos	Retos abordados
Aura (CMU) www.cs.cmu.edu/ aura	Computación centrada en el usuario: la tecnología debe ser "transparente" y debe proporcionarle servicios personalizados y adaptados al contexto/localización en el que se encuentra.	Infraestructura software. Construcción y composición de servicios. Adaptación a la localización. Captura y gestión de las intenciones de los usuarios.
DEAPspace (IBM)	Computación distribuida peer-to-peer en redes inalámbricas ad-hoc de un solo salto, formadas por dispositivos limitados. Su objetivo es minimizar las transmisiones y la cantidad de información a transmitir para realizar un consumo mínimo de las baterías.	Infraestructura software. Formato de descripción de servicios y codificación eficiente. Protocolo peer-to-peer para el descubrimiento y anuncio de servicios.
PIMA (IBM) www.research.ibm.com/PIMA	Modelo en el que los dispositivos son portales, las aplicaciones realizan tareas para el usuario, y el entorno de computación añade información y nueva funcionalidad al espacio físico que rodea al usuario.	Herramientas para el desarrollo de aplicaciones según el modelo definido abarcando el diseño, descarga y ejecución. Adaptación de las aplicaciones al contexto del usuario.
Oxygen (MIT) oxygen.lcs.mit.edu/	Computación centrada en el usuario. Desarrollo de espacios inteligentes que interaccionan con el usuario de forma "invisible" mediante interfaces visuales y lenguaje natural. Adaptación de las aplicaciones al dinamismo y características de las nuevas redes.	Nuevos dispositivos embebidos para crear espacios inteligentes. Infraestructura software para el desarrollo y composición de servicios. Nuevos protocolos de routing en redes ad-hoc. Protocolos de descubrimiento y anuncio de servicios. Soporte a la movilidad.
Portolano (University Washington) portolano.cs.washington.edu	Computación centrada en el usuario. Tres líneas de investigación: interfaces de usuario, computación distribuida basada en código móvil y solventar las deficiencias a nivel infraestructura de red.	Infraestructura software. Descubrimiento de servicios. Construcción y composición de servicios. Arquitectura para el desarrollo de aplicaciones basadas en localización.

Cuadro 2.7: Trabajos relacionados

pleando como infraestructura de comunicación redes de telefonía inalámbrica. Describimos brevemente estas iniciativas porque abordan el problema de implementar plataformas de agentes en la versión limitada de Java y por lo tanto, se enfrentan a problemas similares a los tratados en esta tesis doctoral en cuanto a la adaptación de plataformas de agentes a dispositivos limitados.

2.5.1. Aura de Carnegie Mellon University

El proyecto Aura comenzó en el año 2000 y tiene como objetivo crear una **arquitectura software de computación ubicua que se adapte al contexto y a las necesidades del usuario de forma invisible**, es decir, que su interacción con la tecnología embebida en el entorno sea mínima, de manera que no lo distraiga de sus tareas habituales. Aura se ha desarrollado para un entorno ubicuo en el que interaccionan dispositivos personales, dispositivos embebidos y dispositivos "vestibles" empleando protocolos de comunicación inalámbricos.

En Aura [Garlan et al., 2002] se trabaja con dos conceptos fundamentales: el primero se denomina *proactivity* (proactividad), que es la capacidad que tienen las capas del sistema para anticiparse a las peticiones realizadas a alto nivel,

el segundo se denomina *self-tuning* (autoajuste), que consiste en que las capas adaptan su consumo de recursos y su rendimiento a las peticiones y uso que se están haciendo de ellas. Teniendo como base estos conceptos han definido un arquitectura para dispositivos limitados con las siguientes capas:

- Kernel basado en Linux.
- Odyssey, que permite controlar recursos y da soporte a la adaptación de las aplicaciones al contexto.
- Coda, que proporciona acceso a ficheros adaptado a las diferentes condiciones de ancho de banda de la conexión inalámbrica, a la movilidad de los dispositivos y que permite operación desconectada.
- Spectra, que proporciona un mecanismo de ejecución remoto que emplea el contexto del usuario para decidir cuál es la mejor forma de ejecutar la petición remota realizada.
- Prism, que permite capturar y gestionar las intenciones de los usuarios y es una capa por encima de la de aplicación.

Tanto Odissey como Coda [Satyanarayanan, 1996] eran sistemas existentes desarrollados para entornos de computación móvil, que se han modificado para operar bajo las restricciones que impone los entornos de computación ubicua. La capa Prism es la capa en la que están centrando sus últimas investigaciones.

En Aura para aumentar las capacidades de los dispositivos limitados se emplea lo que se denomina *cyber foreign* que son típicamente PCs que se embeben en los entornos por los que se mueve el usuario y que los dispositivos limitados emplean para delegar ciertas tareas en ellos y a través de los que se conectan a Internet.

Dentro del proyecto Aura se han realizado también desarrollos orientados a lo que se denominan *advisor devices* (dispositivos consejeros) para proporcionar información de las condiciones de la red, y a lo que se denomina *people locator* (localizadores de personas) que permiten ofrecer servicios teniendo en cuenta la localización y el contexto del usuario. Ambos trabajos se han realizado para el protocolo inalámbrico IEEE 802.11.

2.5.2. IBM Research

IBM es una de las empresas que más está apoyando la investigación en computación ubicua, y así lo demuestran los diferentes proyectos que se están llevando a

cabo en sus principales centros de investigación. A continuación describimos dos de los más destacados.

Proyecto DEAPspace de IBM Research Zurich

El proyecto DEAPspace se centra en dar soporte a la computación distribuida peer-to-peer entre dispositivos limitados cuando se aproximan unos a otros y forman una red ad-hoc.

Uno de los principales resultados del proyecto ha sido la definición de un **protocolo de descubrimiento y anuncio de servicios** denominado DEAPspace Algorithm [Nidd, 2001], a través de este algoritmo un dispositivo puede detectar la presencia de dispositivos próximos, compartir información de los servicios que están disponibles en la red y detectar cuando un dispositivo deja de estar disponible. El protocolo ha sido diseñado para operar de forma eficiente en redes inalámbricas ad-hoc de un solo salto, y su objetivo principal es dar respuesta a los cambios frecuentes que se producen en este tipo de entornos, teniendo en cuenta las restricciones de potencia y limitaciones de los dispositivos. El algoritmo se basa en un método "*push*" puro, en el que todos los dispositivos mantienen una "*world view*" (visión del mundo) que transmiten cada cierto periodo de tiempo a sus vecinos mediantes mensajes de broadcast y que actualizan escuchando el "*world view*" que tienen los demás.

En el proyecto se ha definido un **nuevo modelo de servicios** basado en roles [Hermann et al., 2001], el rol que realiza la entidad que proporciona el servicio se denomina *provider* (proveedor) y el que realiza la entidad que accede al servicio se denomina *requester* (peticionario). Los mecanismos de comunicación entre estas entidades son independientes de la localización de las mismas, siguiendo las bases de los modelos tradicionales de computación distribuida. Independientemente del rol que realice una entidad, ésta proporciona al menos una de dos siguientes interfaces: interfaz de entrada e interfaz de salida, estas interfaces aceptan datos en un formato determinado y para que un *requester* pueda acceder a un *provider*, sus interfaz de entrada e interfaz de salida respectivamente, deben poseer algún formato de datos común.

Otra de las contribuciones ha sido la definición del **formato de descripción de los servicios** y de su **mecanismo de codificación** para minimizar la cantidad de datos a transmitir. En DEAPspace se realiza una distinción entre el protocolo de comunicación a través del que se accede a un servicio y el protocolo que se emplea para el descubrimiento, en general puede no ser el mismo y por ello se tiene en cuenta a la hora de definir el formato de descripción de un servicio.

En la actualidad este proyecto no está activo, aunque sus investigaciones se

iban a centrar en solventar los problemas de seguridad, de configuración y gestión existentes en estos entornos, así como introducir aplicaciones sensibles a la localización y multidispositivo.

Proyecto PIMA de IBM T. J. Watson Research Center

En el proyecto PIMA [Banavar et al., 2000] se pretende dar soporte tecnológico a un **nuevo modelo de aplicaciones** que han definido para entornos de computación ubicua. Este modelo está basado en tres principios:

- Un dispositivo es un portal en el espacio de datos y aplicaciones, no es un repositorio de software instalado por el usuario.
- Una aplicación realiza una tarea que precisa el usuario, no es una pieza de software que está escrita para explotar las capacidades de un determinado dispositivo.
- Un entorno de computación proporciona información sobre el espacio físico en el que se encuentra el usuario, no es un espacio que nos permite almacenar y ejecutar software.

El modelo de aplicación se divide en tres partes: diseño, descarga y ejecución. En la parte de **diseño** se necesita tener un modelo de programación que permita al programador abstraerse del dispositivo concreto en el que se va a ejecutar la aplicación, y estructurar el programa en términos de tareas y subtareas.

En la parte de **descarga** es necesario abordar importantes retos: descubrimiento dinámico de las aplicaciones y servicios que se proporcionan a nuestro alrededor, negociación para adaptar los requisitos de las aplicaciones a las capacidades del dispositivo en el que se va a ejecutar, selección del interfaz de usuario más adecuado al dispositivo, e integración y composición de la aplicación descargada con las aplicaciones residentes en el dispositivo.

Por último, en tiempo de **ejecución** es necesario que las aplicaciones se adapten al entorno cambiante y limitado en el que se ejecutan, por lo tanto, deben implementar mecanismos de detección y recuperación de errores, mecanismos de monitorización y redistribución de recursos para adaptarse a las limitaciones de los dispositivos en los que se ejecutan y ser capaces de adaptarse a la conectividad discontinua, empleando para ello mecanismos como la migración de código.

2.5.3. Proyecto Oxygen de MIT

Este proyecto está siendo desarrollado en el *Artificial Intelligence Laboratory* del MIT y tiene como principal objetivo hacer de la computación ubicua una realidad, desde la prespectiva de que ésta debe proporcionarle al usuario lo que necesita. Oxygen aborda este reto considerando tres elementos: los dispositivos personales del usario, *Handy21s* (H21s), que serán el interfaz entre el usario y el entorno ubicuo; los dispositivos embebidos, *Enviro21s* (E21), que permitirán introducir capacidad de comunicación y computación en el entorno que rodea al usuario; y por último las nuevas redes, *Networks21s* (N21s), que permitirán la comunicación espontánea entre usuarios y dispositivos, y entre dispositivos.

En H21 y E21 la investigación se ha realizado fundamentalmente a nivel hardware. En cuanto a las redes, N21, el objetivo de Oxygen es proporcionar redes descentralizadas en las que los dispositivos entren y salgan de forma dinámica sin necesidad de configuraciones automáticas. En N21 se pretenden integrar todo tipo de redes tanto inalámbricas, como terrestres, como satélites y para ello se están definiendo una serie de algoritmos, protocolos y *middleware* para:

- Configuración automática de regiones formadas por un conjunto de dispositivos conectados entre ellos, entre los que existe una relación de confianza.
- Descubrimiento y localización de servicios de forma automática.
- Implementación de mecanismos de seguridad para el acceso a los servicios.
- Adaptación a los cambios constantes en la red, incluyendo congestión, control de errores, balance de carga, latencia, restricciones de potencia y requisitos de las aplicaciones.

En la actualidad existen ya algunas propuestas realizadas para cumplir estos objetivos, la más significativa desde el punto de esta propuesta de tesis son las relacionados con el **descubrimiento y localización de servicios** de forma automática en redes dinámicas: **Intentional Naming System (INS)** [Adjie-Winoto et al., 1999]. Las principales diferencias de INS, respecto a otros protocolos de descubrimiento de servicios, es que trata el descubrimiento de servicios de manera similar a la resolución de nombres en Internet (DNS). INS integra esta resolución de nombres con el encaminamiento, además, los elementos que realizan la resolución incorporan balanceo de tráfico y se soporta replicación manteniendo la consistencia mediante el intercambio de mensajes.

2.5.4. Proyecto Portolano de University of Washington

La Universidad de Washigton es uno de los centros más activos en cuanto a la investigación en computación móvil y ubicua. Uno de los proyectos más importantes que se están desarrollando es el proyecto Portolano.

El proyecto Portolano [Esler et al., 1999] aborda los retos que plantea la computación móvil y ubicua centrándose fundamentalmente en tres elementos. El primero de ellos y más importante son las **interfaces de usuario**, los objetivos principales son: por una parte, permitir que múltiples interfaces diseñadas para distintos tipos de dispositivos interaccionen con la misma aplicación final, es decir, independizar la presentación de la semántica del interfaz; y por otra parte, conseguir interfaces "invisibles" desde el punto de vista del usuario.

El segundo de ellos se centra en los **servicios distribuidos**, en un entorno ubicuo no se deben proporcionar soluciones centralizadas dependientes de dispositivos concretos, y es necesario construir arquitecturas horizontales, de manera que las aplicaciones finales se apoyen en una serie de servicios comunes. Además, las aplicaciones deben estar orientadas a realizar las tareas que desea el usuario y en este sentido en Portolano se pretende aplicar el paradigma de código móvil. Otro reto que es necesario abordar a este nivel es la integración de servicios de manera fácil y flexible para que el usuario pueda acceder a nuevos servicios de forma dinámica y espontánea, sin necesidad de configuraciones o actualizaciones previas.

Por último, plantean la necesidad de realizar un estudio de los retos que aparecen a nivel **infraestructura de red** para alcanzar los objetivos anteriores, para ello van a analizar la viabilidad de utilizar soluciones ya existentes en el campo del descubrimiento dinámico de servicios de forma segura, de la compartición y acceso ubicuo a información almacenada en la red y de la tecnología para computación distribuida.

El proyecto Portolano a su vez se ha dividido en una serie de subproyectos que se han centrado en aportar soluciones en temas concretos de la visión global planteada:

- **Contact: Intrabody Signaling** [Partridge et al., 2000], este proyecto tiene como objetivo permitir la comunicación entre dispositivos ubicuos utilizando como medio de transmisión el cuerpo humano, de esta forma a través de un dispositivo personal del usuario se configurarán otros dispositivos con los que el usuario se ponga en contacto físico.
- **Hydra: Embedded Web Server**, en este proyecto, que ya no está activo, se desarrolló un dispositivo hardware que servía como gateway entre

dispositivos con interfaz RS-232C e Internet. El software del dispositivo está basado en Linux y sobre él se ha desarrollado un servidor web embebido, un daemon telnet y un servidor de ficheros propietario denominado "my-FTP".

- **The Location Stack** [Hightower et al., 2002, Hightower et al., 2001], en este proyecto se aborda el desarrollo de aplicaciones sensibles con la localización, de tal forma que se tengan en cuenta los movimientos y posición del usuario a la hora de proporcionarle servicios. En el proyecto se ha definido un stack de siete capas (similar al modelo OSI) como abstracción para modelar los diferentes aspectos a tener en cuenta al desarrollar una aplicación sensible con la localización.

- **one.world** [Grimm et al., 2002], este proyecto se centra en proporcionar una arquitectura para computación ubicua. Esta arquitectura se basa en una serie de abstracciones para separar los datos de la funcionalidad, siguiendo la visión global del proyecto Portolano. Así una aplicación almacena y intercambia datos utilizando tuplas y está compuesta por componentes que implementan la funcionalidad asociada. Las aplicaciones tienen al menos un entorno en el que almacenan tuplas y en el que sus componentes se instancian. La arquitectura además de dar soporte para desarrollar aplicaciones según este modelo, incluye una serie de servicios básicos: operaciones que ayudan a la gestión de operaciones asíncronas, migración que permite mover o copiar un entorno de un dispositivo a otro, punto de chequeo que permite almacenar el estado de ejecución de una tarea en una tupla para poder recuperarlo después de una migración, *Remote Event Passing* (REP) que permite enviar eventos a servicios remotos, que es el proceso de comunicación básico que se emplea en one.world y por último, descubrimiento que permite encaminar eventos a servicios de los que no se conoce su localización.

  El desarrollo llevado a cabo en este proyecto, es similar al diseño e implementación de una plataforma de agentes móviles, pero basándose en sus propias abstracciones.

2.5.5. Plataformas de agentes en dispositivos limitados

El hecho de que la mayoría de plataformas de agentes se hayan realizado en Java, ha llevado a que, cuando aparecen las primeras implementaciones de referencia de J2ME, algunos grupos de investigación intentaran migrar sus plataformas a esta versión y de esta forma se pudiesen ejecutar agentes en dispositivos

limitados.

En este apartado realizamos una breve descripción de las dos plataformas existentes en la actualidad que se ejecutan sobre la versión CLDC de J2ME, la primera de ellas, LEAP, es el resultado de un proyecto europeo, y la segunda, MAE, de los trabajos realizados en Monash University en Australia.

También describimos el trabajo del grupo FIPA Ad Hoc, formado a principios de este año con el objetivo de adaptar las especificaciones FIPA, para que se puedan implementar en dispositivos limitados que se comunican mediante protocolos inalámbricos y forman de forma espontánea una red ad-hoc.

2.5.6. LEAP

El proyecto LEAP [LEAP Project, 2000] se engloba dentro de los proyectos europeos IST, y está siendo desarrollado por un consorcio de compañías entre las que se encuentran, entre otras, Motorola, British Telecom y Siemens. El objetivo del proyecto es el desarrollo de una plataforma de agentes móviles conforme al estándar FIPA que pueda operar tanto en dispositivos móviles (teléfonos móviles, PDA, pagers) como en PCs. El proyecto comenzó en Enero de 2000 y tiene dos fases, la primera de ellas, ya finalizada, consistió en la revisión de los estándares FIPA y WAP y el diseño e implementación de una plataforma de agentes para dispositivos móviles. La segunda fase consiste en evaluar la plataforma en sistemas reales y analizar las prestaciones obtenidas en dos aplicaciones: asistencia en carretera y tareas de gestión de red. Para que la plataforma de agentes desarrollada en este proyecto opere tanto en sistemas PCs como en dispositivos móviles con capacidades limitadas, han diseñado una arquitectura modular estructurada en dos partes:

- Parte obligatoria, compuesta por varios módulos, uno denominado *kernel* independiente del dispositivo y otros dedicados a las comunicaciones y dependientes del dispositivo en el que se ejecute la plataforma.
- Parte opcional, compuesta por varios módulos para interfaces gráficas, *parsers*, modelado de usuario y datos.

La plataforma de agentes se configurará así mediante un instalador que compondrá los módulos necesarios para el dispositivo concreto en el que se instale.

Para dispositivos móviles emplean las extensión correspondiente de Java, J2ME/CLDC, y para entornos PCs J2SE. El kernel de la plataforma solamente emplea las APIs comunes a ambas especificaciones de Java.

El proyecto LEAP fue pionero a la hora de demostrar la viabilidad de construir plataformas de agentes en dispositivos limitados, aunque su aplicabilidad está centrada en redes con infraestructura y por lo tanto, en los terminales móviles no existía una plataforma completa de agentes como tal y su operación dependía siempre de un sistema intermedio al que estuviese conectado.

Aunque era uno de los objetivos marcados en el proyecto, la movilidad a nivel agente no está soportada, esto se debe fundamentalmente a las propias limitaciones de J2ME, que no permite la definición de cargadores de clase definidos por el usuario.

El desarrollo de LEAP se basa en la plataforma sobre J2SE conforme con FIPA desarrollada en la Universidad de Parma, denominada JADE.

Posteriormente a esta iniciativa han aparecido algunas otras plataformas de agentes conformes con FIPA como son [Micro FIPA-OS, 2001] y Grasshopper MicroEdition [Grasshopper MicroEdition, 2001], para dispositivos limitados que soportan Personal Java. En la actualidad, Personal Java es una versión obsoleta cuyo equivalente en la arquitectura J2ME será el Personal Profile sobre la configuración CDC.

MAE de Monash University

En Monash University [Mihailescu y Kendall, 2002] se ha diseñado e implementado una plataforma de agentes para dispositivos personales móviles basados en PalmOS.

La plataforma desarrollada se denomina MAE y en la actualidad tienen dos implementaciones utilizando dos versiones reducidas de Java: la configuración CLDC de J2ME, y SuperWaba, que es una versión limitada de Java con algunas funcionalidades no soportadas por la versión oficial de Sun, como JNI.

MAE está compuesta por un *device agent execution environment* (DAEE) que proporciona un conjunto de servicios a los agentes que se ejecutan en la plataforma. Estos servicios se agrupan en tres tipos: presentación, que proporciona los servicios que permiten a los agentes interaccionar con el usuario; agente, que proporciona los servicios que permiten a los agentes interactuar con los servicios de red, entre ellos se encuentra el que permite descubrir y anunciar servicios y el que permite conocer el estado de la red; red, que permite a los agentes comunicarse con otros dispositivos.

Esta plataforma soporta movilidad a nivel agente, para solventar el problema de la carga dinámica de clases se han empleando mecanismos específicos de los sistemas PalmOS, por lo que esta característica no es migrable a otro tipo de

dispositivos J2ME.

Comité Técnico FIPA Ad-Hoc

El comité técnico *FIPA Ad Hoc* surge a principios de 2002 con el objetivo de definir una plataforma de agentes conforme con FIPA, que pueda operar en redes ad-hoc. El principal problema al que se enfrentan en este tipo de redes, es que son redes sin infraestructura y entonces, la plataforma existente en cada dispositivo debe ser autónoma, y por lo tanto poseer todos los componentes definidos en el modelo de referencia de FIPA.

En la actualidad el tema más activo del grupo se centra en determinar cómo debe realizarse el descubrimiento de servicios/agentes y por lo tanto, analizan si la definición de *Directory Facilitator* y los mecanismos de federación para búsquedas remotas en FIPA es válido y posible en redes ad-hoc. El grupo fundamenta sus trabajos en tres requisitos:

- Añadir los mínimos cambios a las especificaciones existentes de FIPA.
- Utilizar los mecanismos de descubrimiento dinámico de servicios existentes en la actualidad (Jini, JXTA, SLP, SSDP, etc), de manera que la solución propuesta pueda interoperar con cualquiera de ellos.
- Adaptarse a entornos ad-hoc, pero que la solución propuesta también sea válida para redes con infraestructura.

El grupo está escribiendo un White Paper [WG-AdHoc, 2002] en el que se analizan detalladamente los requisitos anteriores y los retos que éstos plantean, realizando además un estudio de las diversas tecnologías de descubrimiento dinámico de servicios existentes en la actualidad. La autora de esta propuesta de tesis doctoral ha aportado contribuciones al grupo de trabajo que se incluyen en esta memoria en los apéndices C y D y que han sido incluidas en el White Paper.

2.5.7. Resumen

Los proyectos que hemos descrito a lo largo de este apartado tienen una visión común, que coincide con la que nosotros nos hemos planteado: la tecnología desarrollada en entornos de computación ubicua debe adaptarse al usuario, de manera que para él, ésta sea "invisible". Lo que diferencia unos proyectos de otros es la temática concreta que abordan para contribuir a que la tecnología haga que ésto sea una realidad. Algunos de ellos como Oxygen centran gran parte de

su investigación a nivel hardware, construyendo dispositivos que se pueden embeber en el entorno físico del usuario, otros como PIMA se centran en proporcionar entornos de desarrollo que faciliten el diseño e implementación de aplicaciones según el nuevo modelo que impone la computación ubicua.

La característica común de todos ellos es que abordan la necesidad de proporcionar una serie de tecnologías middleware para facilitar el desarrollo de servicios y aplicaciones en entornos de computación ubicua, en concreto:

- Soporte para la ejecución de aplicaciones en dispositivos limitados: arquitectura Aura y one.world de Portolano.
- Soporte para el descubrimiento y anuncio dinámico de servicios: DEAPspace algorithm, INS de Oxygen y el protocolo de descubrimiento dentro de one.world de Portolano.
- Soporte para desarrollar servicios teniendo en cuenta el contexto y localización del usuario: Aura, Oxygen y Portolano.

En esta tesis doctoral planteamos realizar contribuciones a las dos primeras. En concreto, en el caso del soporte para la ejecución de aplicaciones, las propuestas realizadas difieren de la que aquí se plantea en que, en la propuesta de Aura los dispositivos limitados precisan un dispositivo tipo PC para integrarse en el sistema, y en la propuesta de one.world, aunque se basa como nosotros proponemos en conceptos de agentes móviles, define de nuevo una plataforma basada en sus propias abstracciones sin tener en cuenta las investigaciones y estándares que se han realizado hasta la actualidad.

En el caso del descubrimiento dinámico, las contribuciones difieren respecto a la que nosotros nos proponemos en cuanto que el protocolo de one.world utiliza en su algoritmo elementos centrales; el protocolo INS va más allá del descubrimiento de servicios y en él se introducen mecanismos de routing, que desde nuestro punto de vista deben abordarse de manera independiente; por último, el protocolo DEAPspace se aproxima más a los requisitos que nosotros nos planteamos, aunque realiza ciertas consideraciones sobre la propagación de anuncios en los que no se tienen en cuenta las distancias.

En cuanto a los trabajos relacionados para llevar la tecnología de agentes, aunque las propuestas analizadas han realizado importantes contribuciones en cuanto a adaptarlas a las limitaciones de los dispositivos, ninguna de ellas se ha realizado teniendo en mente las restricciones que impone el entorno. En el caso de LEAP, la plataforma se realizó para operar en redes inalámbricas con infraestructura, por lo que la plataforma que se ejecuta en el dispositivo limitado

no es autónoma y depende de un sistema no limitado situado en la red. En cuanto a MAE, su implementación se ha realizado para utilizarla en comercio móvil, y aunque es bastante completa ya que incluye movilidad, se ha realizado utilizando funcionalidades concretas de dispositivos PalmOS y no se ha tenido en cuenta el estándar FIPA para realizar su diseño.

2.6. Conclusiones

A lo largo de este capítulo hemos realizado un estudio de las diferentes tecnologías que es necesario tener en cuenta, a la hora de realizar contribuciones en el campo de la definición de tecnologías middleware para el desarrollo de servicios en entornos de computación ubicua. En las secciones 2.1 y 2.2, hemos visto como la tecnología estos últimos años ha dado respuesta a dos de los retos fundamentales, que van a permitir que la visión precursora de Weiser se haga realidad: dotar a un mayor número de dispositivos de capacidad de comunicación y computación.

En cuanto a la capacidad de comunicación, en la sección 2.1, hemos realizado un análisis que ha ido desde los protocolos inalámbricos de área extensa, hasta los de área local y personal. También hemos estudiado los protocolos desarrollados para el entorno del hogar, por ser este entorno en el que primero se han aplicado los conceptos de computación ubicua. Del estudio realizado concluimos que cada vez más elementos del mundo físico pueden tener capacidad de comunicación, desde electrodomésticos del hogar hasta pequeños dispositivos como sensores. Además, parece claro que la tendencia actual es que un mayor número de dispositivos empleen protocolos inalámbricos, fundamentalmente porque es una solución poco invasiva, al no precisar cableado, y cada vez menos costosa debido a su popularización. El uso de estos protocolos permite de manera flexible la creación de entornos de computación ubicua, pero impone ciertas restricciones: calidad cambiante, conectividad no transitiva, coste de las comunicaciones,..., que es necesario tener en cuenta a la hora de proporcionar soluciones eficientes a alto nivel. También se debe tener en cuenta que cada vez más dispositivos poseen varios interfaces de red y que dependiendo de la aplicación y del coste emplearán uno determinado.

El desarrollo de aplicaciones para los nuevos dispositivos con capacidad de cómputo y comunicación, ha estado cerrado a los propios fabricantes, principalmente por la gran heterogeneidad de sistemas operativos existentes, incluso para el mismo tipo de dispositivo. La aparición de J2ME, como hemos visto en la sección 2.2, facilita el desarrollo de aplicaciones a terceros, gracias a que es un lenguaje multiplataforma, y de esta forma se enriquece el número de servicios

que se pueden ofrecer al usuario. Actualmente, la arquitectura J2ME da soporte a dispositivos personales y de función específica, y aunque no existe todavía ninguna iniciativa que permita introducir Java en dispositivos con mayores limitaciones, parece viable su realización, sobre todo si tenemos en cuenta la existencia de JavaCard. En esta tesis doctoral se empleará la tecnología J2ME para analizar la viabilidad de implementación en dispositivos actuales de las tecnologías middleware definidas.

Una de la tecnologías middleware fundamentales para que la visión de Weiser se haga realidad, además de las anteriores, es el descubrimiento de servicios, que permite que los diferentes dispositivos que ofrecen diversos servicios sepan qué servicios ofrecen los demás y de esta forma interaccionar para ofrecer nuevas aplicaciones al usuario. Además, el proceso de descubrimiento debe realizarse de forma automática y sin necesidad de configuraciones, para conseguir la transparencia de la tecnología, uno de los retos básicos que existen en computación ubicua. En la sección 2.3, hemos descrito las soluciones que se han proporcionado en los últimos años al problema del descubrimiento de servicios en redes de área global y local. Las alternativas existentes son muy diversas, desde las que se han definido para un protocolo de red concreto, hasta las que están integradas dentro de un framework de desarrollo de servicios, pero hasta la actualidad ninguna de ellas se ha implantado como estándar de facto. Las más genéricas son las que se han definido dentro del IETF, ya que su único requisito es que el dispositivo tenga una pila IP, es por ello que en el capítulo 3, vamos a realizar un análisis más detallado de estos protocolos y abordaremos el problema del descubrimiento de forma analítica, con el objetivo de ver si estos protocolos son aplicables a los entornos de computación ubicua. Veremos que, desde nuestro punto de vista, no se cubren las necesidades impuestas y por ello, en el capítulo 4 proponemos un nuevo protocolo de descubrimiento de servicios, denominado Pervasive Discovery Protocol (PDP), cuyas prestaciones se analizan en el capítulo 5.

Como hemos indicado anteriormente uno de los retos de la computación ubicua es la transparencia de la tecnología, es decir, que el entorno físico que rodea al usuario se adapte a sus necesidades sin necesidad de configuraciones previas. Los usuarios interaccionarán con el entorno a través de sus dispositivos personales, éstos no podrán tener preinstalado todo el software que pueden necesitar para acceder a los diversos servicios que se ofrecen en todos los lugares en los que está y en los que existirán una gran heterogeneidad de dispositivos. Así, será necesario habilitar algún mecanismo para que de forma dinámica este software se descargue de forma automática y permanezca sólo en el dispositivo mientras se realice una determinada tarea. Del mismo modo, el usuario podrá necesitar que determinado software se ejecute en otros dispositivos que le ofrecen ciertos servicios que su dispositivo local no tiene. El paradigma de la computación que mejor se adapta a

estos requisitos es el de agentes móviles, que además ofrece ventajas en entornos donde las comunicaciones tienen calidad cambiante y conectividad intermitente. En la sección 2.4, hemos estudiado el concepto de agentes y hemos resumido las principales características de los agentes móviles. En esta tesis proponemos el uso del paradigma de agentes móviles como tecnología middleware para el desarrollo de servicios en entornos ubicuos, para ello es necesario realizar un diseño de plataforma de agentes que se adapte a las limitaciones que presentan estos entornos. En este diseño tomaremos como punto de partida el estándar FIPA e intentaremos, con el mínimo impacto en la especificación, adaptarlo a los nuevos requisitos. En el capítulo 6 vamos a detallar las contribuciones realizadas en este campo y que han sido propuestas en el comité técnico FIPA Ad-Hoc.

Por último, en la sección 2.5 hemos descrito brevemente algunos de los principales proyectos de investigación que tienen objetivos comunes con esta tesis doctoral. Algunos de los proyectos analizados abordan el problema de descubrimiento empleando soluciones existentes o definiendo un protocolo propio, como es el caso del proyecto DEAPspace y Oxygen. En cuanto al uso del paradigma de agentes, tanto en el proyecto Portolano, como en el proyecto one.world se propone el uso de código móvil como un mecanismo adecuado para vencer las limitaciones y heterogeneidad de los dispositivos que existen en un entorno, aunque en ninguno de los casos se propone soluciones basadas en estándares de agentes móviles. Respecto a este último aspecto, en esta sección también se proporciona una descripción de algunos proyectos en los que se han realizado implementaciones de plataformas de agentes en dispositivos limitados para su uso en redes móviles con infraestructura.

Capítulo 3

Estudio de mecanismos de descubrimiento de servicios

El descubrimiento de servicios de manera dinámica no es algo nuevo, existen varias propuestas al respecto, más o menos extendidas en entornos de redes fijas [Richard III, 2000] [Helal, 2002]. El objetivo principal con el que surgieron estos protocolos fue facilitar a un dispositivo móvil el descubrimiento, configuración y uso de los servicios que se ofrecían en la nueva red a la que se conectaba. En general, las diferentes propuestas abordan el problema de distintas formas, desde las que han sido diseñadas para su uso con un determinado protocolo de red, hasta las que están asociadas a un determinado lenguaje de programación.

En este capítulo realizamos un estudio detallado del problema del descubrimiento de servicios y analizamos las soluciones más relevantes que se han proporcionado estos últimos años. Así, en la sección 3.1, repasamos los antecedentes de los protocolos de descubrimiento de servicios. En la sección 3.2, abordamos el estudio del concepto de servicio, viendo las diferentes definiciones que se han dado y relacionándolas con el problema del descubrimiento. En la sección 3.3, realizamos un análisis teórico del problema del descubrimiento de servicios en una red de dispositivos. En la sección 3.4, nos centramos en el estudio de los protocolos de descubrimiento de servicios propuestos en el IETF. Finalizamos, sección 3.5, con las conclusiones del estudio realizado y que nos han llevado a la definición de un nuevo protocolo de descubrimiento de servicios para entornos de computación ubicua.

3.1. Antecedentes

Los protocolos de descubrimiento de servicios están muy relacionados con el intento de minimizar las configuraciones que debe realizar un usuario en un dispositivo para poder conectarse a una red y acceder a los servicios que en ella se ofrecen.

El primer servicio que se quiso proporcionar a los usuarios sin necesidad de intervención manual fue el servicio de nombres. Cuando el número de máquinas conectadas a Internet comenzó a aumentar, se vió la necesidad de asociarles un nombre que fuera más fácil de recordar a las personas que su dirección IP. La solución inicial consistía en tener unos ficheros en los que se listaban las correspondencias entre direcciones IP y nombres de máquinas. Existía un servidor central en el que se actualizaba esta lista y cada noche todas las máquinas se conectaban a él utilizando el protocolo FTP para actualizar su lista local.

Como el número de máquinas conectadas a Internet seguía creciendo, fue necesario proporcionar un nuevo mecanismo para tener un servicio de nombres escalable. El IETF define así en 1982 el Sistema de Nombres de Dominio (DNS), RFC 1034 [Mockapetris, 1987a] y RFC 1035 [Mockapetris, 1987b], que gestiona los nombres de forma jerárquica. Aunque DNS se definió para asociar nombres de máquina a direcciones IP, la realidad es que su funcionalidad se amplió para que proporcionase un servicio de consulta genérico, en el que un usuario solicita cuál es la información asociada a un determinado nombre de dominio.

Cuando los avances de la electrónica hicieron posible que los ordenadores pudieran ser portátiles, el principal incoveniente que tenían los usuarios es que cuando cambiaban su ubicación física, su ordenador debía conectarse a una nueva red, y era preciso configurar de nuevo la interfaz de red para lo que necesitaban datos proporcionados por un administrador. Se define así en 1993 Dynamic Host Configuration Protocol (DHCP), RFC 2131 [Droms, 1997], que permitía de forma dinámica obtener los datos necesarios para configurar de forma automática la conexión de red. Entre los datos que se obtienen por DHCP está la dirección del servidor de DNS al se pueden realizar consultas.

En ese mismo año se forma el grupo SRVLOC[1] del IETF, el objetivo de las investigaciones del grupo consistía en permitir que el usuario además de obtener conectividad de red de manera automática, pudiera, sin necesidad de configuraciones adicionales, obtener también otros servicios que se ofrecen en la red, como pueden ser por ejemplo, un servicio de impresión. El principal resultado del grupo fue la definición del protocolo Service Location Protocol (SLP) que explicaremos

[1] http://www.srvloc.org/

en detalle en la sección 3.4.1 de este capítulo. El grupo definió paralelamente un nuevo esquema de URL para la definición de servicios, que es el que emplea SLP. El grupo concluyó su trabajo proporcionando varios RFCs con extensiones a SLP, en una de ellas se define cómo obtener a través de DHCP el servidor SLP de la red a la que se conecta.

Con la definición de estos protocolos se minimizan las configuraciones que un usuario debe realizar en un dispositivo para conectarse a una red fija y acceder a los servicios que en ella se ofrecen, independientemente que el acceso se realice de forma cableada o a través de protocolos inalámbricos.

Con la proliferación estos últimos años de dispositivos limitados con capacidad de cómputo que emplean protocolos inalámbricos de corto alcance para comunicarse con otros dispositivos, se nos plantean nuevos retos que es necesario abordar. Estos dispositivos al acercarse unos a otros pueden formar redes de manera espontánea en entornos como el hogar, aeropuertos, salas de reuniones o parques. Estas redes denominadas ad-hoc o sin infraestructura, no van a contar con servidores específicos de DHCP o DNS que faciliten la operación de la red, ya que todos ellos precisan un dispositivo que aloje estos servicios y un usuario que los administre. Por lo tanto, se deben definir nuevos protocolos que eliminen la necesidad de administración y configuración por parte de los usuarios.

En esta línea se centran los trabajos del grupo del IETF Zero Configuration Networking (ZeroConf[2]) creado en 1999, que está proponiendo protocolos con configuración nula en cuatro áreas: configuración de la interfaz IP, resolución de nombres, asignación de direcciones IP multicast y descubrimiento de servicios. Describiremos en la sección 3.4.3 los relacionados con el descubrimiento de servicios.

3.2. Definición y descripción de servicios

Los protocolos de descubrimiento de servicios permiten localizar al servidor que ofrece un determinado servicio, pero ¿qué es un servicio?. Varias han sido las respuestas que se han dado en la literatura en los últimos años, citamos a continuación las más relevantes:

"...service can be defined as any hardware or software entity that resides on any mobile device or platform and has distinct functional attributes that characterizes it"[Chakraborty, 2001]

[2]http://www.zeroconf.org/

"... a particular logical function that may be invoked via some network protocol, such as printing, storing a file on a remote disk, or even perhaps requesting delivery of a pizza"[Willians, 2002]

"... we defined such "services" as applications with well-known interfaces that perform computation or actions on behalf of client users"[Czerwinski et al., 1999]

"... a service is an entity that can be used by a person, a program, or another service. A service may be a computation, a storage, a communication channel to another user, a software filter, a hardware device, or another user [...] A service has an interface which defines the operations than can be requested of that service"[Jini, 1999]

En general, podemos decir que existen dos alternativas a la hora de abordar la definición del concepto de servicio: los que lo definen desde el punto de vista de una plataforma de computación distribuida y los que lo definen desde el punto de vista de protocolos de comunicación. En el primer caso se habla de interfaces, en el segundo de protocolos. En ambos casos, un servicio es una entidad que realiza una determinada función de la que un usuario o aplicación puede beneficiarse, la diferencia está en la forma en la que se accede a él y en la forma en la que se describe.

Consideremos por ejemplo una impresora, el usuario puede necesitar en algún momento solicitar la impresión de un documento que está editando en su agenda personal. Si hablamos del servicio impresión a nivel de protocolo de aplicación, esto se traducirá en localizar un elemento en la red que soporte el protocolo `lpr` o `ipp`; si lo hacemos desde el punto de vista de interfaces necesitamos un elemento que soporte una determinada interfaz, por ejemplo `Printer`. Una vez descubierto el servicio, en el primer caso la PDA necesitará conocer el protocolo `lpr` o `ipp`, la dirección IP y el número de puerto que le permita comunicarse con la impresora y enviarle el documento; en el segundo caso, el acceso al servicio consistirá en invocar métodos de la interfaz `Printer` sobre un determinado objeto, cuya localización y acceso remoto aisla la plataforma de computación distribuida con la que se esté trabajando.

Los protocolos de descubrimiento definidos dentro del IETF describen los servicios como protocolos, otras propuestas como Jini, JXTA o Salutation describen servicios a través de interfaces.

La descripción de un servicio debe proporcionar información necesaria para

poder acceder a él y características particulares que nos permitan diferenciar ese servicio de otros similares. En la mayoría de los formatos de descripción de servicios, esta información se proporciona a través de lo que se denominan atributos del servicio. Los tipos de atributos que tiene un servicio dependen del servicio que describen, para permitir la interoperabilidad se deben definir plantillas de descripción de servicios, en las que se indican los diferentes atributos y los valores que pueden tomar.

Un atributo común en la descripción de servicios, es el tipo de servicio. Un tipo engloba a un conjunto de servicios con características comunes. En la mayoría de formatos de descripción de servicios, se emplean jerarquías de tipos de servicios que van desde servicios más abstractos a más concretos. El tipo de servicio es el atributo que se emplea para realizar búsquedas, el servidor comprueba que ofrece un servicio del tipo solicitado antes de responder. Esta comprobación se realiza de forma sintáctica, aunque existen algunas propuestas en la literatura que permiten introducir semántica en las búsquedas [Ankolenkar et al., 2002].

3.3. Estudio teórico de mecanismos de descubrimiento

En esta sección realizamos un estudio sobre las diversas alternativas teóricas que existen para el descubrimiento de servicios de forma dinámica. Estas alternativas no se consideran protocolos de descubrimiento de servicios como tal, aunque son la base sobre las que se han definido los protocolos que han aparecido estos últimos años.

3.3.1. Planteamiento del problema

En todas las redes de comunicación existen un conjunto de dispositivos, denominados **servidores**, que ofrecen determinados servicios que pueden ser utilizados por otros dispositivos conectados a la red, denominados **clientes**.

Como hemos visto en el apartado anterior, la manera tradicional en la que un cliente descubre y accede a los servicios que ofrecen los servidores, es mediante configuraciones realizadas por los administradores de la red a la que está conectado. Cuando el número de servicios es muy elevado y además las redes son muy dinámicas: los dispositivos son móviles y por lo tanto, cambian de red frecuentemente; es necesario que estas configuraciones se realicen de forma automática, sin necesidad de intervenciones manuales continuas.

Para abordar este objetivo es necesario solucionar tecnológicamente dos problemas:

1. Descubrimiento de servicios, es decir, permitir que los clientes sepan qué servicios están disponibles.

2. Acceso a los servicios, es decir, permitir que los clientes, una vez descubiertos los servicios que necesitan en la red, hagan uso de ellos.

En los siguientes apartados estudiamos las diferentes alternativas que se plantean para solucionar el problema del descubrimiento de servicios de forma dinámica y analizamos las ventajas e incovenientes de cada una de ellas, dependiendo del escenario de aplicación.

3.3.2. Posibles soluciones

Solución I: funcionamiento distribuido tipo *pull*

La primera solución que se plantea es que los clientes cuando necesiten acceder a un servicio de un tipo determinado, envíen peticiones solicitándolo, a las que responden los servidores que ofrecen dicho servicio. Las peticiones de servicio se envían por multicast o broadcast mientras que las respuestas de los servidores se envían por unicast al cliente que lo ha solicitado, Figura 3.1. A este modo de funcionamiento se le denomina distribuido de tipo *pull*.

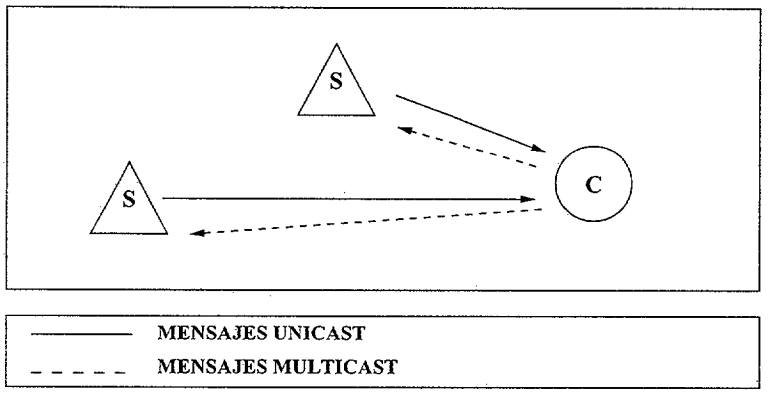


Figura 3.1: Funcionamiento distribuido tipo *pull*.

Esta forma de realizar el descubrimiento de servicios es fácil de implementar tanto en los clientes como en los servidores, pero tiene problemas de escalabilidad

en redes con un gran número de dispositivos debido al consumo excesivo de ancho de banda.

Una manera de reducir este problema es introducir cachés de servicios remotos en los clientes, de tal forma que un cliente almacena de forma local los servicios que va descubriendo. Cuando este cliente quiera acceder de nuevo a un tipo determinado de servicio, antes de enviar un nuevo mensaje de petición a la red, comprueba si lo tiene almacenado en su caché y si es así, no envía la petición.

De esta forma se minimiza el número de mensajes transmitidos pero se introduce un nuevo problema: ¿el servicio almacenado en la caché sigue estando disponible en la red?. Para reducir esta incertidumbre, los servidores anuncian sus servicios con un tiempo de vida asociado, que indica cuál es el tiempo máximo que un cliente puede tener almacenado ese servicio en la caché.

Aunque el mecanismo anterior reduce la probabilidad de que un servicio almacenado en la caché no esté disponible, no la elimina. Existen algunas técnicas que se pueden emplear para reducir más este problema:

- Sondeo: el cliente de forma periódica comprueba si los servicios almacenados en su caché están disponibles o han sufrido alguna modificación, poniéndose en contacto con el servidor que los ofrece.

- Suscripción y notificación: los clientes pueden suscribirse en los servidores, para que se les notifique los cambios o caídas de los servicios almacenados en su caché, cuando éstos se producen.

Si los cambios o caídas de los servidores son muy frecuentes la introducción de estas técnicas tiene un consumo excesivo de ancho de banda por lo que en entornos muy dinámicos, no debe hacerse uso de ellas.

Solución II: funcionamiento distribuido tipo *push*

La segunda solución que se plantea es que sean los propios servidores los que de forma activa anuncien periódicamente los servicios que ofrecen. Los clientes almacenan estos anuncios de forma local y cuando quieren acceder a un servicio de un tipo determinado, comprueban si es uno de los servicios anunciados previamente en la red o esperan a que se produzca un anuncio de ese tipo. Estos anuncios se envían por difusión Figura 3.2. A este modo de funcionamiento se le denomina distribuido de tipo *push*.

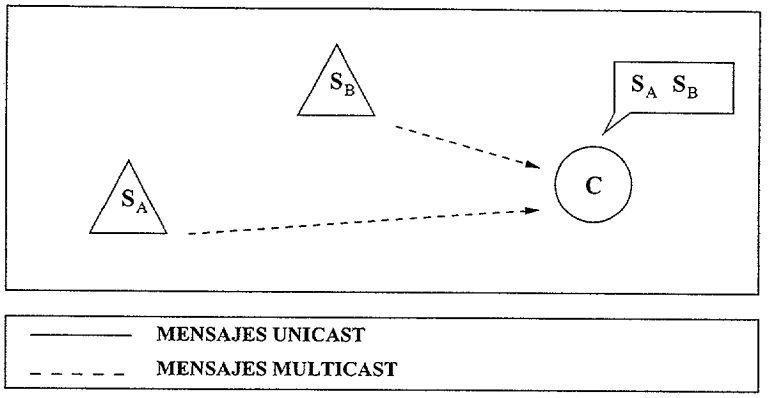


Figura 3.2: Funcionamiento distribuido tipo *push*.

Esta forma de realizar el descubrimiento de servicios introduce mayor complejidad en los clientes y en los servidores, ya que los clientes deben almacenar todos los anuncios que escuchan en la red y los servidores periódicamente deben anunciar los servicios que tienen disponibles, aún cuando ningún cliente precise ese servicio.

Como los clientes almacenan de forma local los servicios que se anuncian en la red, se nos plantea el mismo problema que al emplear cachés en el modo *pull*: ¿el servicio estará disponible cuando el cliente quiera acceder a él?. En el modo *push* los servidores envían los anuncios según una periodicidad que comunican al cliente, de tal forma, que un cliente cuando no recibe un anuncio en el intervalo previsto, elimina este servicio de su caché. En este caso el tiempo de vida de un servicio en la caché de los clientes, es el periodo de tiempo entre anuncios consecutivos del servidor que lo ofrece.

Entre anuncios consecutivos de un servidor se pueden producir caídas del mismo o modificaciones, que el cliente no detectará hasta el próximo anuncio. Por lo tanto, parece recomendable que los servidores anuncien con una frecuencia elevada sus servicios, pero de esta forma se consume un gran ancho de banda con el consecuente problema de escalabilidad del protocolo. Si se quiere reducir este consumo, minimizando la probabilidad de que un cliente no detecte cambios en el servicio, se pueden introducir técnicas de sondeo y de suscripción y notificación como se ha explicado en la solución tipo *pull*.

Solución III: funcionamiento centralizado

La tercera solución que se plantea es que existan unos elementos en la red que sirvan de intermediarios entre clientes y servidores para descubrirse, a estas

entidades se les denomina **directorios**. Así, los clientes descubren los servicios que ofrecen los servidores a través de los directorios, es por esto que a este modo de funcionamiento se le denomina centralizado: sin un directorio los clientes no pueden descubrir servicios.

Los pasos que llevan a cabo clientes, servidores y directorios para poder descubrir los servicios ofrecidos son los siguientes, Figura 3.3.a:

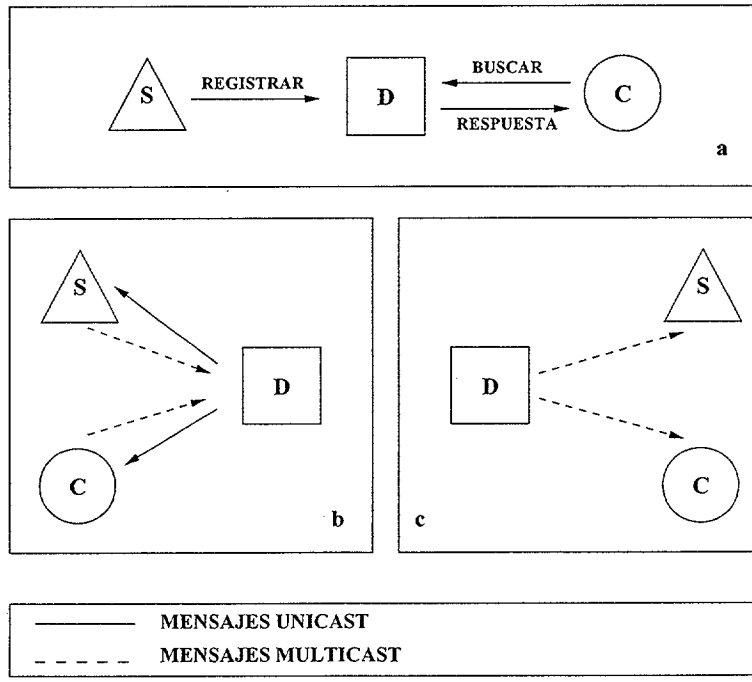


Figura 3.3: Arquitectura centralizada.

- Los servidores registran las descripciones de los servicios que ofrecen en uno o varios directorios.
- Los directorios almacenan estas descripciones de forma persistente.
- Los clientes solicitan a uno o varios directorios la búsqueda de un determinado servicio.
- Los directorios buscan en las descripciones que tienen almacenadas, si alguna se corresponde con el servicio solicitado, si la búsqueda es satisfactoria, devuelven esta descripción al cliente.

En los pasos descritos anteriormente se considera que tanto clientes como servidores conocen al menos a un directorio. Así que la pregunta que se nos plantea es: ¿cómo los clientes y servidores descubren directorios?. Se contemplan varias alternativas posibles:

- Configuración: tanto clientes como servidores tienen configurados uno o varios directorios.
- Dinámicamente: tanto clientes como servidores descubren dinámicamente a los directorios existentes.

La configuración es el mecanismo menos costoso pero menos flexible, debido a que si los directorios que tienen configurados los clientes y los servidores dejan de estar disponibles, los clientes no podrían descubrir a los servidores hasta que se configure manualmente un nuevo servicio de directorio. Además esta solución se contradice con la idea de que los protocolos de descubrimiento de servicios surgen para minimizar el número de configuraciones manuales que es necesario realizar.

El mecanismo dinámico no deja de ser una solución al problema de cómo descubrir un servicio de directorio de forma distribuida, por lo tanto se plantean dos alternativas:

- Modo *pull*: los clientes y servidores envían por multicast o broadcast peticiones de búsquedas de servicio de directorios, a las que responden los directorios por unicast, Figura 3.3.b.
- Modo *push*: los directorios se anuncian periódicamente y tanto los clientes como los servidores escuchan estos anuncios para descubrir a los directorios. Estos anuncios se envían por multicast o broadcast, Figura 3.3.c.

También se pueden emplear técnicas que combinan ambos modos:

- Cuando los clientes y servidores arrancan o entran en una nueva red, llevan a cabo lo que se denomina *búsqueda agresiva* de servicio de directorio, en la que se emplea un mecanismo *pull* en el que de forma periódica se envían peticiones de búsqueda de servicio de directorio. Esta búsqueda agresiva se realiza durante un periodo de tiempo limitado, después del cual no se realizan más peticiones de este tipo.
- Los directorios además se anuncian periódicamente, siguiendo un mecanismo *push*. El periodo entre anuncios se duplica en cada nuevo anuncio, para disminuir el tráfico generado por este tipo de mensajes.

El objetivo de emplear los dos mecanismos es por una parte, localizar lo antes posible a todos los servicios de directorio existentes en la red para que clientes y servidores comiencen a descubrirse, y por otra, que los clientes y servidores sepan qué directorios siguen activos o si existe algún directorio nuevo en la red.

En general, los directorios se introducen para aumentar la escalabilidad de los mecanismos de descubrimiento en redes amplias. Además, los directorios suelen estar asociados a divisiones administrativas, de tal manera que los clientes y servidores de una organización emplean un determinado directorio. El punto débil de este modo de funcionamiento es la dependencia de un único punto de fallo, ya que la caída de un directorio supone que clientes y servidores no pueden descubrirse.

Para disminuir el número de consultas que realizan los clientes a los directorios para descubrir servicios, los clientes pueden emplear cachés de servicios remotos de forma local. Es el directorio en este caso, el que les indica cuál es el tiempo máximo que pueden almacenar ese servicio en su caché. Como en los casos anteriores, al existir cachés locales en los clientes, se pueden introducir las técnicas siguientes:

- Sondeo: los servidores se ponen en contacto con los directorios en los que tienen registrados sus servicios, para indicar los cambios producidos en ellos. Los clientes se ponen en contacto con los directorios para contrastar si la información que tiene almacenada de forma local es válida.

- Suscripción y notificación: los servidores se ponen en contacto con los directorios para indicar los cambios producidos en el servicio. Los directorios notifican estos cambios a todos los clientes que estén suscritos a notificaciones.

3.4. Revisión de protocolos actuales

En esta sección vamos a describir los protocolos de descubrimiento de servicios que se han propuesto en los últimos años dentro del IETF. Como hemos visto en el capítulo 2 existen otras propuestas, pero nos hemos centrado en el estudio de los protocolos de Internet por considerarlos los más genéricos ya que no están ligados a ningún entorno de computación distribuida ni a una determinada tecnología de red, su único requisito es la existencia de una pila IP.

3.4.1. Service Location Protocol

Service Location Protocol (SLP), RFC 2608 [Guttman et al., 1999b], es el principal resultado del *Service Location Protocol Working Group* (SVRLOC) del IETF. El objetivo del grupo fue la definición de un protocolo de descubrimiento automático de servicios en redes IP, descentralizado y extensible.

SLP tiene una arquitectura distribuida, y por escalabilidad del protocolo también soporta un modo de funcionamiento centralizado. Para definir servicios se emplea un nuevo esquema de URL: "`service:`", RFC 2609 [Guttman et al., 1999a].

Introducción

En SLP se definen tres tipos de agentes: los *User Agents* (UA), que son los que realizan el descubrimiento de servicios para satisfacer las necesidades que demandan las aplicaciones de los usuarios finales, los *Service Agents* (SA), que son los responsables de anunciar las características y localización de los servicios y los *Directory Agents* (DA), que son los responsables de almacenar información sobre los servicios que se están anunciando en la red y que actúan como directorios.

SLP tiene dos modos de funcionamiento, con DAs, en cuyo caso los SAs registran en ellos los servicios que ofrecen y los UAs buscan en ellos los servicios que precisan, siguiendo un funcionamiento centralizado; o sin DAs, en cuyo caso los UAs envían por difusión peticiones de un tipo de servicios, a las que los SAs que ofrecen el servicio responden mediante mensajes unicast, siguiendo un funcionamiento distribuido tipo *pull*.

Todos los agentes en SLP tienen asociado uno o varios ámbitos, que se emplean para crear entornos administrativos para el uso de servicios, de tal manera que los SAs y DAs sólo responden a búsquedas de servicios de UAs que pertenezcan a su mismo ámbito. Los diferentes ámbitos existentes en una red deben ser configurados por el administrador.

Formato de descripción de servicios

En SLP para describir un servicio se emplean esquemas URL. Aunque se puede emplear cualquier esquema URL, siempre que permita indicar la dirección en la que se encuentra un servicio, el SRVLOC ha definido en la RFC 2609 un esquema específico para la definición de servicios denominado `Service URL`. En esta RFC también se describe cómo se definen atributos asociados a un servicio.

Los atributos permiten al usuario o aplicación seleccionar el servicio que mejor se adapta a sus necesidades entre varios servicios del mismo tipo.

La información contenida en una `Service URL` debe ser suficiente para que un cliente pueda acceder al servicio descubierto. Su formato es el siguiente:

```
``service:''<service-type>``:''<site><url-part>
```

donde:

- `<service-type>` es el tipo de servicio,
- `<site>` es una dirección de red,
- `<url-part>` que puede estar compuesta por `<url-path>` con información de dónde localizar el servicio y `<attr-list>` con la lista de atributos que son requeridos para acceder al servicio.

El tipo de servicio puede ser un protocolo de red particular, en este caso se habla de tipo concreto, o un tipo abstracto, que agrupa en una categoría común a un conjunto de protocolos de red. El campo `<service-type>` tiene el siguiente formato:

```
[<abstract-type>]:<concrete-type>
```

donde `<abstract-type>` es opcional.

Para ejemplificar, a los tipos de servicio concretos para acceder a un servicio de impresión `lpr`, `ipp`, etc. se les asocia el tipo abstracto `printer`.

En SLP las búsquedas de servicios se realizan indicando el tipo de servicio requerido, en el que se puede incluir sólo el tipo abstracto, el tipo concreto o ambos.

Los servicios se deben definir formalmente en `Service Templates` en las que se incluyen al menos cinco elementos:

- Tipo de servicio: nombre y, opcionalmente, autoridad de nombrado.
- Versión: número de versión de la plantilla.
- Descripción: texto legible por personas en el que se describe el servicio proporcionado.

- Sintaxis de `url-part` en la `Service URL`.
- Definición de atributos: la definición de un atributo incluye los siguientes campos:
  - Identificador o nombre del atributo.
  - Descriptor, indicando el tipo del atributo (`string`, `boolean`, ...) y flags (si el atributo es opcional se indica con `o`, si es obligatorio se indica con `m`,...).
  - Lista de valores por defecto del atributo.
  - Descripción legible por personas del significado del atributo.
  - Lista de valores no permitidos del atributo.

  Cuando se transmite un atributo se envía la cadena `<identificador> = <valor>`.

Funcionamiento del protocolo

En el modo de funcionamiento distribuido, un UA solicita la búsqueda de un servicio, enviando por multicast un mensaje `SrvRqst`, los SAs que ofrecen este servicio responden a esta petición enviando un mensaje unicast `SrvRply`, Figura 3.4.

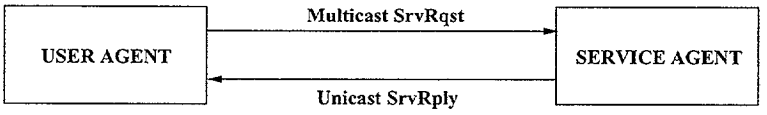


Figura 3.4: SLP modo distribuido.

Para el modo de funcionamiento distribuido se define un mecanismo de convergencia multicast, un UA puede transmitir mensajes consecutivos `SrvRqst` para una misma búsqueda, con el objetivo de obtener el mayor número de servicios de ese tipo disponibles en la red. Para minimizar el consumo de ancho de banda, se incluye en cada nuevo mensaje las direcciones de los servidores que han respondido en mensajes anteriores, para que no vuelvan a contestar, de esta forma se finaliza la búsqueda cuando no se obtiene ninguna respuesta o ha pasado un tiempo determinado, definido en el protocolo, desde la primera transmisión realizada.

En el modo de funcionamiento centralizado, un SA registra, mediante mensajes unicast `SrvReg`, todos los servicios que ofrece en los DAs existentes. Los

DAs contestan a estos registros con un mensaje unicast de asentimiento `SrvAck`. Por otra parte, los UAs envían por unicast búsquedas de servicios, `SrvRqst`, a los DAs, los cuales les responden con un mensaje unicast `SrvRply` listando todos los servicios que cumplen el criterio de búsqueda, Figura 3.5.

Figura 3.5: SLP modo centralizado.

SLP define varios mecanismos para que los UAs y los SAs descubran DAs de forma dinámica, el primero de ellos es el modo pasivo, en el que los SAs y los UAs escuchan los mensajes multicast, `DAAdvert`, enviados por los DAs, en los que anuncian su existencia periódicamente; y el segundo de ellos es el modo activo, en el que los SAs y UAs envían un mensaje multicast, `SrvRqst`, a los que los DAs responden enviando el mismo mensaje de anuncio, `DAAdvert`, por unicast, Figura 3.6.

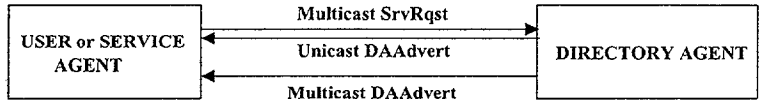


Figura 3.6: SLP descubrimiento de DAs.

También se contempla en el protocolo obtener las direcciones de los DAs de forma estática, empleando las opciones 78 y 79 de DHCP [Perkins y Guttman, 1999].

En SLP se definen una serie de mensajes opcionales que describimos brevemente a continuación:

- `SrvTypeRqst` y `SrvTypeRply` que se utilizan para buscar todos los tipos de servicios ofrecidos en la red. Se introducen estos mensajes para soportar aplicaciones tipo "buscador" de servicios: primero se solicitan todos los tipos de servicios y después, para cada tipo, se obtienen los servicios existentes en la red.

- `AttrRqst` y `AttrRply` que se utilizan para solicitar y obtener los atributos asociados a un servicio. Estos mensajes se envían por unicast entre el UA que solicita los atributos de un servicio que ha descubierto y el SA que ofrece ese servicio.

- `SrvDeReg` que se utiliza para que un SA puede eliminar servicios que tiene registrados en un DA y no quiere que sigan siendo anunciados. El DA debe responder a esta petición con un `SrvAck`.

Consideraciones de seguridad

En SLP se incluyen datos de autenticación en las descripciones de servicios (`URL Entries`) para garantizar que el que ofrece el servicio es una entidad confiable, para ello las `URL Entries` van firmadas empleando claves públicas. El algoritmo por defecto que deben implementar los agentes SLP es DSA con SHA-1.

La complejidad que esto supone, sobre todo en cuanto a la gestión de claves, hace que el soporte para seguridad sea una característica opcional del protocolo. Se indica explícitamente en la definición del protocolo que no se da soporte ni a integridad, ni a confidencial de los mensajes SLP y en caso de que una implementación así lo requiera, se delega a los mecanismos definidos en IPSec [Kent y Atkinson, 1998].

Uso de direcciones y puertos

SLP es un protocolo definido sobre UDP, opcionalmente los UAs cuando reciben mensajes de respuesta truncados pueden establecer una conexión TCP para obtener la respuesta completa de la búsqueda realizada. El puerto reservado es el 427 y se emplea la dirección IP multicast 239.255.255.253 con un TTL multicast por defecto de 255. En redes aisladas se puede trabajar con broadcast en vez de multicast.

Extensiones del protocolo

Se han propuesto una serie de extensiones sobre el protocolo inicial, la primera de ellas, RFC 3059 [Guttman, 2001], permite que los UAs puedan solicitar que en la respuesta a un mensaje de búsqueda se incluyan los atributos asociados al servicio. De esta forma, no es necesario que envíen un nuevo mensaje `AttrRqst` para obtenerlos.

En la RFC 3082 [Kempf y Goldschmidt, 2001], se introducen mecanismos de notificación y suscripción en SLP para permitir notificar los cambios que se producen en descripciones de servicios que estén almacenadas en los UAs o en los DAs, sin necesidad de que ellos realicen sondeos periódicos.

En la RFC 3421 [Zhao et al., 2001] se introducen extensiones para permitir que los UAs cuando realicen una búsqueda puedan indicar el número máximo de `URL Entries` del mensaje de respuesta o que éstas estén ordenadas según un determinado criterio indicado en el mensaje de búsqueda.

3.4.2. Simple Service Discovery Protocol

Simple Service Discovery Protocol (SSDP) [Goland et al., 1999] es un protocolo definido dentro de la iniciativa Universal Plug-and-Play (UPnP) para el descubrimiento dinámico de servicios. UPnP es una arquitectura propuesta para el descubrimiento, anuncio y uso de servicios en entornos dinámicos, centrada en estandarizar protocolos de comunicación basados en XML para la interacción entre los diferentes elementos de la arquitectura. El *UPnP Forum*[3] es el promotor de esta iniciativa, que está liderada por Microsoft.

SSDP se presentó como un draft al SVRLOC del IETF, con el objetivo de realizar una simplificación de SLP, y que de esta forma se pudiera implementar en un mayor número de dispositivos. En la actualidad este draft está obsoleto.

Introducción

En SSDP se distinguen dos tipos de elementos: clientes SSDP, que son los elementos que buscan servicios y los servidores SSDP, que son los elementos que ofrecen servicios. Los servidores SSDP envían por difusión un mensaje de anuncio de sus servicios, con el fin de notificar al resto de los dispositivos su presencia. Los clientes SSDP pueden almacenar de forma local estos anuncios y de esta forma conocer qué tipos de servicios están disponibles en la red. Además, los clientes SSDP también pueden descubrir servicios de forma activa, a través del envío por difusión de mensajes de búsqueda, a los que responden los servidores SSDP que ofrecen el servicio. Por lo tanto, SSDP soporta tanto el modo distribuido tipo *pull*, como el *push*.

Formato de descripción de servicios

Los servicios en SSDP están identificados por un par de URIs (Universal Resource Identifier) compuesto por el tipo de servicio y el nombre único del servicio, Unique Service Name (USN). Los USNs permiten diferenciar instancias de servicios del mismo tipo.

Los mensajes de respuesta a una petición y de anuncio de servicios contienen además información adicional sobre la instancia del servicio:

- Información de localización, que indica cómo se puede acceder a una instancia de un servicio. Esta información se expresa a través de URIs.

[3] http://www.upnp.org

- Información de tiempo de expiración, que indica cuánto tiempo puede un cliente SSDP almacenar de forma local en su caché la información asociada a una instancia de servicio.

Un cliente SSDP puede almacenar la información asociada a una instancia de un servicio de forma local, en la Figura 3.7 se muestra un ejemplo de caché de un cliente.

USN URI	Service Type URI	Expiration	Location
upnp:uuid:k91..	upnp:clockradio	3 days	http://foo.com/cr
uuid:x7z...	ms:wince	1 week	http://msce/win

Figura 3.7: Ejemplo de caché en un cliente SSDP.

La información que se recibe en un mensaje es prioritaria respecto a la que existe almacenada en la caché de un cliente SSDP, de tal forma que siempre que llegue un mensaje con información asociada a un determinado USN que ya existe en la caché, se eliminará dicha información y se sustituirá por la nueva información recibida.

Funcionamiento del protocolo

SSDP está construido sobre HTTP sobre UDP [Goland, 1999] tanto en su versión unicast, HTTPU, como multicast, HTTPMU.

Los servidores SSDP pueden anunciar los servicios que ofrecen para que los clientes SSDP conozcan su presencia en la red. Este mecanismo de anuncio se identifica como `ssdp:alive`. Los anuncios se envían sobre HTTPMU utilizando el método HTTP NOTIFY, definido en GENA[4] [Cohen y Aggarwal, 98]. Este mensaje debe contener obligatoriamente la cabecera `USN`, en la que se incluye el identificador único de instancia de servicio y la cabecera `NT`, en la que se incluye el tipo de servicio ofrecido. Además se debe indicar la localización del servicio a través de la cabecera `AL` y/o `Location` y el tiempo de expiración a través de la cabecera `cache-control: max-age` o de la cabecera `expires`. Si en el mensaje de anuncio no se proporciona información sobre el tiempo de expiración del

[4]Draft propuesto en el IETF para definir una arquitectura de envío y recepción de notificaciones sobre HTTP, en la actualidad está obsoleto.

servicio, un cliente SSDP no podrá almacenar en su caché local la información contenida en éste. En la Figura 3.8 se muestra un ejemplo de `ssdp:alive` para una instancia de un tipo de servicio `ge:fridge`.

```
NOTIFY * HTTP/1.1
Host: 239.255.255.250:reservedSSDPport
NT: ge:fridge
NTS: ssdp:alive
USN: uuid:solracetelos-2feb-11d0-a765-00a0c91e6bf6
AL: <blender:ixl><http://foo/bar>
Cache-control: max-age = 7393
```

Figura 3.8: Ejemplo de `ssdp:alive`.

Los clientes SSDP no responden con ningún mensaje a estos anuncios enviados por los servidores SSDP.

Del mismo que se anuncia la presencia de un servicio, los servidores SSDP pueden indicar que uno de sus servicios ya no está activo mediante un mecanismo similar que se identifica como `ssdp:byebye`. Los anuncios de cese de un servicio se envían sobre HTTPMU utilizando el método HTTP NOTIFY. Este mensaje debe contener obligatoriamente la cabecera `USN`, en la que se incluye el identificador único de instancia de servicio y la cabecera `NT`, en la que se incluye el tipo de servicio ofrecido. En la Figura 3.9 se muestra un ejemplo de `ssdp:byebye` para una instancia de un tipo servicio `ge:fridge`.

```
NOTIFY * HTTP/1.1
Host: 239.255.255.250:reservedSSDPport
NT: ge:fridge
NTS: ssdp:byebye
USN: uuid:solracetelos-2feb-11d0-a765-00a0c91e6bf6
```

Figura 3.9: Ejemplo de `ssdp:byebye`.

Cuando los clientes SSDP reciben este mensaje deben eliminar de sus cachés, si existe, la información almacenada relativa a la instancia del servicio con el USN indicado.

Los clientes SSDP no responden con ningún mensaje a estos anuncios enviados por los servidores SSDP.

Los clientes SSDP cuando quieren localizar una instancia de un tipo de servicio pueden consultar su caché local para ver si se ha anunciado algún servicio de tipo requerido, o pueden enviar de forma activa una petición de búsqueda. Las peticiones de búsqueda de servicios en SSDP se realizan utilizando el nuevo método HTTP M-SEARCH[5], este mecanismo de búsqueda se identifica como `ssdp:discover`. Esta petición debe incluir obligatoriamente la cabecera `ST` en la que se indica la URI correspondiente al tipo de servicio que el cliente quiere descubrir. El envío de estas peticiones se realiza sobre HTTPMU. En la Figura 3.10, se muestra un ejemplo de mensaje de petición de búsqueda del servicio `ge:fridge`.

```
M-SEARCH * HTTP/1.1
S: uuid:ijklmnop-20jan-11d0-a765-00a0c91e6bf6
Host: 239.255.255.250:reservedSSDPPort
Man: ssdp:discover
ST: ge:fridge
MX: 3
```

Figura 3.10: Ejemplo de petición de `ssdp:discover`.

Sólo los servidores SSDP que ofrezcan el tipo de servicio solicitado responden a esta petición por HTTPU. Las respuesta a un `ssdp:discover` debe indicar la localización del servicio, incluida en la cabecera HTTP `AL`. Opcionalmente puede incluir la cabecera `ST`, cuyo valor debe coincidir con el de mensaje de búsqueda, y la cabecera `USN` indicando el identificador único de instancia de servicio. En la Figura 3.11 se muestra un ejemplo de respuesta al mensaje de petición de búsqueda anterior.

Como se observa en el ejemplo, en la cabecera `AL` se pueden indicar varias URIs de acceso al servicio. En una respuesta a un `ssdp:discover` se debe incluir también información sobre el tiempo de expiración del servicio, para ello se emplean las cabeceras HTTP `cache-control: max-age` o `expires`, teniendo prioridad la primera respecto a la segunda en caso de existir las dos. Si una respuesta no contiene ninguna de estas dos cabeceras, el cliente SSDP no podrá almacenar en su caché la información contenida en ella.

SSDP permite que un cliente realice una búsqueda de todos los servicios disponibles en la red, para ello define un tipo de servicio especial identificado como

[5]Este método es una modificación del método SEARCH definido en [Reddy et al., 99].

```
HTTP/1.1 200 OK
S: uuid:ijklmnop-20jan-11d0-a765-00a0c91e6bf6
Ext:
Cache-Control: no-cache=''Ext'', max-age = 5000
ST: ge:fridge
USN: uuid:solracetelos-2feb-11d0-a765-00a0c91e6bf6
AL: <blender:ixl><http://foo/bar>
```

Figura 3.11: Ejemplo de respuesta a un `ssdp:discover`.

`ssdp:all`. La petición se realiza siguiendo el mecanismo `ssdp:discover` pero poniendo el valor de la cabecera ST a `ssdp:all`.

Consideraciones de seguridad

En SSDP no se llegó a especificar ningún mecanismo de seguridad, en el último draft se indicaba que se definiría en próximas versiones.

3.4.3. DNS Service Discovery

Los protocolos de descubrimiento de servicios tienen como objetivo localizar servidores que ofrecen un determinado tipo de servicio. Como hemos visto en los protocolos anteriores, como información de localización del servidor se proporciona, entre otros datos, su nombre de dominio, por lo tanto, para acceder al servicio se debe realizar una consulta DNS que nos dé la dirección IP del servidor.

Si no existe una infraestructura DNS, los protocolos anteriores deben trabajar con direcciones IP, lo que desde el punto de vista del protocolo no presenta ningún inconveniente pero si desde el punto de vista del usuario, ya que si existen varios servidores que ofrecen un mismo tipo de servicio, normalmente se le mostrará al usuario la información de localización obtenida para que éste seleccione el servidor al que prefiere acceder, si esta información es una lista de direcciones IP puede ser bastante frustrante y poco significativo para él.

Como consecuencia de esta dependencia, el grupo de trabajo Zeroconf del IETF ha propuesto emplear el protocolo DNS para descubrir servicios, de tal manera que los dispositivos no tienen porqué tener una implementación de DNS y otra del protocolo de descubrimiento de servicios, sino que con lo primera de ellas es suficiente.

Introducción

DNS Service Discovery (DNS-SD) [Cheshire, 2003a] surge con el objetivo de proporcionar soporte al descubrimiento de servicios a través de DNS, de tal manera que sin realizar ningún cambio a DNS, se puedan realizar consultas que permitan al usuario obtener una lista de instancias de un tipo determinado de servicio. En la actualidad se encuentra en estado de internet draft.

DNS-SD puede operar sobre DNS, por lo tanto, con una arquitectura centralizada basada en una estructura jerárquica de servidores; o sobre las modificaciones de DNS para la resolución de nombres en redes sin infraestructura, como Multicast DNS o LLMNR, con una arquitectura totalmente distribuida.

Formato de descripción de servicios

El tipo de registro de recursos SRV en DNS definido en la RFC 2782 [Gulbrandsen et al., 2000] permite localizar instancias de un determinado tipo de servicio. En la Figura 3.12 se muestra el formato de este tipo de registro de recurso.

```
_Service._Proto.Name TTL Class SRV Priority Weight Port Target
```

Figura 3.12: Formato del registro de recurso SRV.

donde,

- `Service`: nombre simbólico del servicio, que se corresponde habitualmente con el nombre reservado en el IANA para un protocolo a nivel de aplicación.
- `Proto`: nombre simbólico del protocolo de transporte. Los valores habituales para este campo son `_tcp` o `_udp`.
- `Name`: dominio al que se refiere el registro de recurso.
- `TTL`: tiempo de vida asociado al registro de recurso, y que indica el tiempo que se puede almacenar en la memoria de un sistema remoto.
- `Class`: clase de registro, habitualmente IN.
- `Priority`: prioridad asociada al registro de recurso. El valor que puede tomar está en el rango 0-65535.

- `Weight`: peso relativo asociado a entradas con la misma prioridad. A mayor peso, mayor es la probabilidad de seleccionar una instancia. El valor que puede tomar está en el rango 0-65535.

- `Port`: puerto en el que se proporciona el servicio.

- `Target`: nombre de dominio para el host que ofrece el servicio.

La definición de este nuevo tipo de registro de recurso tiene como objetivo facilitar a los administradores la tarea de mover un servicio de un host a otro, con el menor impacto posible, y balancear la carga de los servidores designando algunos como primarios y otros como secundarios. Desde el punto de vista del cliente todas las instancias que se listan en una consulta DNS del tipo SRV ofrecen el mismo servicio, y para seleccionar a cuál se accede, se debe elegir siempre la de menor prioridad, y a igual prioridad se elige una empleando un mecanismo pseudoaleatorio ponderado por el peso relativo asociado.

DNS-SD es una iniciativa conjunta del grupo Zeroconf y el grupo DNSEXT. Consiste en emplear la sintaxis y semántica de los registros de recurso SRV para el descubrimiento de servicios, pero añadiendo un nivel de indirección, para que a través de consultas DNS el usuario pueda obtener instancias de tipos de servicios de distintas características.

El descubrimiento de servicios en DNS-SD se divide en dos pasos:

- **Enumeración de instancias de servicios**: en vez de realizar una consulta DNS del tipo SRV para `<Service>.<Domain>` se realiza una consulta del tipo PTR, en la que se obtienen los nombres de instancias de servicios con el siguiente formato:

  ```
  <Instance>.<Service>.<Domain>
  ```

  donde `Instance` es una etiqueta DNS, que contiene un nombre comprensible para el usuario y que identifica la instancia del servicio, `Service` formado por dos etiquetas DNS, una indicando el nombre del protocolo de aplicación y la otra indicando el nombre protocolo de transporte [6], y `Domain` que es el nombre de dominio en el que se buscan instancias del servicio.

[6] en la nomenclatura empleada en el RFC 2782 este campo se corresponde con los campos `_Service._Proto`

- **Resolución de nombres de instancia de servicios**: para acceder a una instancia de un servicio descubierta en el paso anterior, se realiza una consulta DNS de tipo SRV del nombre de la correspondiente instancia. En caso de obtenerse varias respuestas, se seguirán la reglas de prioridad y peso definidas en la RFC 2782.

Para acceder a determinados servicios es necesario más información además de la dirección IP y del número de puerto, por ejemplo, para imprimir vía el protocolo `lpr` normalmente es necesario el nombre de la cola de impresión. Además, en ciertas ocasiones el usuario quiere conocer datos específicos sobre una instancia de un servicio para poder seleccionar el que más le interesa, por ejemplo, si la impresora imprime en color o no. Para suplir esta necesidad, en DNS-SD se utilizan los registros de recursos DNS de tipo TXT para almacenar, en forma de atributos, la información adicional asociada a una instancia de un servicio.

El significado de los atributos depende del tipo de servicio concreto, por lo que en DNS-SD sólo se define la sintaxis que se debe utilizar, que consiste básicamente en almacenar en un string un atributo con el formato `nombre=valor`, el conjunto de strings formará el registro DNS-TXT correspondiente. En general, para cada tipo de servicio se debe definir una serie de atributos comunes, y esto es lo que se denomina en el draft como perfil DNS-SD.

Para que en DNS-SD se dé soporte a aplicaciones tipo buscador, se define una meta-consulta especial, que consiste en una consulta DNS de tipo PTR con el nombre `_services._mdns._udp.<Domain>` como resultado se obtiene una lista de registros PTR cuyo campo `rdata` contiene el tipo de servicio.

Funcionamiento del protocolo

La información de nombres de dominio se gestiona a través de una estructura jerárquica de servidores DNS. Los clientes DNS cuando quieren realizar consultas se ponen en contacto con el servidor DNS que tienen configurado. Las consultas se pueden realizar de dos formas: recursivas, en las que si el servidor al que se le hace la consulta no puede resolver el nombre solicitado, se pone en contacto con otros servidores DNS para que la resuelvan; o iterativas, en las que si el servidor al que se le hace la consulta no puede resolver el nombre solicitado, responde al cliente con el nombre del siguiente servidor de DNS en la jerarquía al que le pueden hacer la consulta.

En una consulta DNS se indica el nombre de dominio que se quiere resolver y el tipo de información (tipo de registro de recurso) asociada a ese nombre de dominio que se quiere obtener. En DNS-SD, se realizan consultas de cuatro tipos:

PTR para obtener la lista de instancias de un tipo de servicio en un determinado dominio, SRV para obtener el nombre del servidor y el número de puerto de una instancia, A/AAAA para obtener la dirección IP del servidor y, opcionalmente, TXT para obtener atributos asociados con una instancia de un servicio. En una respuesta DNS el servidor puede incluir datos adicionales, normalmente cuando se realiza la primera consulta PTR, el servidor podrá incluir las demás (SRV, A/AAAA y TXT) como datos adicionales, reduciendo el número de paquetes intercambiados entre servidor y cliente.

Los registros de recursos PTR, SRV y TXT asociados a instancias de servicios deben ser dados de alta en la base de datos del correspondiente servidor DNS, esta tarea puede delegarse a los administradores o emplear DNS dinámico, RFC 2136 [Vixie et al., 2000], para que sean los propios servidores los que automáticamente registren sus propios registros en el servidor DNS.

DNS es un protocolo que precisa infraestructura de red y una fuerte gestión administrativa debido a la normativa existente en cuanto a la asignación de nombres de dominio. Independientemente de que DNS se emplee para el descubrimiento de servicios o no, parece claro que se deben proporcionar soluciones para que en redes sin infraestructura, redes ad-hoc, exista un mecanismo eficiente de resolución de nombres. En la actualidad existen dos propuestas activas en el IETF sobre este tema, ambas en fase de draft, y que pueden emplearse también para el descubrimiento de servicios basado en DNS:

- Multicast DNS [Cheshire, 2003b]
- Linklocal Multicast Name Resolution [Esibov et al., 2003]

Ambas propuestas parten del protocolo DNS pero eliminan su arquitectura centralizada basada en una estructura jerárquica de servidores, para sustituirla por una arquitectura totalmente distribuida, en la que todos los dispositivos que forman la red poseen su propio servidor DNS, y en las que las consultas se transmiten por multicast. Analizamos a continuación cada una de ellas.

Multicast DNS Multicast DNS es, al igual que DNS-SD, una iniciativa conjunta de los grupos Zeroconf y DNSEXT. En Multicast DNS se define un nuevo dominio de alto nivel denominado `.local.`, todos los nombres que pertenecen a este dominio sólo tendrán significado en la red local en la que han sido definidos. No existe una autoridad de nombrado encargada de gestionar este dominio, sino que cualquier usuario o software puede crear sus propios nombres con el sufijo `.local.`, siempre y cuando no entren en conflicto con otro.

Cuando se solicita la resolución de un nombre con sufijo `.local.` debe emplearse el protocolo Multicast DNS. En todos los dispositivos de la red existe un "cliente Multicast DNS", que realiza peticiones de resolución de nombres por multicast y un "servidor Multicast DNS", que resuelve las peticiones realizadas por otros clientes.

En Multicast DNS se considera que las aplicaciones que solicitan la resolución de un nombre de dominio pueden tener tres modos de funcionamiento:

- Una petición – una respuesta: el cliente sólo tiene en cuenta la primera respuesta recibida a su petición y descarta cualquier otra.
- Una petición – varias respuestas: el cliente espera a recibir varias respuestas a una petición, tantas como considere necesarias, para ello puede repetir la misma petición varias veces.
- Peticiones continuas: el cliente realiza la misma petición de forma periódica para monitorizar la existencia del recurso en la red.

Para reducir el tráfico que puede suponer en la red las consultas de los dos últimos tipos, en las peticiones se incluyen las posibles respuestas ya conocidas por el cliente (almacenadas en la caché), de tal manera que un servidor no responderá si no conoce alguna respuesta no incluida en la petición. Para incluir las respuestas conocidas se emplea la sección de respuestas de un paquete DNS. Si todas las respuestas conocidas no caben en único paquete, debe indicarse con el bit correspondiente en la cabecera y a continuación, enviar las restantes en una nueva respuesta que no incluye pregunta.

Los servidores responden por multicast, de tal manera que todos los dispositivos de la red pueden conocer las respuestas proporcionadas por éstos y de esta forma tener actualizadas sus cachés. Además, esto permite detectar rápidamente conflictos entre nombres de dominio empleados por los distintos dispositivos. Para evitar colisiones los servidores retardan sus respuestas un tiempo aleatorio generado uniformemente entre 20–120 ms.

El TTL[7] recomendado en Multicast DNS es 120 minutos, lo que supone que un cliente puede tener información errónea en la caché hasta un máximo de 2 horas. Disminuir el TTL permitiría reducir el tiempo que permanecen datos erróneos en la caché, pero aumentaría notablemente el tráfico en la red.

Para disminuir el número de entradas erróneas en la caché se establecen varios mecanismos:

[7]TTL definido en DNS

- Mensajes de "goodbye": si un servidor detecta que va a abandonar la red o se va a apagar, envía un mensaje de respuesta gratuito (sin petición asociada) incluyendo todos sus registros de recursos pero con un TTL = 0, por lo que cuando se actualicen las cachés en los clientes, se borrarán inmediatamente.

- Actualización de entradas: si existe algún cambio en un registro de recurso, el servidor envía un mensaje de respuesta gratuita indicando los cambios producidos.

- Borrado de entradas en la caché local:

  - Si se detecta un fallo al emplear un registro de recurso (por ejemplo, acceder a un servicio en el caso de un registro de recurso del tipo SRV), la aplicación que lo detecta debe poder indicarle a la capa de Multicast DNS que se borre esa entrada de la caché local.
  - Si se detecta un cambio en la topología de la red, se debe eliminar todas las entradas de la caché o disminuir suficientemente su TTL para que expiren en pocos segundos.

Siempre que se detecte algún conflicto en un registro de recurso es necesario resolverlo, el draft explica un mecanismo automático que permite que los dispositivos cambien sus nombres, sin necesidad de intervención manual del usuario o administrador del sistema.

Linklocal Multicast Name Resolution Linklocal Multicast Name Resolution (LLMNR) es una iniciativa del grupo DNSEXT. Los objetivos que pretende alcanzar son los mismos que en el caso de Multicast DNS, es decir, permitir la resolución de nombres en entornos en los que no está disponible un servidor DNS tradicional. Su manera de abordar el problema es mucho más conservadora que la propuesta anterior, no modificándose el uso de los campos de los paquetes DNS ni en las peticiones ni en las respuestas y no definiendo un nuevo dominio de ámbito local.

En LLMNR, los dispositivos tienen un "cliente LLMNR", que realiza peticiones de resolución de nombres y un "servidor LLMNR", que resuelve las peticiones realizadas por los clientes. Las peticiones se envían habitualmente por multicast, excepto en los siguientes casos que se transmiten por unicast:

- Si la respuesta recibida a una petición por multicast tiene el bit de la cabecera TC activo, el cliente puede repetir la consulta por unicast a través de una conexión TCP para obtener todas las respuestas posibles.

- Si la consulta realizada se corresponde con una resolución del tipo PTR para una dirección IP en la zona "in-addr.arpa" o "ip6.arpa".

Cuando un cliente envía una petición por multicast debe esperar un tiempo, denominado en el draft `LLMNR_TIMEOUT`, para recibir todas las respuestas posibles. Si no recibe ninguna, puede retransmitir la petición hasta un máximo de tres veces. En el caso de que la petición se transmita por unicast, sólo esperará ese tiempo si no recibe ninguna respuesta.

Los servidores siempre responden a las peticiones de los clientes por unicast, y sólo con registros de recursos para los que están autorizados, y por lo tanto, nunca con los almacenados en su caché. En LLMNR se restringe el concepto de autorización respecto al empleado en DNS: un servidor DNS tiene como zona autorizada todos los nombres de dominio debajo de su raíz, excepto los que estén delegados en zonas separadas; por el contrario, un servidor LLMNR sólo está autorizado para la raíz de su zona.

En LLMNR se emplea un mismo TTL para todos los registros de recursos que envía un servidor en sus respuestas. Este TTL debe ajustarse dependiendo del dinamismo de la red, así en redes muy dinámicas, como las redes ad-hoc, se recomienda que su valor sea cero, y que se aumente para redes menos dinámicas. El principal incoveniente es que cuanto menor sea el TTL mayor será el tráfico generado en la red.

Tanto los clientes como los servidores LLMNR realizan chequeos sobre la dirección origen de las respuestas y peticiones recibidas, antes de aceptarlas o descartarlas. Así, un cliente sólo acepta respuestas de servidores cuya dirección IP sea *on-link*, es decir, pertenezca a la misma subred, y un servidor sólo responde peticiones unicast que provengan de una dirección *on-link* o peticiones enviadas a direcciones multicast de ámbito local. Además, un servidor en sus respuestas debe incluir sólo registros de recursos que sean alcanzables en la misma subred.

En LLMNR también se pueden producir conflictos entre registros de recursos que deben ser únicos, en caso de que estos se produzcan en el draft se indica que deben ser resueltos por el administrador de la red.

Consideraciones de seguridad

En el draft de DNS-SD se indica que se debe emplear DNSSEC, RFC 2535 [Eastlake, 1999], para garantizar la autenticidad de la información proporcionada por los dispositivos en la red.

En las versiones de DNS en redes sin infraestructura se reflexiona sobre el

problema que plantea introducir mecanismos de seguridad en este tipo de redes, en las que por sus propias características no siempre se pueden establecer relaciones de confianza entre los dispositivos que las forman. Las consideraciones concretas realizadas en cada una de las propuestas son:

- En Multicast DNS se aconseja el uso de firmas digitales para distinguir paquetes de dispositivos confliables de los que no los son, para ello se recomienda el uso de los mecanismos de firma definidos en IPSEC y/o DNSSEC.
- En LLMNR se indica que no es necesario el uso de autenticación, y en caso de que una implementación así lo precise se aconseja emplear IPSEC.

  Además, en LLMNR se realizan otras consideraciones sobre seguridad como el uso de cachés independientes por cada interfaz de red, para que en el caso de que se produzcan ataques de llenado de caché con registros de recursos falsos, no afecten a todos las interfaces y la utilización de TTL a nivel IP de valor 1 para que los mensajes no se propaguen a otras subredes.

3.5. Conclusiones

En este capítulo hemos realizado un estudio teórico del problema del descubrimiento de servicios, analizando las diferentes soluciones teóricas que existen. Primero, hemos visto las de funcionamiento distribuido, denominadas modo *push*, en la que los servidores se anuncian periódicamente y así los clientes los descubren, y modo *pull*, en la que los clientes son los que preguntan por servicios cuando los necesitan y son los servidores los que responden a estas peticiones. Por último, hemos descrito la solución centralizada, en la que se introducen elementos en la red que actúan como servicios de directorio, en los que los servidores registran sus servicios y a los que los clientes solicitan las búsquedas, de esta forma se mejora la escalabilidad del mecanismo de descubrimiento en redes amplias.

A continuación, hemos descrito de forma detallada las propuestas realizadas estos últimos años en el IETF para el descubrimiento de servicios de forma dinámica, por ser las soluciones más genéricas que se han realizado ya que su único requisito es que en los dispositivos exista una pila IP. Así, hemos analizado el protocolo SLP, único estándar en la actualidad, el protocolo SSDP que se planteó como una simplificación a SLP para ser implementado en dispositivos limitados pero que no pasó del estado de draft, y por último, las recientes iniciativas de los grupos DNSEXT y ZeroConf basadas en modificaciones a DNS y que soportan modos de funcionamiento adaptados tanto a redes amplias como a redes sin infraestructura.

Uno de los objetivos planteados en la realización de esta tesis doctoral es la definición de un protocolo de descubrimiento adaptado a los retos que plantean los escenarios de computación ubicua, caracterizados fundamentalmente por:

- Ámbito local: en la computación ubicua la distancia vuelve a ser importante, un concepto que con Internet había desaparecido, ya que los dispositivos van a tener más necesidad de interactuar cuanto más próximos se encuentren entre sí.

- Limitaciones de los dispositivos: los avances de la electrónica hacen que cada vez existan más dispositivos con capacidad de computación y comunicación que pueden ofrecer nuevos servicios al usuario. Pero estos dispositivos tienen limitaciones tanto de memoria como de procesamiento, y en los casos en los que funcionan con baterías es importante reducir el consumo energético para aumentar su autonomía.

- Entornos cambiantes: los usuarios son cada vez más móviles y entran y salen continuamente de espacios diferentes, en los que se les ofrecen nuevos servicios. Por lo tanto, es necesario que sus dispositivos personales, que les permiten interaccionar con el entorno que les rodea, puedan comunicarse y descubrir los servicios ofrecidos por los demás dispositivos sin necesidad de configuraciones previas, y sin depender de conectividad a nivel global.

Los protocolos que hemos analizado en este capítulo no cubren totalmente estas necesidades, y veremos en cada caso el porqué:

- SLP es un protocolo diseñado para redes fijas de ámbito local y global, en el que la solución distribuida diseñada para redes locales funciona siguiendo un modo *pull*. El principal inconveniente de aplicar esta solución a entornos ubicuos es que un modo *pull* consume un gran ancho de banda, lo que supone un consumo de baterías excesivo.

- SSDP es un protocolo diseñado teniendo en cuenta las limitaciones de los nuevos dispositivos, introduce el uso de cachés para reducir el número de mensajes transmitidos, algo opcional en SLP, y soporta tanto un funcionamiento distribuido tipo *pull* como *push*, pero no indica cuándo debe emplearse cada uno de ellos. El principal inconveniente de esta solución es precisamente este último, ya que no se realiza ninguna optimización en cuanto a número de mensajes transmitidos.

- DNS-SD es una propuesta muy reciente que todavía sigue estando en versión de draft pero de forma activa. El principal inconveniente de esta solución es

que al basarse en DNS la descripción de los servicios es poco flexible porque debe estar basada en registros de recursos.

Las modificaciones a DNS para su aplicación a redes sin infraestructura, Multicast DNS y LLMNR, sobre las que podría emplearse DNS-SD, cubren algunas de las necesidades de los entornos de computación ubicua, pero adaptadas al problema que abordan, que es la resolución de nombres y no el descubrimiento de servicios.

Por lo tanto, es necesario definir un nuevo protocolo de descubrimiento de servicios que proporcione una solución eficiente para entornos de computación ubicua. En el próximo capítulo realizaremos un diseño razonado en el que veremos cómo la solución propuesta cubre los requisitos que impone este nuevo paradigma de la computación.

Como se indica en la sección 3.2 de este capítulo, no es objetivo de esta tesis doctoral definir un formato de descripción de servicios, y el mecanismo de descubrimiento que vamos a proponer es independiente de este formato y de cómo se realiza la correspondencia entre el servicio buscado y el servicio ofrecido. Independientemente de esto, no queremos finalizar este capítulo sin aclarar que nuestra interpretación de servicio es *"una entidad software o hardware que reside en un dispositivo que realiza una función y/o proporciona una información"*.

Capítulo 4

Propuesta para el descubrimiento de servicios en entornos de computación ubicua

Después del estudio realizado en el capítulo anterior, hemos visto que los protocolos que se han definido estos últimos años no cubren las necesidades impuestas por los nuevos entornos de computación ubicua. Por ello a lo largo del desarrollo de la tesis, nos planteamos un nuevo objetivo, que se ha convertido en una de las principales contribuciones de la misma: la definición de un nuevo protocolo de descubrimiento de servicios, que hemos denominado Pervasive Discovery Protocol (PDP).

PDP es un protocolo de descubrimiento de servicios adaptado a entornos de computación ubicua, que permite a los dispositivos obtener información sobre la existencia y localización de servicios proporcionados en el entorno próximo que les rodea. PDP elimina la necesidad de que el usuario conozca previamente el tipo y los servicios que pueden proporcionarle otros dispositivos con los que puede comunicarse en algún momento. El usuario, a través de sus aplicaciones, podrá solicitar la búsqueda de un tipo de servicio al que quiere tener acceso, y si así lo requiere, podrá conocer también todos los servicios que los dispositivos que le rodean pueden ofrecerle.

Este capítulo lo hemos estructurado de la siguiente forma: en la sección 4.1 planteamos el problema de descubrimiento de servicios en entornos ubicuos, en la sección 4.2 enumeramos los objetivos que hemos seguido al realizar el diseño del nuevo protocolo. Posteriormente, en la sección 4.3, describimos la solución propuesta, es decir, el algoritmo que describe el funcionamiento del protocolo PDP, justificando de forma detallada y apoyada en estudios analíticos y de simu-

lación el diseño realizado, sección 4.4. Finalizamos, con las secciones 4.5, 4.6 y 4.7 en las que se realizan algunas consideraciones sobre el diseño de PDP y con las conclusiones en la sección 4.8.

4.1. Planteamiento del problema

En entornos de computación ubicua es imprescindible proporcionar un mecanismo de descubrimiento y anuncio de servicios, debido al dinamismo de estos entornos y al mayor número y diversidad de dispositivos que aparecen y que pueden ofrecer y demandar servicios. Además, la distancia en estos escenarios es importante, a medida que un dispositivo se aleja de una zona, las interacciones con sus vecinos deben decaer a su vez. Por ejemplo, no es deseable que sigamos intercambiando mensajes con un sensor de temperatura cuando hemos abandonado la sala en la que se encuentra.

Con este ejemplo, vemos que en computación ubicua dos dispositivos van a tener más necesidad de interactuar cuanto más próximos se encuentren, mientras que la necesidad de comunicación entre ellos será menos probable cuanto más alejados estén. Satyanarayanan [Satyanarayanan, 2001] formula esta necesidad de comunicación, por analogía con las leyes de la física, como inversamente proporcional al cuadrado de la distancia. Por lo tanto, el protocolo de descubrimiento de servicios a diseñar tendrá aplicación en ámbitos locales, y esta característica se tendrá en cuenta en su diseño.

Algunos de los protocolos de descubrimiento de servicios que han aparecido en los últimos años, han tenido en cuenta la movilidad de los dispositivos que forman la red, pero la mayoría no han considerado las limitaciones tanto hardware como software de éstos, ni la posibilidad de que la redes ad-hoc que se forman entre ellos de manera espontánea cuando se aproximan unos a otros, no incluyan ningún elemento fijo sin limitaciones, tipo PC.

En general, estos protocolos se pueden aplicar tanto a ámbitos locales como globales, algunos de ellos, como SLP o DNS-SD tienen versiones adaptadas para cada caso. Para ámbitos globales, por escalabilidad del protocolo, aparece un directorio de servicios (SLP, Salutation, Jini,...) en el que los que ofrecen servicios se registran y al que los que demandan servicios preguntan cuando quieren encontrar algun servicio determinado. En entornos de computación ubicua, el problema de escalabilidad no es prioritario debido al ámbito local de aplicación, y además no es posible que existan servidores centrales, ya que las redes son muy dinámicas y en algunos casos, los dispositivos que las forman tienen capacidades de almacenamiento limitadas, por lo que es muy costoso que uno de ellos afronte

el papel de servidor.

Los protocolos para ámbitos locales, como SSDP o las versiones de SLP o DNS-SD, no emplean servicio de directorio, pero realizan un uso excesivo de transmisiones para permitir que los clientes, en busca de un servicio, lo busquen en la red o para que los que proporcionan servicios se anuncien periódicamente. Esto supone un gran coste de baterías, recurso muy crítico en la mayoría de dispositivos limitados.

El mecanismo más empleado para minimizar transmisiones en soluciones distribuidas es el uso de cachés, en las que cada dispositivo de forma local almacena las descripciones de servicios escuchadas anteriormente en la red, de manera que cuando se busca un servicio, antes de transmitir el mensaje de búsqueda, se comprueba si existe en la caché y si es así, no se realiza ninguna transmisión. El problema que plantea el uso de cachés es que algunas de las entradas almacenadas se correspondan con servicios falsos, es decir, con servicios que ya no existen en la red. Este problema es mayor en entornos dinámicos, por lo que estos errores deben minimizarse introduciendo mecanismos de consistencia de cachés que no aumenten en exceso el tráfico generado.

4.2. Objetivos de diseño

Teniendo en cuenta el problema que queríamos abordar, se diseñó un nuevo protocolo de descubrimiento, el PDP, con el objetivo de definir un algoritmo de descubrimiento de servicios que se adapte a entornos de computación ubicua, y su diseño se ha basado en alcanzar los siguientes objetivos:

- **Minimizar el número de transmisiones**

  Para que se adapte a las restricciones de los protocolos inalámbricos y reduzca el consumo de batería de los dispositivos limitados.

- **Funcionar sin infraestructura fija**

  Para que pueda funcionar en redes ad-hoc en la que los dispositivos entran y salen de forma espontánea, y en las que el funcionamiento de la red no depende de ningún elemento central que precise administración.

- **Adaptarse tanto a entornos dinámicos como estáticos**

  Para que pueda obtener las mejores prestaciones adaptándose al entorno en el que se encuentra y a las características de los dispositivos que lo forman, por ejemplo, si existen dispositivos con menores limitaciones, tipo PC, que el funcionamiento del protocolo se apoye más en ellos.

- **Adaptarse a las características de las aplicaciones**

  Para que se puedan obtener mejores prestaciones teniendo en cuenta las necesidades de las aplicaciones que realizan las búsquedas de servicios en la red, y que son, en última instancia, las que hacen uso del protocolo.

- **Maximizar la cooperación entre los dispositivos**

  Para que puedan obtener un mayor beneficio reduciendo el coste, ya que si cada uno comparte con los demás la información sobre los servicios que ha descubierto o avisa de los que ya no están disponibles en la red, la visión del entorno que tiene cada uno de ellos será más acertada.

- **Simple y poco costoso computacionalmente**

  Para que sea implementable en dispositivos con restricciones de capacidad de proceso.

4.3. Solución propuesta

PDP es un protocolo de descubrimiento de servicios de ámbito local, totalmente distribuido, cuyo funcionamiento no depende de directorios. Los servicios se descubren bajo demanda, es decir, hasta que algún dispositivo lo solicite no se transmite por la red un mensaje indicando la localización del servicio. Todos los mensajes, incluidos los de respuesta, se transmiten por difusión de tal forma que los demás dispositivos aunque no hayan solicitado el servicio, pueden conocer la existencia de éste. Cada dispositivo almacena en una caché local los servicios que va escuchando. La información almacenada en esta caché se consulta cada vez que una aplicación de usuario solicita un servicio, con el objetivo de minimizar el número de transmisiones necesarias para satisfacer la búsqueda.

La base fundamental del protocolo es que, la información que almacena cada dispositivo en su caché se comparte entre todos los dispositivos que forman la red, para que de esta forma todos tengan una visión acertada de los servicios que se ofrecen en el entorno con un mínimo coste. Esta idea se basa en el hecho de que la unión de la información almacenada en cada una de esas cachés, contiene la misma información que tendría un directorio de servicios, pero en este caso está distribuida entre todos los dispositivos de la red.

Además, cuando se transmiten peticiones de búsqueda, los dispositivos incluyen los servicios ya conocidos y almacenados en sus cachés, de tal forma que sólo responden los dispositivos que conocen nuevos servicios. Para minimizar el

número de mensajes de respuesta transmitidos, se prioriza que responda primero el dispositivo que conoce más servicios nuevos.

En última instancia PDP es un protocolo que emplean las aplicaciones de usuario para encontrar servicios, dependiendo de las características de estas aplicaciones se puede mejorar las prestaciones del protocolo. Así, hemos distinguido tres tipos de aplicaciones según el tipo de búsquedas que realizan:

- Aplicaciones que quieren localizar algún dispositivo que ofrece un determinado servicio, pero sin importar cuál es el dispositivo que lo ofrece. Por ejemplo, si queremos conocer la temperatura de un sala y en ella existen varios sensores de temperatura, nos da igual cuál de ellos nos proporciona esta información.
- Aplicaciones que quieren localizar todos los dispositivos que ofrecen un determinado servicio en el entorno, para elegir cuál es el que mejor se adapta a sus necesidades. Por ejemplo, si queremos imprimir un documento, nos gustará saber qué impresoras existen en nuestro entorno para seleccionar una de ellas, por ejemplo, la que imprime en color y a doble cara.
- Aplicaciones tipo buscador, que quieren localizar todos los servicios que se ofrecen en la red.

Una vez descritas las principales características del protocolo diseñado, en las siguientes apartados realizamos una explicación formal del mismo, presentando primero el escenario en el que se aplica y algunas de las consideraciones que se deben tener en cuenta.

4.3.1. Escenario de aplicación

Consideramos que en un entorno de computación ubicua, existen `D` dispositivos formando una red, cada uno de ellos ofrece `S` servicios, que mantiene almacenados en un array, `Local`, y espera estar disponible en el entorno durante `T` segundos. Este tiempo `T`, denominado *tiempo de disponibilidad*, está previamente configurado en el dispositivo y depende de sus características de movilidad.

Cada dispositivo tiene un Agente de Usuario PDP, denominado PDP_UA (PDP User Agent) y un Agente de Servicio, denominado PDP_SA (PDP Service Agent). El PDP_UA es un proceso que emplean las aplicaciones de usuario para buscar los servicios ofrecidos en la red. El PDP_SA es un proceso que da a conocer los servicios ofrecidos por un dispositivo. El PDP_SA siempre incluye el tiempo de disponibilidad del dispositivo en sus anuncios.

Cada dispositivo tiene una caché, `Cache`, que contiene una lista de los servicios que han sido anunciados en la red. Cada elemento de esta caché tiene dos campos: *descripción del servicio* y *tiempo de expiración*. El tiempo de expiración es el tiempo estimado que el servicio permanecerá disponible en la red y se calcula como el mínimo entre el tiempo de disponibilidad local (del dispositivo donde se almacena), y el tiempo de disponibilidad del dispositivo que lo anuncia. Cuando finaliza su tiempo de expiración las entradas se borran de la caché. Además, esta caché tiene un número limitado de entradas, por lo que se eliminarán aquellas con un tiempo de expiración menor, cuando no haya espacio para almacenar entradas nuevas.

Si un dispositivo tiene varias interfaces de red, por ejemplo, IEEE 802.11b y Bluetooth, tendrá una caché asociada a cada interfaz de manera independiente, de manera que en esa caché sólo se almacenan servicios que han sido anunciados en esa interfaz de red.

4.3.2. Descripción del algoritmo

PDP tiene dos mensajes obligatorios: `PDP_Service_Request`, que se emplea para enviar peticiones de búsqueda de servicios, y `PDP_Service_Reply`, que se utiliza para responder a `PDP_Service_Request` y permite anunciar los servicios disponibles en la red. En PDP se define el mensaje opcional `PDP_Service_Deregister`, que permite informar a los dispositivos de la red que un servicio ya no está disponible.

Agente de Usuario PDP

Cuando una aplicación o el usuario final de un dispositivo necesita un servicio de un tipo determinado, se lo solicita a su PDP_UA. En PDP hemos definido dos tipos de peticiones, según el tipo de aplicación que solicita la búsqueda:

- **una petición–una respuesta** (`1/1`): la aplicación necesita conocer un dispositivo, que ofrece un servicio determinado, pero le da igual cuál.

- **una petición–varias respuestas** (`1/n`): la aplicación necesita descubrir todos los dispositivos que ofrecen un tipo de servicio en la red. En este tipo de petición, definimos un tipo especial de servicio, denominado `ALL`, que da soporte a aplicaciones tipo buscador, que quieren descubrir todos los servicios, de cualquier tipo, disponibles en la red.

Los dos tipos de búsqueda emplean el mismo mensaje, `PDP_Service_Request`, y mediante un flag en la cabecera se indica si es `1/1` ó `1/n`.

El comportamiento del PDP_UA cuando la búsqueda es del tipo "una petición–una respuesta" se detalla en la Figura 4.1. El PDP_UA busca un servicio del tipo `service_type` en la lista de servicios locales y en su caché. Si lo encuentra, responde a la aplicación con la correspondiente descripción del servicio sin necesidad de realizar transmisiones por la red. Si por el contrario el servicio no se encuentra, el PDP_UA envía por difusión un `PDP_Service_Request` y espera `CONFIG_WAIT_RPLY` segundos por mensajes de respuesta. Si se producen, el PDP_UA le proporciona a la aplicación las descripciones de servicios recibidas y en caso contrario, le indica que el servicio solicitado no está disponible.

```
searchAny(service_type) {
        foreach (s ∈ Local)
          if (s.type==service_type) return(s);
        foreach (e ∈ Cache)
          if (e.type==service_type) return(e);
        broadcast(PDP_Service_Request(service_type, 1/1));
        set_timer(CONFIG_WAIT_RPLY, EXPIRED);
        service_remote= hear_network(PDP_Service_Reply);
        return(service_remote);
    EXPIRED:
        return(NULL);
}
```

Figura 4.1: PDP_UA – implementación en pseudocódigo de "una petición–una respuesta".

En la Figura 4.2 se detalla el comportamiento del PDP_UA cuando el tipo de búsqueda es "una petición–varias respuestas". El PDP_UA construye una lista de servicios conocidos del tipo especificado, es decir, una lista con los servicios de ese tipo ofrecidos localmente y almacenados en la caché. En el caso en el que el tipo de servicio sea `ALL`, la lista de servicios conocidos contendrá todos los servicios locales más todos los de la caché. A continuación, el PDP_UA envía un `PDP_Service_Request` incluyendo esta lista, espera `CONFIG_WAIT_RPLY` segundos y cuando finaliza este tiempo, responde a la aplicación con la descripción de los servicios de la lista de servicios conocidos, más la de los nuevos servicios recibidos, si han llegado mensajes de respuesta.

Los PDP_UAs en todos los dispositivos están continuamente escuchando todos los mensajes PDP transmitidos por la red, tanto de peticiones como de respuesta, para actualizar sus cachés con los servicios incluidos en ellos. La caché tiene un

```
search(service_type) {
        foreach (s ∈ Local)
          if (s.type==service_type) OR (service_type==ALL) known_services+=s;
        foreach (e ∈ Cache)
          if (e.type==service_type) OR (service_type==ALL) known_services+= e;
        broadcast(PDP_Service_Request(service_type, 1/n, known_services));
        set_timer(CONFIG_WAIT_RPLY,EXPIRED);
        loop (forever)
          service_list+= hear_network(PDP_Service_Reply);
    EXPIRED:
        return(service_list + known_services);
}
```

Figura 4.2: PDP_UA – implementación en pseudocódigo de "una petición–varias respuestas".

tamaño limitado, por lo que cuando un PDP_UA escucha el anuncio de un nuevo servicio y su caché está llena, debe eliminar la entrada que tiene un tiempo de expiración asociado menor y almacenar la nueva que ha escuchado.

Agente de Servicio PDP

Un PDP_SA es el encargado de anunciar los servicios disponibles en un dispositivo. Debe procesar mensajes `PDP_Service_Request` y generar el correspondiente `PDP_Service_Reply` cuando proceda. Un PDP_SA para responder a una petición de servicio, consulta la lista de servicios locales y los servicios almacenados en la caché asociada al mismo interfaz de red por el que ha recibido el correspondiente `PDP_Service_Request`.

Para minimizar el número de transmisiones, el PDP_SA tiene en cuenta el tipo de búsqueda que le ha solicitado el PDP_UA remoto. Cuando un PDP_SA recibe un `PDP_Service_Request 1/1` (ver Figura 4.3), comprueba si el servicio solicitado es uno de sus servicios locales. Si es así, genera un `PDP_Service_Reply` y retarda su envío un tiempo aleatorio inversamente proporcional al tiempo de disponibilidad del dispositivo. Durante este tiempo, si escucha que otro dispositivo responde a esa misma búsqueda, descarta su mensaje de respuesta, porque la aplicación remota sólo va a tener en cuenta la primera respuesta recibida. Si el temporizador expira sin que se hayan producido respuestas en la red, el PDP_SA envía su `PDP_Service_Reply`. De esta forma, el algoritmo permite que respondan primero los dispositivos que van a permanecer más tiempo en la red.

Cuando un PDP_SA recibe un `PDP_Service_Request 1/n` (ver Figura 4.4), comprueba si el servicio solicitado es uno de sus servicios locales o está almacenado

```
receive(PDP_Service_Request(service_type, 1/1)) {
        foreach (s ∈ Local)
          if (s.type==service_type) new_service_list+= s;
        if (new_service_list != NULL) {
          set_timer(generate_random_time(1/T),EXPIRED);
          loop (forever) {
            hear_network(PDP_Service_Reply);
            exit;
          }
    EXPIRED:
        broadcast(PDP_Service_Reply(new_service_list));
        }
}
```

Figura 4.3: PDP_SA – implementación en pseudocódigo de "una petición–una respuesta".

en su caché. Si es así, genera un tiempo aleatorio inversamente proporcional al tiempo de disponibilidad del dispositivo y al número de servicios que conoce del tipo solicitado (locales y almacenados en la caché). Durante este tiempo, el PDP_SA escucha la red para procesar los mensajes `PDP_Service_Reply` de la misma búsqueda. Cuando finaliza el temporizador, comprueba si conoce nuevos servicios que no han sido anunciados todavía en la red, si es así construye y envía su `PDP_Service_Reply`. De esta forma, el algoritmo permite que responda primero el que lleva más tiempo en la red y el que conoce más servicios, que será el que tiene una visión más acertada del entorno en el que se encuentra.

En ciertos casos, es posible detectar cuando un dispositivo va a cambiar de red o apagarse. Si es así, el PDP_SA del dispositivo tiene que borrar todas las entradas de la caché del dispositivo y enviar un `PDP_Service_Deregister`, listando todos sus servicios locales, de tal forma que los PDP_UA, que escuchen estos mensajes, deben eliminar de sus cachés estos servicios.

Cuando una aplicación intenta acceder a un servicio obtenido en una búsqueda y comprueba que no está disponible, debe borrarlo de la caché del dispositivo y comunicárselo a su PDP_SA, para que envíe un mensaje `PDP_Service_Deregister` informando al resto de los dispositivos que ha detectado que ese servicio no está disponible. Los PDP_UAs que reciben este mensaje deben borrarlo de sus cachés.

```
receive(PDP_Service_Request(service_type, 1/n, known_services)) {
        foreach (s ∈ Local)
          if (s.type==service_type) OR (service_type==ALL) new_service_list+= s;
        foreach (e ∈ Cache)
          if (e.type==service_type) OR (service_type==ALL) new_service_list+= e;
        new_service_list = new_service_list - (new_service_list ∩ known_services);
        if (new_service_list != NULL) {
          set_timer(generate_random_time(1/(T*new_services_list.length)),EXPIRED);
          loop (forever)
            remote_service_list+= hear_network(PDP_Service_Reply);
    EXPIRED:
          if not (new_service_list ⊆ remote_service_list) {
            new_service_list= new_service_list - (remote_service_list ∩ new_service_list);
            broadcast(PDP_Service_Reply(new_service_list));
          }
        }
}
```

Figura 4.4: PDP_SA – implementación en pseudocódigo de "una petición–varias respuestas".

4.4. Diseño razonado

Una vez descrito el funcionamiento del protocolo propuesto, en este apartado realizamos una explicación detallada de cada una de sus características, justificando cada una de las decisiones tomadas mediante análisis teórico, cuando la complejidad del problema así lo permite, y a través de simulaciones en caso contrario.

Las simulaciones se han realizado empleando programas específicos para este propósito, codificados empleando Modsim, un lenguaje de simulación modular y orientado a objetos. Seguimos el paradigma de simulación de eventos discretos orientado a proceso. Para calcular el final de las simulaciones se ha empleado el método de medias por bloques (batch-mean) con un nivel de confianza del 90 % y un intervalo de confianza del 10 %, ver anexo A.

El objetivo de las simulaciones es estudiar el comportamiento del protocolo de descubrimiento de servicios en sí, por lo que las capas inferiores han sido simplificadas, así que tanto los detalles de la capa física como la capa MAC han sido eliminados.

Durante la simulación, los dispositivos se unen a la red de manera aleatoria, solicitan y ofrecen servicios de forma aleatoria y abandonan la red después de un tiempo aleatorio. El número de dispositivos que existen en la red, varía con el tiempo, pero su media permanece estacionaria. Los tiempos aleatorios siguen

una distribución exponencial, mientras que la asignación de servicios ofrecidos a dispositivos sigue una distribución uniforme. Por simplificación, consideramos que cada dispositivo ofrece un único servicio.

Los parámetros de las simulaciones realizadas son los siguientes:

- Número medio de dispositivos.
- Número de tipos de servicios.
- Para cada dispositivo:
  - Tiempo medio que permanece el dispositivo en la red.
  - Tiempo medio entre búsquedas de servicios.
  - Tamaño de la caché.

Como resultado de las simulaciones hemos seleccionado los siguientes variables:

- **Número de mensajes por búsqueda**: el número de mensajes transmitidos en la red, normalizado al número de búsquedas de servicios.
- **Tasa de descubrimiento de servicios**: el porcentaje del número de servicios descubiertos, respecto al número total de servicios disponibles en la red.
- **Tasa de descubrimientos falsos**: el porcentaje del número de servicios descubiertos que en realidad no están disponibles en la red, respecto al número total de servicios descubiertos.

El escenario de simulación base del que partimos en las simulaciones realizadas en este apartado es el siguiente:

- Número medio de dispositivos: 20.
- Número de tipos de servicios: 5.
- Para cada dispositivo:
  - Tiempo medio que permanece el dispositivo en la red: 600 segundos.
  - Tiempo medio entre búsquedas de servicios: 60 segundos.

- Tamaño de la caché: 100.

El tiempo de disponibilidad con el que se anuncian los dispositivos está ajustado al tiempo medio que permanecen los dispositivos en la red, en este caso 600 segundos.

Las simulaciones realizadas muestran como varían los resultados de simulación cuando se modifica alguno de sus parámetros, en cada caso indicaremos detalladamente el rango de valores para cada uno de ellos. Siempre que no se indique algún parámetro de simulación, se debe considerar el valor indicado en el escenario base.

Cada uno de los siguientes apartados tiene como título una característica del protocolo diseñado y en él se explica de forma detallada el porqué de cada una de ellas.

4.4.1. No emplea directorio de servicios

Las redes que se forman en entornos ubicuos son dinámicas y heterogéneas. Dinámicas porque los dispositivos entran y salen de ellas de forma espontánea, y heterogéneas porque los dispositivos que la forman tienen características muy distintas: desde dispositivos móviles con capacidad limitada de proceso y comunicación y que funcionan a través de baterías, hasta equipos fijos sin limitaciones.

En este tipo de entornos, podemos encontrarnos con redes en las que todos los dispositivos son móviles y limitados, y por lo tanto, no es posible apoyar el funcionamiento de un protocolo de descubrimiento de servicios en un dispositivo central que ejerza de servicio de directorio, pues:

- Los dispositivos son limitados, lo que dificulta que uno de ellos actúe como servicio de directorio.
- Los dispositivos son móviles, por lo que si uno de ellos toma el papel de servicio de directorio y abandona la red, deben existir mecanismos de selección automática del nuevo directorio, que se ejecutarán muy frecuentemente, con el correspondiente gasto en ancho de banda y con el problema añadido de que hasta que no se selecciona un nuevo directorio no está disponible el mecanismo de descubrimiento.

En escenarios en los que algunos de los dispositivos que forman la red son dispositivos no limitados y no móviles, parece viable plantearse el uso de directorios. Como los dispositivos no tendrán la dirección del directorio configurada a

priori, es necesario emplear mecanismos dinámicos de descubrimiento de directorios, explicados en la sección 3.3.2. Si existen dispositivos muy móviles, el tráfico que generan estas búsquedas de directorio provoca un gran consumo de ancho de banda.

Como justificación a este último punto, hemos realizado una simulación de un directorio teórico, es decir, una red de dispositivos en la que existe uno que ejerce de directorio de servicios, y al que los dispositivos que entran en la red deben descubrir. En la Figura 4.5, vemos como el número de mensajes transmitidos aumenta cuando el tiempo de vida de los dispositivos disminuye, debido al tráfico generado por los mensajes de búsqueda de directorio. Para esta simulación hemos empleado el escenario base, modificando la media del tiempo de vida de los dispositivos entre 37,5 y 19200 segundos.

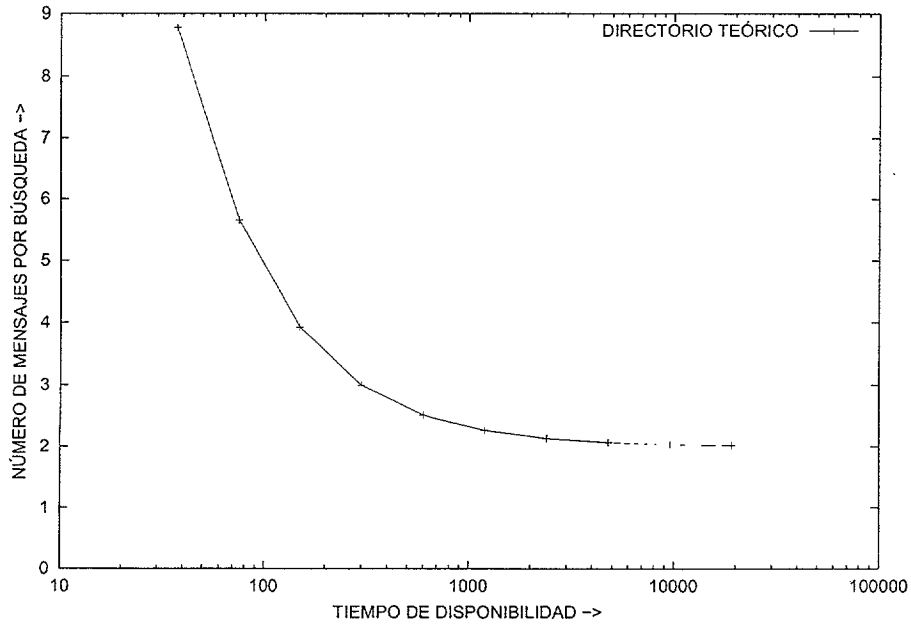


Figura 4.5: Número de mensajes respecto al tiempo de disponibilidad en un directorio teórico.

Este análisis nos lleva a decir que el diseño de un nuevo protocolo de descubrimiento de servicios en entornos de computación ubicua no debe depender de un servicio de directorio, y por ello el protocolo PDP diseñado tiene un funcionamiento totalmente distribuido.

4.4.2. Mezcla características de los métodos *pull* y *push*

Partiendo de que no tenemos un servicio de directorio, las peticiones y/o los anuncios de servicios deben solicitarse por multicast/broadcast siguiendo alguno de los modos descritos en la sección 3.3: *push* o *pull*.

El método *pull* es el menos costoso computacionalmente tanto para los que demandan servicios como para los que los ofrecen, ya que el servicio se descubre cuando se necesita, por lo que los clientes no tienen ningún tipo de memoria asociada. El método *push* introduce mayor complejidad en ambos sistemas, porque los servidores deben anunciar de forma periódica los servicios que ofrecen y los clientes deben almacenarlos localmente para utilizar esta información cuando precisen acceder a ellos.

En un análisis teórico de las prestaciones de ambos modos, podemos comprobar que el método *pull* proporciona tasas de descubrimiento de servicios del 100 % y tasas de descubrimiento de servicios falsos del 0 %, porque cada vez que se necesita un servicio se pregunta y todos los servidores disponibles responden a la petición y por lo tanto, no se descubre ningún servicio falso. En cuanto al número de mensajes transmitidos, dependerá si el dispositivo que realiza la búsqueda ofrece o no el servicio, así podemos decir que si el número de servicios distintos es k, la media de dispositivos en la red es n, y cada dispositivo ofrece un único servicio que se le asigna de forma uniforme entre los k posibles, habrá en la red una media de n/k dispositivos que ofrecen el mismo tipo de servicio, entonces:

- Un dispositivo ofrece un tipo concreto de servicio con una probabilidad p igual a:

$$\mathrm{p} = \frac{1}{k} \tag{4.1}$$

  y por lo tanto, el número medio de servidores que ofrecen el servicio en la red y que responderán a la petición de búsqueda son:

$$\frac{n}{k} - 1 \tag{4.2}$$

  entonces, el número de mensajes transmitidos por búsqueda ponderado por la probabilidad es igual a:

$$\left(\frac{1}{k}\right)\left(\frac{n}{k}\right) \tag{4.3}$$

- Un dispositivo no ofrece un tipo concreto de servicio con una probabilidad q igual a:

$$\mathrm{q} = 1 - \mathrm{p} = \frac{(k-1)}{k} \tag{4.4}$$

y por lo tanto, el número medio de servidores que ofrecen el servicio en la red y que responderán a la petición de búsqueda es:

$$\frac{n}{k} \tag{4.5}$$

entonces, el número de mensajes transmitidos por búsqueda ponderado por la probabilidad es igual a:

$$\frac{(k-1)}{k}\left(\frac{n}{k}+1\right) \tag{4.6}$$

Por lo tanto, el número de mensajes por búsqueda en un modo *pull* es igual a la suma de la ecuación 4.3 y la ecuación 4.6:

$$\mathrm{NumeroMensajes} = \frac{k+n-1}{k} \tag{4.7}$$

En la Figura 4.6, podemos ver una gráfica en la que se observa cómo aumenta el número de mensajes por búsqueda a medida que aumenta el número de dispositivos, dependiendo del número de tipos de servicios distintos que existen en la red.

En cuanto a las prestaciones del método *push*, el análisis es más complejo. Si consideramos que el tiempo entre anuncios de los servicios ofrecidos por los dispositivos es T_a y que los dispositivos realizan una petición según una distribución exponencial de media ν, el número de mensajes por búsqueda será:

$$\mathrm{NumeroMensajes} = \frac{\nu}{T_a} \tag{4.8}$$

Por ejemplo, si consideramos que la media entre mensajes de búsqueda es $\nu = 60$ segundos, el valor del número de mensajes para un tiempo entre anuncios $T_a = 12$ segundos es 5, para $T_a = 60$ segundos es 1 y para $T_a = 300$ segundos es 0,2. Por lo tanto, el número de mensajes por búsqueda aumenta cuanto menor es el tiempo entre anuncios, T_a.

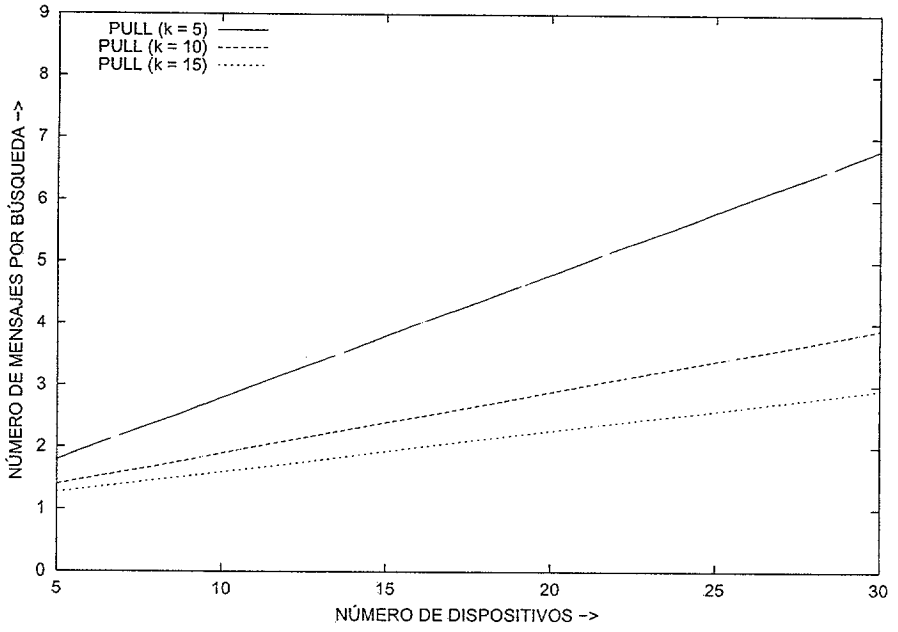


Figura 4.6: Número de mensajes respecto al número de dispositivos en modo *pull*.

Para obtener el valor de la tasa de descubrimiento de servicios, suponemos que el tiempo de vida de los dispositivos sigue una distribución exponencial de media μ. Por lo tanto, la probabilidad p de que el servicio ofrecido por un dispositivo A haya sido descubierto por otro dispositivo B es:

- Si B lleva en la red más de $t > T_a$ segundos, entonces $p = 1$ puesto que al menos ha tenido que escuchar un anuncio de A.
- Si B lleva un tiempo $t < T_a$, entonces $p = \frac{t}{T_a}$

Por lo tanto, teniendo en cuenta que la distribución del tiempo de vida de B es exponencial, la probabilidad p de que un servicio sea descubierto es:

$$p = \int_0^{T_a} \frac{1}{\mu} e^{\frac{-t}{\mu}} \frac{t}{T_a} dt + \int_{T_a}^{\infty} \frac{1}{\mu} e^{\frac{-t}{\mu}} dt \tag{4.9}$$

$$\mathrm{p} = \frac{\mu}{T_a}(1 - e^{\frac{-T_a}{\mu}}) \tag{4.10}$$

En la Figura 4.7, representamos el valor de la tasa de descubrimiento de servicios (TasaServiciosDescubiertos $= 100 * p$), considerando que el tiempo medio de vida de los dispositivos es $\mu = 600$ segundos, para varios valores de T_a. Como

podemos observar, cuanto menor es el tiempo de vida de los dispositivos, la tasa de descubrimiento es menor y en estos casos, este valor mejora cuanto menor es T_a. Podemos comprobar que para garantizar tasas de descubrimiento del 95 %, es necesario que el tiempo entre anuncios, T_a, sea 10 veces menor que el tiempo medio de vida de los dispositivos, μ. Por ejemplo, si queremos que dispositivos que tengan un tiempo medio de vida, $\mu = 10$ minutos, descubran el 95 % de servicios disponibles en la red, es necesario que todos los dispositivos anuncien sus servicios cada minuto, con el gran consumo de ancho de banda que esto supone.

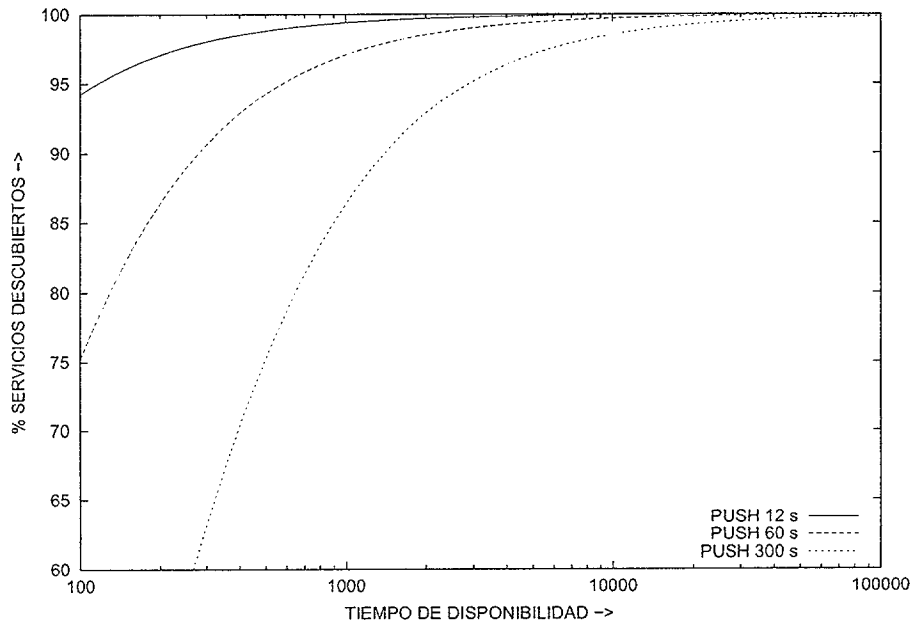


Figura 4.7: Tasa de servicios descubiertos respecto al tiempo de disponibilidad en modo *push*.

Calculamos de igual modo la probabilidad f de que un servicio descubierto sea falso, que es lo mismo que calcular que la probabilidad de que un servicio almacenado en la caché sea falso. Realizamos una deducción similar a la anterior, consideramos cuál es la probabilidad de que en un tiempo t exista una entrada falsa en la caché:

- Si $t < T_a$, la probabilidad que en un tiempo t exista una entrada en la caché falsa es $f = \frac{1}{T_a}$
- Si $t > T_a$, la probabilidad es $f = 0$ porque el tiempo de vida asociado a las entradas de la caché es el mismo que el tiempo entre anuncios.

Por lo tanto, si el tiempo de vida del dispositivo que ha realizado el anuncio es exponencial, la probabilidad f de que un servicio almacenado en la caché sea falso es:

$$f = \int_0^{T_a} \frac{1}{T_a} e^{\frac{-t}{\mu}} \, dt \tag{4.11}$$

$$f = 1 - \frac{\mu}{T_a}(1 - e^{\frac{T_a}{\mu}}) \tag{4.12}$$

Que comparándolo con la ecuación 4.10 se corresponde con:

$$f = 1 - p \tag{4.13}$$

En la Figura 4.8, representamos el valor de la tasa de descubrimiento de servicios falsos (TasaDescubrimientosFalsos $= 100 * f$) en el mismo escenario que para la figura anterior. Como se puede observar, cuanto menor es el tiempo de vida de los dispositivos mayor debe ser el tiempo entre anuncios para que el ratio de error se aproxime a 0.

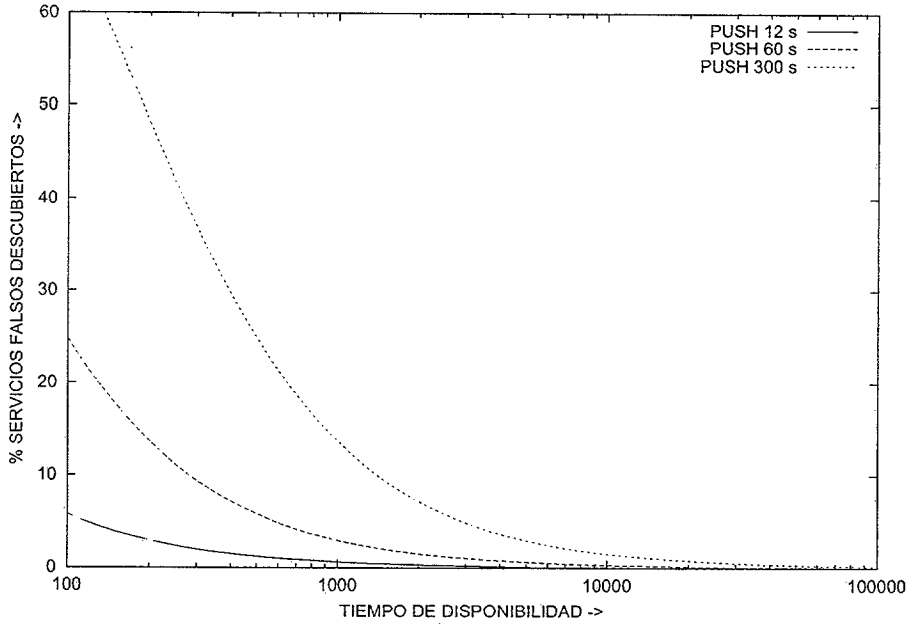


Figura 4.8: Tasa de servicios falsos respecto al tiempo de disponibilidad en modo *push*.

De los análisis anteriores, concluimos que el modo *pull* presenta como principal inconveniente la gran cantidad de mensajes que se transmiten por petición, y

como ventajas que su tasa de descubrimiento de servicios es muy alta y además no se transmite ningún mensaje en la red si nadie realiza búsquedas de servicios. En cuanto al modo *push*, el porcentaje de mensajes transmitidos disminuye, pero si queremos alcanzar tasas de descubrimiento de servicios altas, los dispositivos deben anunciar sus servicios con una periodicidad muy alta, con el consiguiente consumo de ancho de banda. Además, estos anuncios se deben realizar periódicamente aunque nadie esté realizando búsquedas.

Uno de los objetivos de PDP es disminuir el número de mensajes transmitidos y por lo tanto, el coste en batería que esto supone a los dispositivos. El diseño de PDP surge con la idea de mezclar las características de ambos modos, beneficiándose de las ventajas de cada uno de ellos, de tal forma que sólo se transmite una petición de servicio a la red cuando una aplicación de usuario lo solicita (modo *pull*) y la respuesta a esta petición se transmite por difusión para que todos los dispositivos puedan almacenar este servicio en una caché local (modo *push*). Es decir, los servicios se anuncian con la periodicidad marcada por las peticiones de ese servicio realizadas por los dispositivos que forma la red.

En los siguientes apartados analizamos estas características de PDP, desde el punto de vista de modificaciones al modo *pull*, primero vemos qué prestaciones tiene un modo *pull* con caché y después qué efecto tiene que las respuestas a las búsquedas se realicen por difusión.

Caché de servicios en el modo *pull*

El uso de cachés locales, en la que los dispositivos almacenan lo servicios que aprenden al escuchar los mensajes que se transmiten en la red, permite minimizar el número de transmisiones realizadas por los dispositivos, ya que si el dispositivo encuentra el servicio en la caché no transmite el correspondiente mensaje de búsqueda.

El precio a pagar en este caso, es que se introduce cierta incertidumbre sobre la disponibilidad del servicio que se le proporciona a la aplicación del usuario, porque un servicio puede permanecer almacenado en la caché aunque el dispositivo que lo ofrece ya no está disponible. Este problema se agrava en la medida en que el entorno sea más dinámico.

En las Figuras 4.7 y 4.8, ya hemos visto como influye este problema a las prestaciones del método *push*. A continuación, analizamos mediante simulación qué es lo que ocurre si a un método *pull* le introducimos una caché, en la que los servicios permanecen almacenados el tiempo medio de vida del dispositivo que lo ofrece.

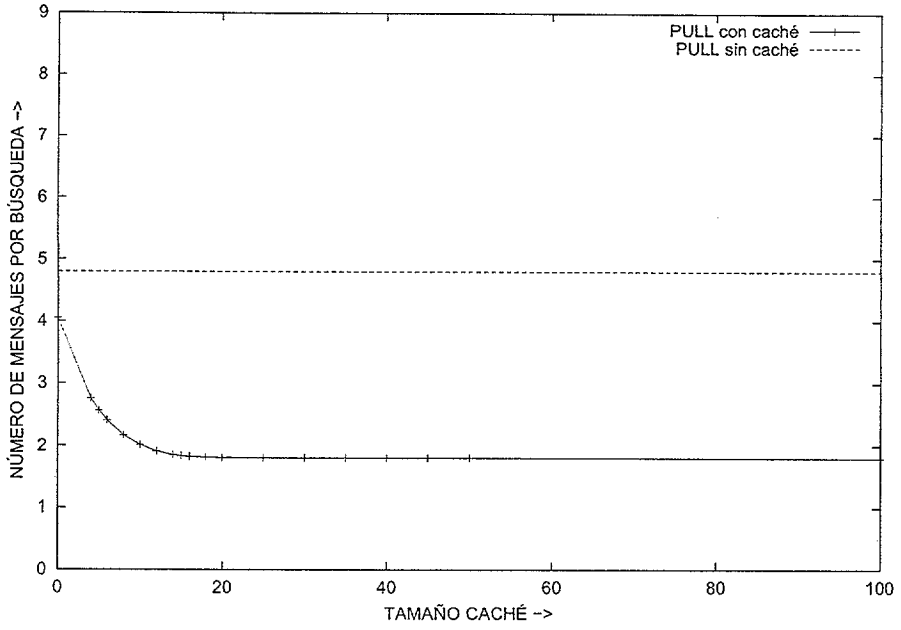


Figura 4.9: Número de mensajes respecto al tamaño caché en modo *pull* con caché.

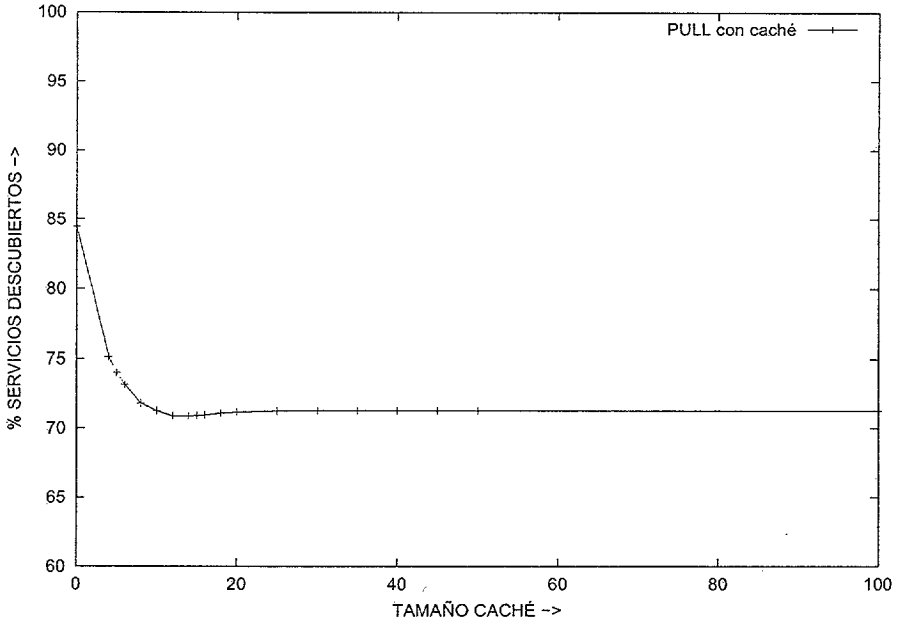


Figura 4.10: Tasa de descubrimiento de servicios respecto al tamaño caché en modo *pull* con caché.

En la Figura 4.9, vemos cómo se reduce de forma significativa el número de transmisiones que se producen. El escenario simulado se corresponde con el escenario base en el que se ha variado el tamaño de la caché de los dispositivos entre 0 y 100. En las Figuras 4.10 y 4.11 vemos que la tasa de descubrimiento de servicios y de servicios falsos se han degradado considerablemente al introducir la caché, disminuyendo el número de servicios descubiertos y descubriendo un mayor porcentaje de servicios falsos. Recordemos que en este escenario, las prestaciones de un modo *pull* sin caché son: número de mensajes por búsqueda 4,8, tasa de descubrimiento de servicios 100 % y tasa de servicios falsos 0 %.

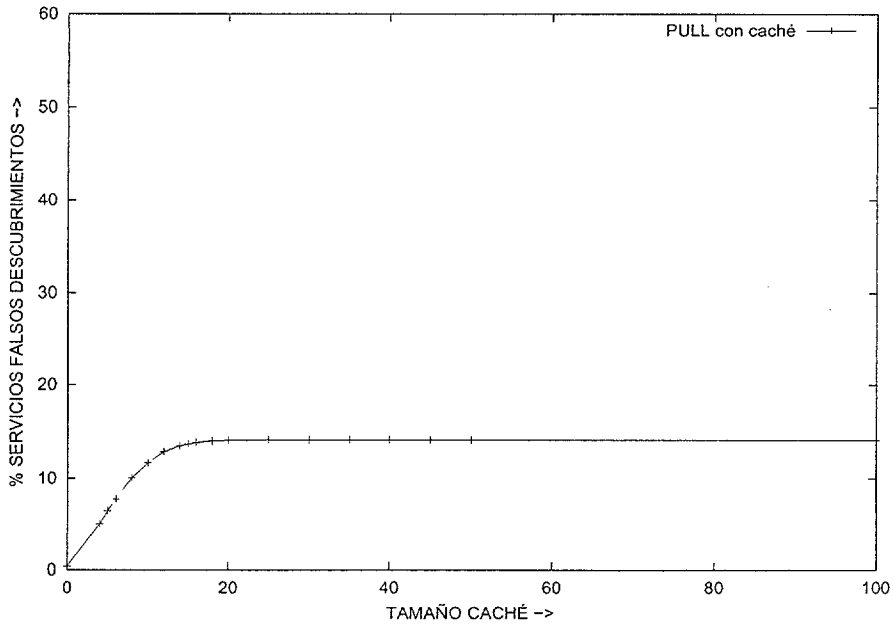


Figura 4.11: Tasa de servicios falsos respecto al tamaño caché en modo *pull* con caché.

Si nos fijamos en la figura 4.9, el valor obtenido para tamaño de caché 0 es menor que el calculado para el modo *pull* sin caché, que es 4,8. Esto es debido a que en el modo *pull* con caché si el dispositivo ofrece el servicio de forma local tampoco se envía un mensaje de petición, con lo que el número de mensajes por búsqueda se reduce en este caso al valor de la ecuación 4.6, es decir:

$$\text{NumeroMensajes} = \frac{(k-1)}{k}\left(\frac{n}{k}+1\right) \tag{4.14}$$

que si evaluamos con los datos del escenario que estamos considerando es 4.

Como hemos visto, la introducción de una caché en el modo *pull* nos permite

disminuir el número de mensajes transmitidos, pero es necesario introducir mecanismos de consistencia de caché para mejorar las prestaciones del protocolo, eliminando el número de servicios falsos en la caché que provocan que la tasa de descubrimiento de servicios se reduzca y aumente la de servicios falsos.

Mensajes de respuesta por difusión en el modo *pull*

Si en un modo *pull* los mensajes de respuesta se transmiten por difusión, estos mensajes se convierten en anuncios de servicios para aquellos dispositivos que no realizaron activamente la búsqueda y por lo tanto, se obtiene un mecanismo semejante al modo *push*. La principal ventaja del modo *push* es la reducción del número de mensajes, ya que las cachés de todos los dispositivos, gracias a estos anuncios gratuitos, están más actualizadas que en el caso de que las respuestas se transmitan sólo al dispositivo que solicitó la búsqueda.

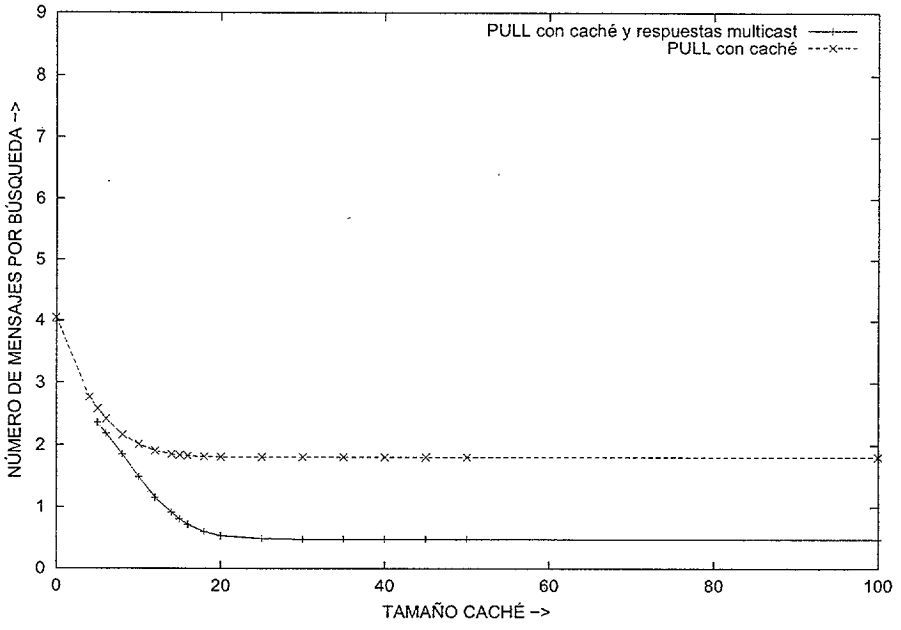


Figura 4.12: Número de mensajes respecto al tamaño caché en modo *pull* con caché y respuestas multicast.

En las Figuras 4.12, 4.13 y 4.14, vemos cuál es el efecto de que los mensajes de respuesta se transmitan por difusión en un modo *pull* con caché. El número de mensajes se reduce considerablemente y a partir de tamaños de caché 20 es inferior a 0,5, esto es debido a que las cachés están más actualizadas y un mayor número de búsquedas se satisfacen con la información almacenada en la caché, y

por lo tanto no es necesario realizar búsquedas en la red. Por este mismo motivo, la tasa de descubrimiento de servicios es mayor, se aproxima al 85 % y la tasa de servicios falsos también aumenta, se aproxima al 30 %.

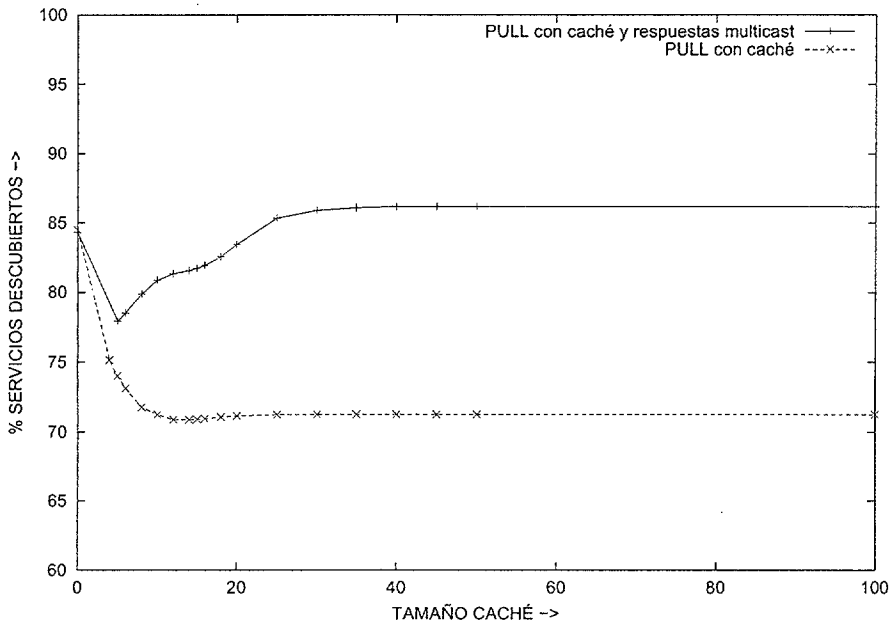


Figura 4.13: Tasa de descubrimiento de servicios respecto al tamaño caché en modo *pull* con caché y respuestas multicast.

4.4.3. Se incluyen servicios ya conocidos en los mensajes de búsqueda

Como hemos visto, introducir en el modo *pull* cachés de servicios y hacer que las respuestas se envíen por difusión permite disminuir el número de mensajes y tener una tasa de descubrimiento de servicios razonable (85 %), aunque con una tasa no despreciable de servicios falsos. Por lo tanto, debemos centrarnos en conseguir mayores tasas de descubrimiento de servicios y disminuir la tasa de servicios falsos, manteniendo un valor reducido en el número de mensajes por búsqueda.

A partir de esta nueva modificación introducida en el modo *pull*, denominamos al protocolo diseñado como Pervasive Discovery Protocol, PDP.

Para mejorar la tasa de servicios descubiertos, en PDP cuando se solicita la búsqueda de un servicio, aunque se encuentre en la caché, se transmite el mensaje de búsqueda pero incluyendo la lista de servicios conocidos, es decir, una lista con

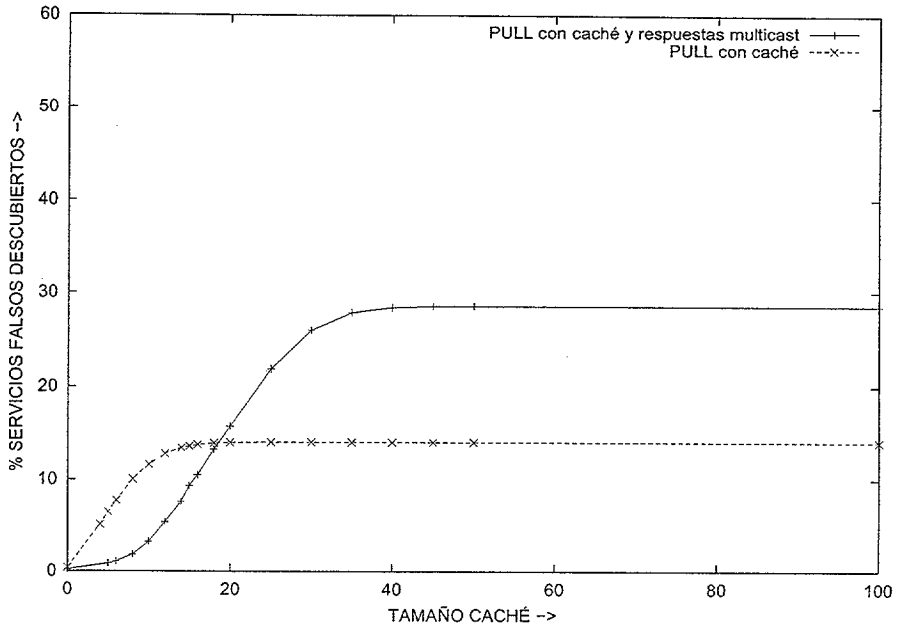


Figura 4.14: Tasa de servicios falsos respecto al tamaño caché en modo *pull* con caché y respuestas multicast.

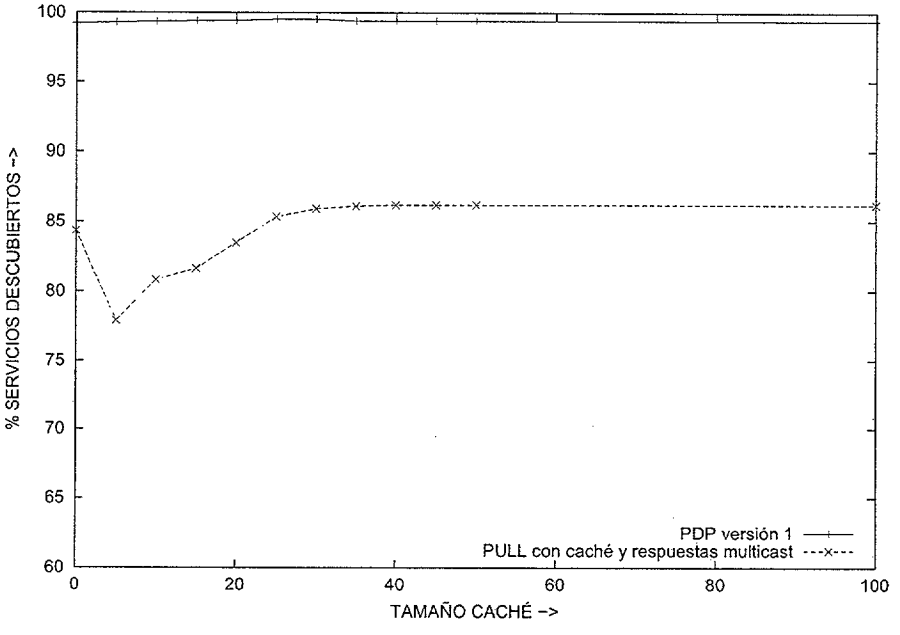


Figura 4.15: Tasa de descubrimiento de servicios respecto al tamaño caché en PDP versión 1.

los servicios del tipo solicitado que están almacenados en la caché. De esta forma, los servidores que ofrecen el servicio y que deberían responder a esta petición, si ya se ven incluidos en la lista de servicios conocidos no transmiten su mensaje de respuesta.

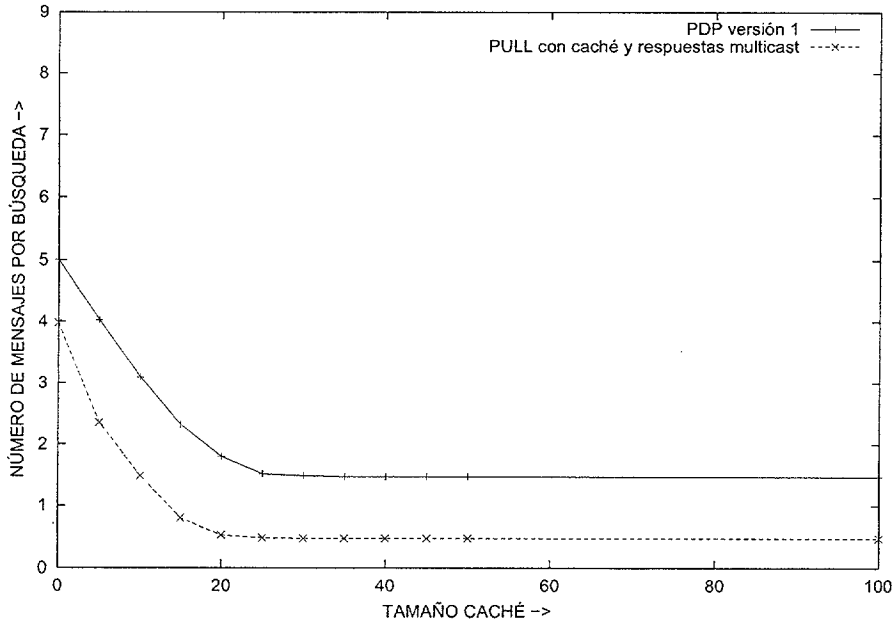


Figura 4.16: Tasa de descubrimiento de servicios respecto al tamaño caché en PDP versión 1.

En la Figura 4.15, se observa que con esta modificación hemos conseguido aumentar la tasa de servicios descubiertos hasta casi un 100 % (en las simulaciones para contabilizar el número de servicios descubiertos se espera un tiempo determinado para recibir todas las respuestas, en el transcurso de este tiempo pueden aparecer en la red nuevos dispositivos que ofrecen el servicio y que no han escuchado el mensaje de búsqueda y por ello, nunca se alcanza el valor del 100 %). El coste a pagar es, como se observa en la Figura 4.16, el aumento del número de mensajes, ya que aunque encontremos el tipo de servicio en la caché, transmitimos siempre un mensaje de búsqueda en la red para contrastar si esta información es completa.

Respecto a la tasa de servicios descubiertos falsos, se observa en la Figura 4.17 que se mantiene en los mismos valores que en el caso anterior.

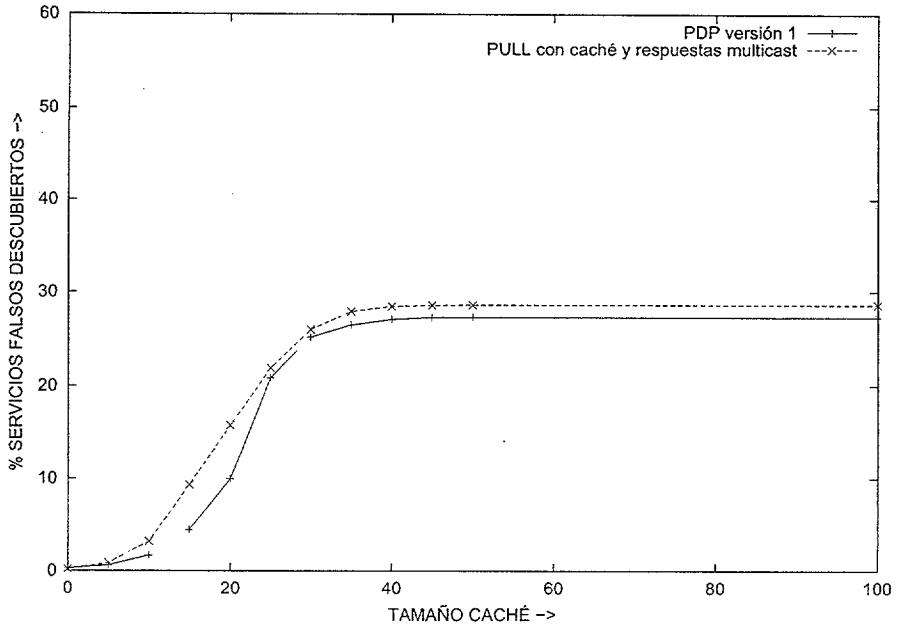


Figura 4.17: Tasa de servicios falsos respecto al tamaño caché en modo PDP versión 1.

4.4.4. Se responde con los servicios almacenados en la caché

En entornos de computación ubicua la cooperación entre los dispositivos es fundamental, gracias a ella los dispositivos pueden realizar tareas más complejas a un coste menor. Con esta idea en mente hemos introducido nuevos mecanismos en PDP que mejoran las prestaciones del protocolo.

En PDP todos los dispositivos que conozcan un servicio pueden responder a los mensajes de búsqueda, es decir, no sólo responden aquellos dispositivos que ofrecen el servicio sino también aquellos que tienen almacenados servicios del tipo solicitado en su caché.

Además, a medida que se transmiten las respuestas, los dispositivos comprueban si conocen algún servicio que no se haya anunciado todavía, y si es así transmiten una respuesta con esos nuevos servicios. De esta forma, entre todos los dispositivos de la red se construye una respuesta que incluirá todos los servicios disponibles minimizando el número de mensajes transmitidos (ver Figura 4.18).

Este nuevo mecanismo se mejora, en cuanto a número de mensajes, si responde antes el que más servicios conoce. Como las respuestas se envían por difusión, para que no se produzcan colisiones es necesario generar un retardo aleatorio, en

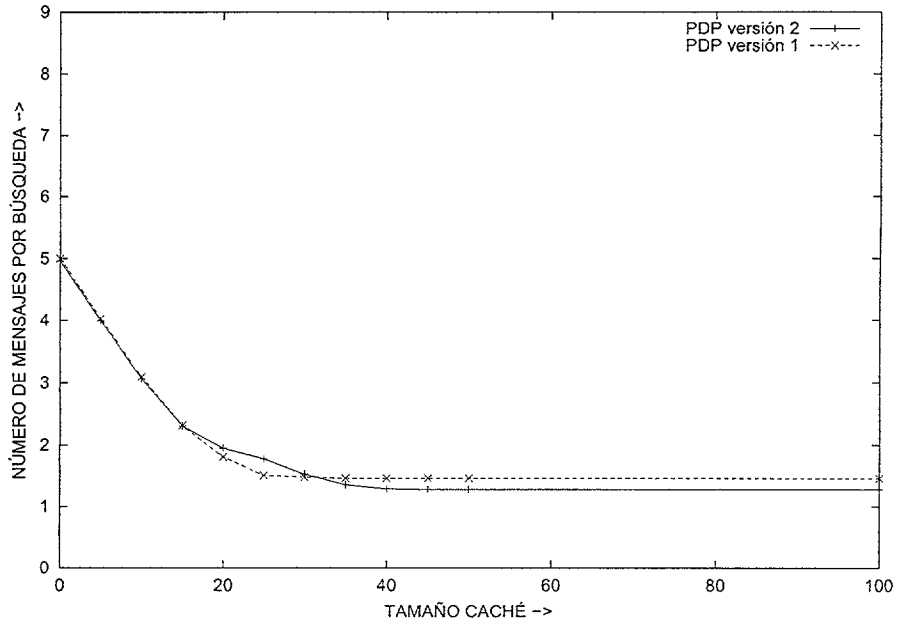


Figura 4.18: Número de mensajes respecto al tamaño caché en PDP versión 2.

la mayoría de los protocolos este retardo sigue una distribución uniforme, pero en nuestro caso este retardo es inversamente proporcional al tiempo de disponibilidad y al número de servicios conocidos por el dispositivo. De esta forma, responde primero el que más servicios conoce y el que lleva más tiempo en la red, por lo tanto, el que tiene una mejor visión del entorno en el que se encuentra.

Esta forma de generar los tiempos de espera permite además, que los dispositivos con un menor tiempo de disponibilidad y menores tamaños de caché, que se corresponderán con dispositivos limitados, respondan a menos solicitudes de búsqueda que los dispositivos con mayor tiempo de disponibilidad y mayor caché, que se corresponderán normalmente con dispositivos fijos y por lo tanto, con menos limitaciones. Gracias a ello, el consumo de baterías se reduce más en los dispositivos limitados, que son lo que tienen mayores restricciones en cuanto a consumo.

Con la introducción de esta mejora la tasa de servicios descubiertos se mantiene mientras que la de servicios falsos aumenta ligeramente, como se puede comprobar en la Figura 4.19. Esto se produce porque al responder con las entradas que existen en las cachés, se pueden propagar entradas falsas entre los dispositivos. Veremos cómo se corrige este efecto en el apartado 4.4.6.

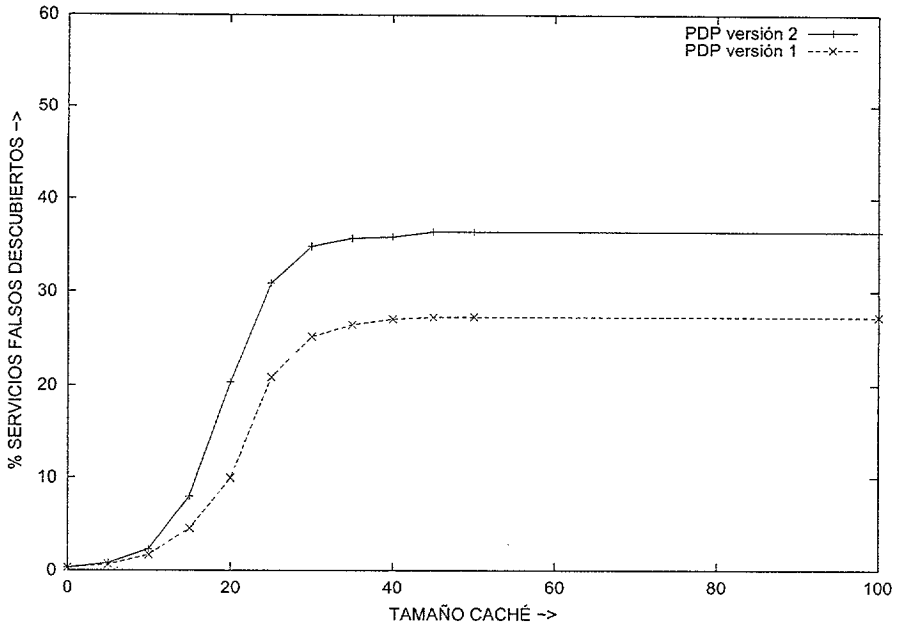


Figura 4.19: Tasa de servicios falsos respecto al tamaño caché en PDP versión 2.

4.4.5. Se actualiza la caché con los servicios incluidos en los mensajes de búsqueda

En PDP hemos permitido, siguiendo la filosofía de explotar la cooperación entre los dispositivos, que las cachés no se actualicen solamente a través de los mensajes de respuesta sino también con los servicios que se incluyen en los propios mensajes de búsqueda. De esta forma, como se observa en la Figura 4.20 se disminuye ligeramente el número de mensajes por búsqueda y además, para tamaños de caché menores de 30 se reduce la tasa de servicios falsos, Figura 4.21.

4.4.6. Se introducen mecanismos de consistencia de cachés

Como hemos visto en los apartados anteriores, los nuevos mecanismos que hemos introducido en PDP, permiten disminuir el número de mensajes transmitidos, proporcionando una tasa de servicios descubiertos alta. El punto débil del protocolo es la tasa de servicios falsos, debido principalmente al número de entradas falsas existentes en las cachés y que se propagan a los dispositivos existentes en la red. Así, es fundamental introducir mecanismos de consistencia de cachés en el protocolo. Veremos en esta sección qué mecanismos hemos incluido en PDP

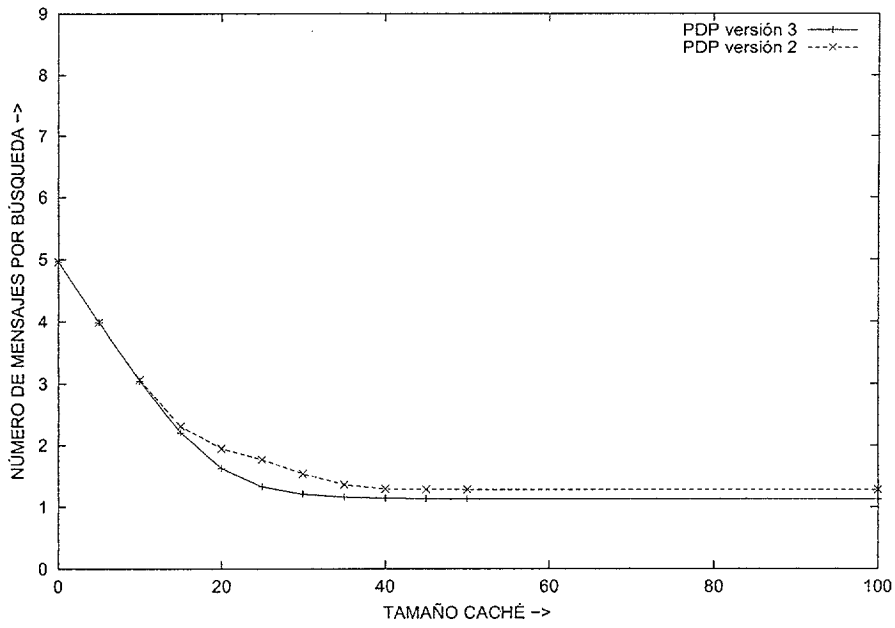


Figura 4.20: Número de mensajes respecto al tamaño caché en PDP versión 3.

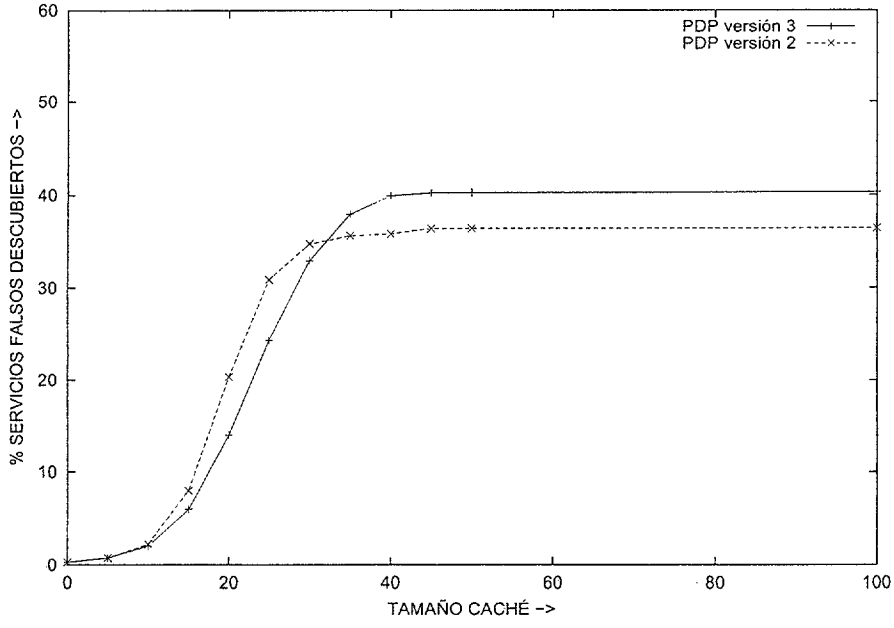


Figura 4.21: Tasa de servicios falsos respecto al tamaño caché en PDP versión 3.

y que nos han permitido disminuir la tasa de servicios falsos descubiertos.

El primer mecanismo es el cálculo del tiempo de expiración cuando se almacena un servicio en la caché, este tiempo se calcula como el mínimo entre el tiempo de disponibilidad local (del dispositivo que lo almacena), y el tiempo de disponibilidad del dispositivo que lo anuncia. De esta forma, servicios falsos almacenados en las cachés de dispositivos con tiempos de disponibilidad bajos (móviles) se propagarán con un tiempo de expiración bajo y por lo tanto, se borrarán rápidamente de las cachés de los dispositivos que los hayan almacenado. Estos dispositivos serán los que potencialmente introducen más entradas falsas en una red, ya que son los que se mueven con mayor facilidad entre distintas redes.

Otra de las técnicas que hemos introducido, que aparece también en otros protocolos de descubrimiento de servicios, consiste en enviar un mensaje `PDP_Service_Deregister` para que se borren de las cachés servicios que se prevé o se sabe que no van a estar disponibles en la red. Veremos a continuación cada uno de estos casos:

- **Detección de cambio de red o apagado del dispositivo**

  Cuando un dispositivo detecta que va a cambiar de red o que va a apagarse, debe realizar las siguientes tareas:

  1. Eliminar todas los servicios que tienen almacenados en su caché.
  2. Enviar un mensaje `PDP_Service_Deregister` listando todos sus servicios locales, de tal forma que todos los dispositivos que escuchan este mensaje deben eliminar estos servicios de sus cachés.

  Este mecanismo presenta el problema que no siempre se pueden detectar este tipo de eventos. Este técnica para la consistencia de cachés también se contempla en el protocolo SSDP.

- **Detección de errores en el acceso al servicio**

  Cuando una aplicación que ha solicitado la búsqueda de un servicio, intenta acceder a él y detecta que no está disponible debe informar a la capa de descubrimiento para que se realicen las siguientes acciones:

  1. Eliminar de su caché ese servicio.
  2. Enviar un mensaje `PDP_Service_Deregister` de ese servicio, de tal forma que todos los dispositivos que escuchan este mensaje deben eliminarlo de sus cachés.

  Este mecanismo aparece propuesto también en Multicast-DNS.

Hemos realizado varias simulaciones del protocolo PDP haciendo uso del mensaje opcional `PDP_Service_Deregister`, tanto para el caso en el que los dispositivos pueden detectar un cambio de red o que se va a apagar, como en el caso de fallo al acceder a un servicio. En ambos casos las prestaciones obtenidas son similares, en las Figura 4.22, 4.23 y 4.24 vemos como a partir de tamaños de caché 20 el número de mensajes por búsqueda es 1,26 y la tasa de descubrimiento de servicios es del 99,5 % mientras que la tasa de descubrimientos falsos cae hasta el 0,5 %. Se obtienen unas prestaciones similares al modo *pull* pero minimizando considerablemente el número de mensajes por búsqueda.

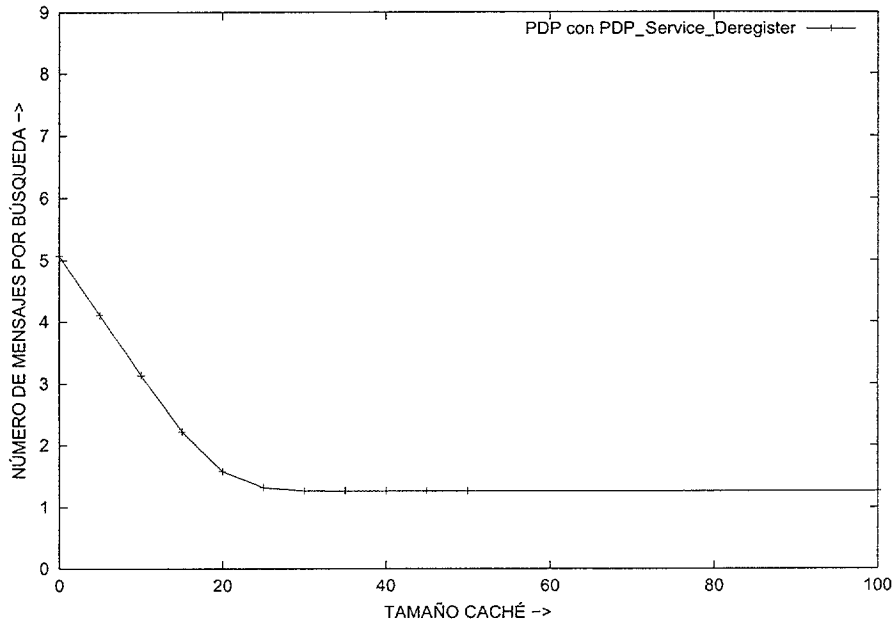


Figura 4.22: Número de mensajes respecto al tamaño caché en PDP con PDP_Service_Deregister.

4.4.7. Optimización para aplicaciones que realizan búsquedas "una petición–una respuesta"

Los protocolos de descubrimiento de servicios que han aparecido estos últimos años no diferencian entre aplicaciones del tipo "una petición-una respuesta", de las del tipo "una petición-varias respuestas", a la hora de realizar optimizaciones del protocolo. En PDP hemos diferenciado entre estos dos tipos de peticiones de búsqueda, con el objetivo de minimizar el ancho de banda consumido, la idea es que si sabemos que a la aplicación que solicita el servicio le da igual qué servidor

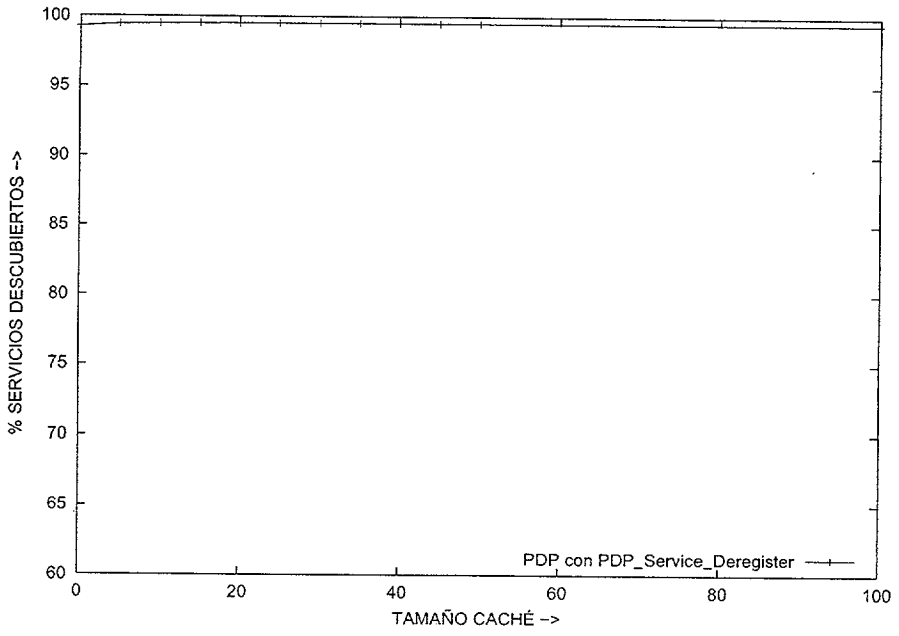


Figura 4.23: Tasa de descubrimiento de servicios respecto al tamaño caché en PDP con PDP_Service_Deregister.

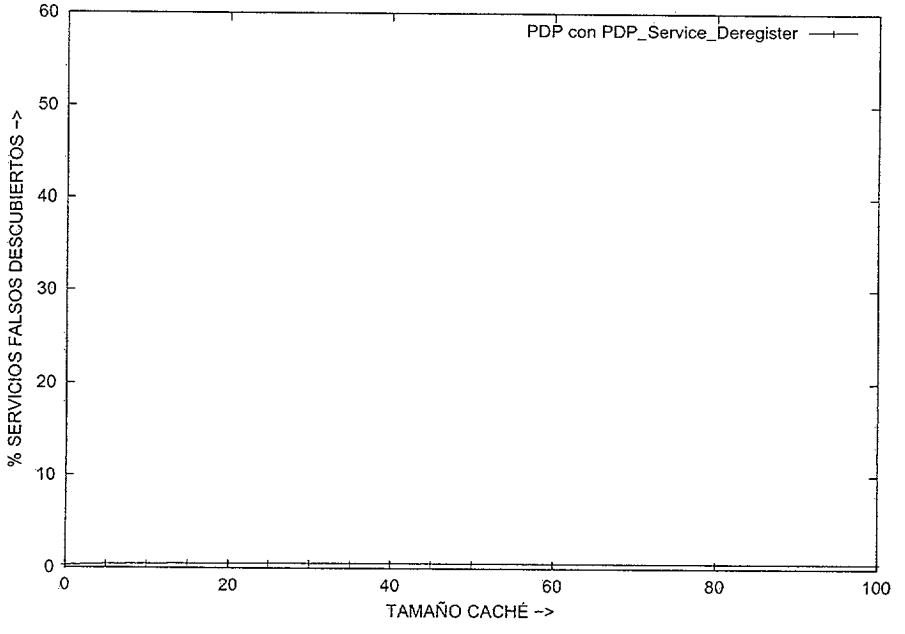


Figura 4.24: Tasa de servicios falsos respecto al tamaño caché en PDP con PDP_Service_Deregister.

se lo ofrezca, hagamos que sólo responda un servidor. Vemos en la Figura 4.25 como se disminuye el número de mensajes transmitidos en PDP realizando esta optimización, respecto al número de mensajes en un modo *pull*. Para esta simulación hemos empleado el escenario base, modificando la media del tiempo de vida de los dispositivos entre 37,5 y 19200 segundos.

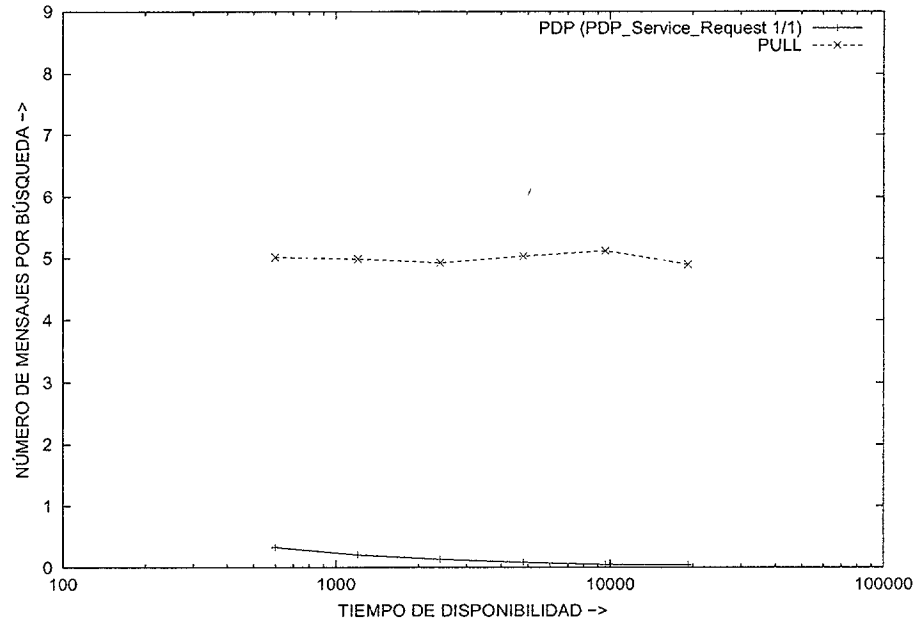


Figura 4.25: Número de mensajes respecto al tiempo de disponibilidad en PDP en búsquedas del tipo "una petición-una respuesta".

4.5. Consideraciones sobre el tiempo de disponibilidad de un dispositivo

Uno de los valores clave en PDP es el valor del tiempo de disponibilidad asociado a un dispositivo, éste se emplea para determinar el tiempo de expiración de una descripción de servicio almacenada en una caché, y para priorizar las respuestas a un mensaje de búsqueda.

En general, todos los protocolos de descubrimiento de servicios en los que es posible que los clientes almacenen temporalmente descripciones de servicios en cachés locales, emplean un tiempo de vida asociado a cada descripción que indica el tiempo máximo que se pueden almacenar. Repasamos el uso de este parámetro en los protocolos que hemos analizado en el capítulo 3:

- En SLP toda URL `Entry` tiene un tiempo de vida asociado que indica cuánto tiempo puede almacenarse esta URL `Entry` en las cachés de los UAs y en los DAs, en el caso centralizado. El valor máximo es de 18 horas, por restricción del tamaño de este campo en los paquetes SLP, y no se indica cómo debe establecerse este tiempo.

- En SSDP las entradas en la caché tiene un tiempo de expiración asociado, pero en el draft tampoco se dan indicaciones de cómo debe establecerse.

- En DNS-SD este tiempo de expiración se corresponde con el TTL empleado en DNS, en las propuestas DNS para redes sin infraestructura se realizan las siguientes consideraciones sobre el valor del TTL a emplear:

  - En Multicast-DNS se recomienda un valor de TTL igual a 120 minutos.
  - En LLMNR se recomienda emplear un TTL igual a cero en entornos muy dinámicos como redes ad-hoc y valores mayores en entornos más estables, con el objetivo de disminuir el tráfico.

Inicialmente en PDP este parámetro se considera que está configurado manualmente y dependiendo si el dispositivo es más móvil o menos se establecerá un valor menor o mayor, respectivamente. Como un trabajo futuro podría pensarse en que en determinados casos este valor cambie dinámicamente con el tiempo y se adapte a las características de movilidad detectadas en el funcionamiento del dispositivo. Siguiendo con la clasificación de dispositivos realizada en el capítulo 1, podemos agrupar los tipos de dispositivos en dos grupos: el primero formado por los dispositivos móviles, en el que entrarían los dispositivos personales que acompañan al usuario en todo momento, y en otro los dispositivos fijos, que lo formarían los dispositivos de función específica y los sensores y actuadores. Los dispositivos móviles deberían tener un tiempo de disponibilidad variable dependiendo de los propios horarios del usuario: jornada laboral de 4 horas por la mañana, comida en casa y descanso de 2 horas y vuelta al trabajo de nuevo otras 4 horas, también puede ocurrir que un día tenga una reunión fuera de su oficina y esté 3 horas en otro lugar. Si este usuario tiene una PDA podemos considerar que en horario de trabajo, el tiempo de disponibilidad que tiene asociado es de 4 horas, si en la agenda de trabajo tiene una reunión programada durante 3 horas, en esa nueva situación la PDA empleará como tiempo de disponibilidad 3 horas... y así podríamos seguir enumerando otras muchas situaciones.

Estos patrones de movilidad pueden obtenerse en dispositivos personales gracias a información personal del usuario, como por ejemplo su agenda personal, o gracias a determinas tecnologías de red, como, por ejemplo, GSM o a sistemas específicos de localización basada en protocolos inalámbricos.

En los dispositivos fijos, el tiempo de disponibilidad no es tan crítico de cara al funcionamiento del protocolo. Aunque en estos dispositivos, los administradores podrán emplear el tiempo de disponibilidad para reducir o limitar su uso, por ejemplo, el aire acondicionado sólo estará en funcionamiento desde las 8:00 hasta las 20:00, después de este tiempo no podrá ser regulado por ningún dispositivo, así que el tiempo de disponibilidad con el que se ofrece variará dependiendo de la hora a la que se anuncie, siendo de 12 horas a las 8:00 y de 2 horas a las 18:00.

En las simulaciones que hemos realizado a lo largo de este capítulo, los dispositivos se creaban siguiendo una distribución exponencial cuyo valor medio es el que tomábamos como tiempo de disponibilidad, por lo tanto, ya estábamos introduciendo cierto error entre el tiempo de disponibilidad con la que se anuncia un dispositivo y su tiempo de vida real en la red.

De todas formas, hemos realizado una serie de simulaciones introduciendo mayor error en el tiempo de disponibilidad para ver qué efecto produce en las prestaciones del protocolo PDP cuando no existen mecanismos de consistencia de cachés[1]. Consideramos que el tiempo de vida de los dispositivos sigue una distribución exponencial de media μ, el tiempo de disponibilidad con el que se anuncian es $T = \mu \times \varepsilon$, donde ε es el error realizado en la estimación del tiempo de disponibilidad. En las Figuras 4.26, 4.27 y 4.28 vemos cómo afecta el error en el número de mensajes, en la tasa de servicios descubiertos y en la tasa de servicios falsos, respectivamente. Como se puede observar estimar al alza el tiempo de disponibilidad de un dispositivo produce un incremento considerable de la tasa de servicios falsos (que puede ser corregido mediante mecanismos de consistencia de cachés), mientras que estimar a la baja aumenta muy ligeramente el número de mensajes y reduce de manera significativa la tasa de servicios falsos. Por lo tanto, es mejor estimar a la baja el valor del tiempo de disponibilidad para obtener mejores prestaciones del protocolo.

4.6. Consideraciones sobre seguridad

El protocolo propuesto en esta tesis permite reducir considerablemente el número de mensajes transmitidos, manteniendo una tasa de descubrimiento de servicios alta y una tasa de servicios falsos muy reducida. Esto se consigue en gran medida por explotar la cooperación de los dispositivos que forman la red y por lo tanto, se parte de la creencia de que los dispositivos que forma la red son confiables, aunque esto no siempre es cierto. Por ello hemos analizado, los pro-

[1]Si existen mecanismos de consistencia de caché, las prestaciones del protocolo no se ven afectadas

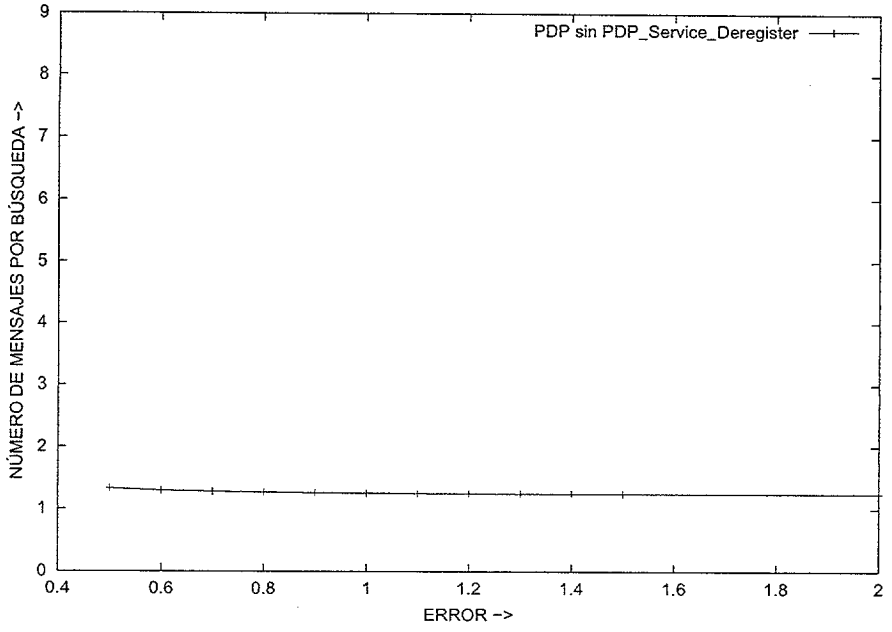


Figura 4.26: Número de mensajes respecto al error en tiempo de disponibilidad en PDP.

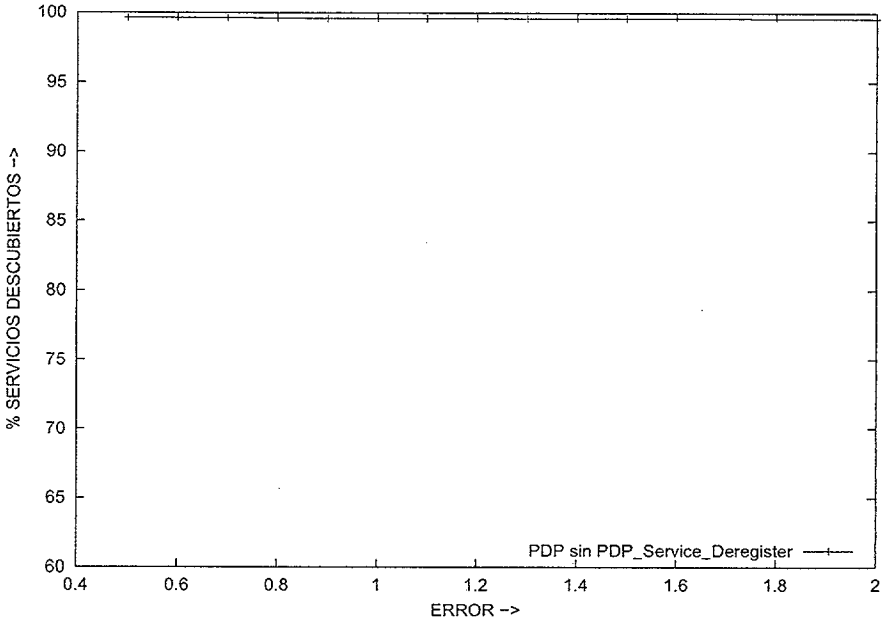


Figura 4.27: Tasa de descubrimiento de servicios respecto al error en tiempo de disponibilidad en PDP.

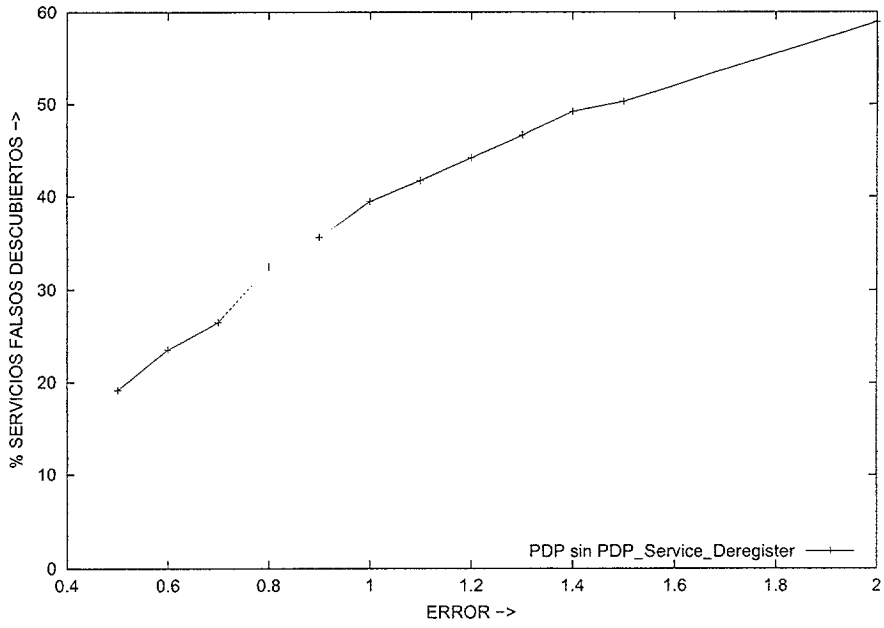


Figura 4.28: Tasa de servicios falsos respecto al error en tiempo de disponibilidad en PDP.

blemas de seguridad que presenta el protocolo y realizamos un análisis detallado de los más importantes:

1. **Llenado de cachés con servicios falsos**

   PDP basa gran parte de sus optimizaciones en el uso de una caché de servicios en todos los dispositivos. Esta caché tiene un tamaño limitado y si se escuchan servicios nuevos que no caben en la caché, se eliminan los que tienen un menor tiempo de expiración para almacenar estos nuevos.

   Un dispositivo malicioso podría anunciar continuamente servicios falsos, que obliguen a los demás dispositivos a borrar entradas correctas en su caché para almacenar estas entradas falsas. El principal efecto sería el aumento del número de mensajes por petición, porque al transmitir un mensaje de búsqueda, la lista de servicios conocidos es menos acertada respecto a lo que existe realmente en la red, por lo que se producirán más respuestas a esta petición.

   Además, los mecanismos de consistencia de caché introducidos en el protocolo hacen que cuando otros dispositivos detecten la indisponibilidad de estos servicios falsos, se envíen mensajes `PDP_Service_Deregister` para que se eliminen, con lo que se contribuye a aumentar más el número de

mensajes. Pero gracias a estos mecanismos, este tipo de ataque no afecta significativamente a la tasa de servicios falsos descubiertos.

En cuanto a la tasa de servicios descubiertos, tampoco se ve afectada porque la respuesta a la búsqueda de servicio, siempre se contrasta transmitiendo un mensaje de búsqueda, por lo que los que ofrecen el servicio podrán responder a la petición.

2. **Eliminación de entradas correctas en las cachés**

   Los mecanismos de consistencia de caché se introducen con el objetivo de disminuir el número de entradas falsas que existen en las cachés. En PDP permitimos que los dispositivos envíen mensajes `PDP_Service_Deregister` indicando la indisponibilidad de servicios de terceros, no sólo de sus propios servicios. Este mecanismo debe emplearse cuando se intenta acceder a dicho servicio y éste no se encuentra disponible.

   Un dispositivo malicioso podría realizar envíos de `PDP_Service_Deregister` de servicios de terceros sin detectar realmente su indisponibilidad. Esto no evitaría que se descubriesen estos servicios, ya que, como se explicó anteriormente, siempre se transmite un mensaje de búsqueda para contrastar la respuesta.

3. **Colapso de la red por envío masivo de mensajes PDP**

   Esto provocaría el colapso de la red debido al gran número de mensajes PDP, este ataque está dentro de los que se denominan ataques de denegación de servicio, pero que es inherente a todos los protocolos de red.

En PDP cada interfaz de red del dispositivo posee su propia caché de servicios, de esta forma aunque se estén realizando ataques en una determinada interfaz, el resto de interfaces no se verán afectadas.

En general, los problemas de seguridad que presenta PDP, están asociados también a otros protocolos de Internet, en concreto aquellos que tienen un ámbito de aplicación local, que emplean cachés y que hacen uso de broadcast/multicast, lo que no ha impedido que hayan sido ampliamente utilizados. Un ejemplo de estos protocolos, es ARP, RFC 826 [Plummer, 1982], que se definió para traducir direcciones IP a direcciones físicas. ARP utiliza una tabla, llamada también caché ARP, para realizar esta traducción. Cuando la dirección no se encuentra en la caché ARP, se envía por broadcast a la red una petición. Si una de las máquinas en la red reconoce su propia dirección IP en la petición, devolverá una respuesta ARP al host que la solicitó. La respuesta contendrá la dirección física del hardware así como su dirección IP. Por lo tanto, dispositivos maliciosos podrían emplear ARP para responder con direcciones físicas falsas o con direcciones

de otros dispositivos, introducir información errónea en la caché ARP o colapsar la red con mensajes broadcast...

En la mayoría de los protocolos de descubrimiento de servicios que hemos estudiado de forma detallada en el capítulo 3, los mecanismos de seguridad se centran en proporcionar autenticación basada en Infraestructura de Clave Pública (PKI), e incluso en protocolos como SLP estos mecanismos son opcionales debido a su complejidad y necesidad de configuraciones por parte de administradores para la gestión de claves. Como ya se indica en las consideraciones de seguridad de LLMNR, en redes dinámicas y sin infraestructura los mecanismos basados en PKI no son viables y por ello en LLMNR no se requiere autenticación.

La razón por la que las soluciones basadas en PKI no son aplicables en redes sin infraestructura se ilustra con el siguiente ejemplo: si imaginamos que un dispositivo desconocido se conecta a una red y presenta un certificado, tenemos que determinar la validez de ese certificado, y para ello necesitamos conectarnos con la Autoridad de Certificación que lo ha emitido. Si esta Autoridad no está preconfigurada como confiable o no es accesible por la red, no podremos validar la autenticidad de ese mensaje.

Esta consideración realizada para el caso de autenticación, es extensible para cualquier otro mecanismo de seguridad, como integridad, confidencialidad o control de acceso. Por lo tanto, es necesario un nuevo mecanismo para gestionar la seguridad en redes sin infraestructura. En esta línea en este Departamento se está trabajando en un modelo de confianza distribuido y en un protocolo de recomendación que permita establecer relaciones de confianza entre dispositivos de forma automática y minimizando las configuraciones manuales. Estos modelos tendrán en cuenta las acciones realizadas por los dispositivos para aumentar o disminuir el grado de confianza asignado, así por ejemplo, las acciones indicadas anteriormente de uso malicioso del protocolo PDP podrían emplearse a la hora de disminuir el grado de confianza otorgado a un dispositivo concreto. Para un análisis detallado de este problema y de las posibles soluciones se recomienda la lectura de los trabajos realizados por Florina Almenárez.

4.7. Consideraciones sobre la descripción de servicios

PDP ha sido diseñado como un algoritmo de descubrimiento de servicios, que propone un nuevo comportamiento de los clientes, a la hora de enviar peticiones de búsquedas de servicios, y de los servidores, a la hora de generar las correspondientes respuestas. Por lo tanto, este algoritmo es aplicable a otros pro-

tocolos de descubrimiento de servicios con arquitectura distribuida, sin necesidad de realizar cambios significativos en la estructura de los mensajes, y totalmente independiente de cómo se realice la descripción de los servicios y de cómo se detecta si el servicio solicitado corresponde con alguno de los servicios ofrecidos.

En el anexo B se realiza una descripción detallada de PDP que permite realizar implementaciones interoperables del mismo. Nos hemos basado en la RFC del protocolo SLP v2, por ser el estándar actual en Internet, centrándonos en su funcionamiento distribuido e incluyendo los mínimos cambios en el formato de los paquetes y reutilizando la descripción de servicios propuesta, basada en URLs y especificada en la RFC 2609.

Basándose en esta especificación se han realizado las implementaciones del protocolo PDP en el simulador de red Network Simulator [Casillas, 2003] y en el lenguaje de programación multiplataforma para dispositivos limitados, J2ME, que nos ha permitido probarlo en dispositivos reales. En concreto, en esta última implementación se ha empleado la máquina virtual J9 de IBM y el desarrollo se ha llevado a cabo sobre el Personal Profile que proporciona soporte a multicast, el ejecutable obtenido ocupa 22 KB sin emplear ninguna técnica para optimizar su tamaño y se ha probado satisfactoriamente en un Pocket PC, [Perea, 2003]. Se puede encontrar información detallada de estos desarrollos en `http://www.it.uc3m.es/celeste/tesis/`.

4.8. Conclusiones

En este capítulo hemos realizado un diseño razonado del protocolo PDP, para ello hemos partido de un modo *pull* puro sobre el que hemos introducido una serie de modificaciones hasta llegar a nuestra propuesta. Primeramente, se ha realizado un estudio analítico de las prestaciones de los protocolos teóricos (*pull*, *push* y directorio) en cuanto a número de mensajes transmitidos por búsqueda de servicio, tasa de servicios descubiertos y tasa de servicios falsos. De este estudio hemos obtenido dos conclusiones importantes, la primera es que el protocolo propuesto debe ser distribuido y no debe basarse en un servicio de directorio, y la segunda es que para obtener unas mejores prestaciones debemos mezclar las características de los modos *pull* y *push*.

Así, partiendo de un modo *pull* con caché en el que los mensajes de respuesta se envían por difusión, como en un modo *push*, hemos construido varias versiones del protocolo propuesto PDP, apoyando la introducción de cada una de los modificaciones propuestas mediante resultados de simulación y valorando en cada caso la mejora obtenida. De este modo, se ha llegado a una versión final del

protocolo, en el que se ha comprobado que nuestra propuesta disminuye considerablemente el número de mensajes por búsqueda, mientras que mantiene una tasa de servicios descubiertos próxima al 100 % y de servicios falsos próxima al 0 %, incluso en escenarios en los que todos los dispositivos tienen un tiempo de vida muy reducido.

A modo de resumen de los resultados expuestos en este capítulo, en las Figuras 4.29, 4.30 y 4.31 se presenta la evolución de las prestaciones del protocolo con las sucesivas modificaciones, se ha etiquetado cada curva con el apartado correspondiente en el que se ha descrito y analizado.

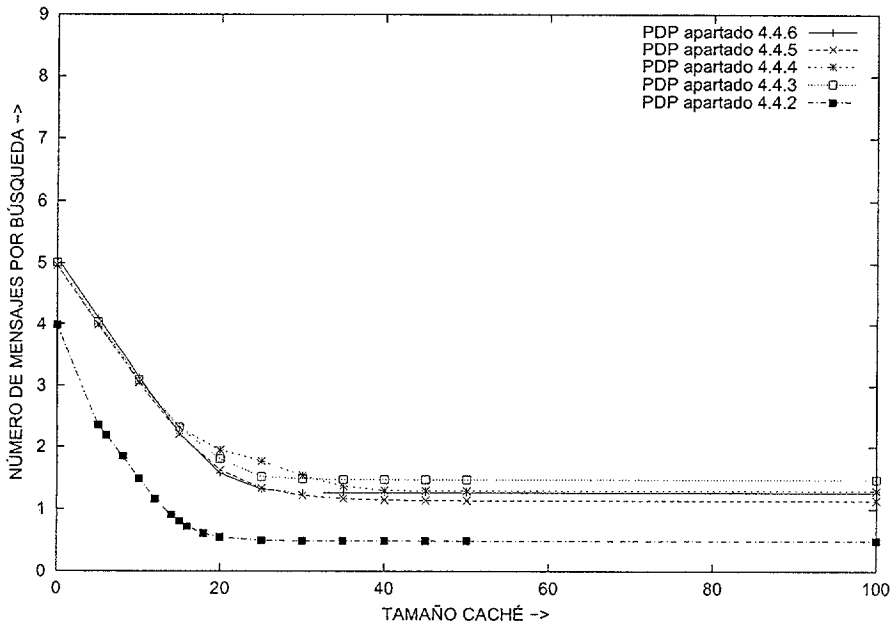


Figura 4.29: Evolución del diseño de PDP en cuanto a número de mensajes por búsqueda.

En este capítulo también se han realizado consideraciones sobre la descripción de servicios y sobre la seguridad del protocolo diseñado, y hemos visto cómo influye en las prestaciones del protocolo las desviaciones en la estimación del tiempo de disponibilidad de los dispositivos, viendo que una estimación conservadora nos permite seguir obteniendo una buenas prestaciones del protocolo, incluso sin emplear mecanismos de consistencia de cachés.

En el próximo capítulo, realizamos una evaluación de prestaciones más detallada de PDP, y lo comparamos con algunos de los protocolos descritos en el capítulo 3 en diferentes escenarios.

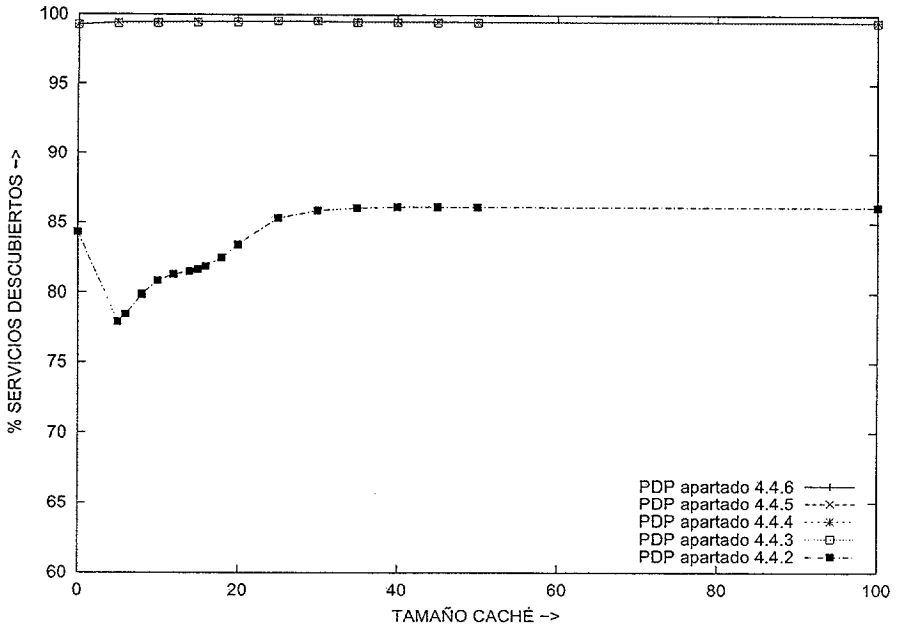


Figura 4.30: Evolución del diseño de PDP en cuanto a tasa de servicios descubiertos.

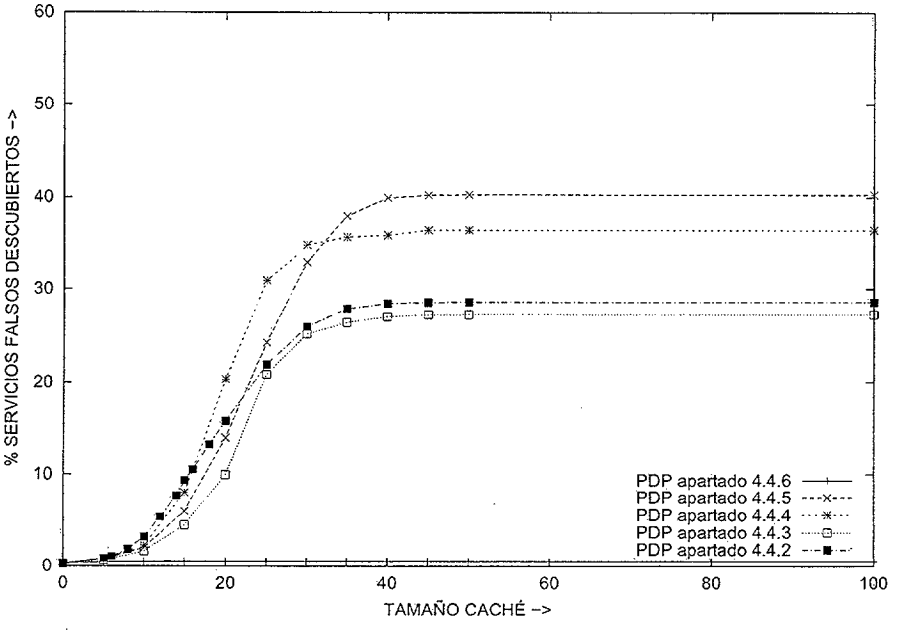


Figura 4.31: Evolución del diseño de PDP en cuanto a servicios falsos descubiertos.

Capítulo 5

Análisis de prestaciones de Pervasive Discovery Protocol

En el capítulo 4 hemos realizado una descripción detallada de PDP, justificando mediante análisis y simulación su diseño. Hemos visto que PDP cumple los objetivos que nos hemos impuesto para el diseño de un protocolo de descubrimiento de servicios para entornos de computación ubicua, ya que es un protocolo totalmente distribuido, en el que se maximiza la cooperación entre los dispositivos para obtener unas mejores prestaciones. Además, dependiendo del tipo de búsquedas que realicen las aplicaciones, el protocolo se comporta de forma distinta para reducir el número de mensajes transmitidos. Así, hemos comprobado que PDP permite reducir considerablemente el número de mensajes transmitidos por búsqueda de servicio realizada, con unas tasas de servicios descubiertos y de servicios falsos adecuadas, próximas al 100 % y al 0 %, respectivamente.

En este capítulo realizamos un estudio detallado de las prestaciones de PDP en diferentes escenarios y respecto a parámetros clave del protocolo. Además, incluimos un estudio comparativo de PDP respecto a otros protocolos de descubrimiento de servicios para evaluar las mejoras obtenidas con nuestra propuesta. Así, en la sección 5.1, se estudian las prestaciones de PDP respecto al tiempo de disponibilidad de los dispositivos, al número de dispositivos que existen en la red y al tamaño de su caché. En la sección 5.2 realizamos la comparativa de PDP respecto a otros protocolos tanto teóricos, *push*, *pull* y directorio, como de Internet, SLP y SSDP. Finalizamos con las conclusiones, en la sección 5.3.

5.1. Prestaciones de Pervasive Discovery Protocol

En esta sección, analizamos en detalle las prestaciones de PDP respecto a diferentes parámetros, en concreto, veremos cómo se comporta dependiendo del tiempo de disponibilidad de los dispositivos que forman el entorno, respecto al número de dispositivos y al tamaño de su caché.

5.1.1. Prestaciones de PDP respecto al tiempo de disponibilidad

En las simulaciones realizadas para analizar las prestaciones de PDP respecto al tiempo de disponibilidad, hemos partido del siguiente escenario: en la red existen una media de 20 dispositivos, con 5 tipos diferentes de servicios, cada dispositivo busca un servicio de manera aleatoria, siguiendo una distribución exponencial de media 60 segundos, su tiempo de disponibilidad toma los valores 600, 1200, 2400, 3600 y 9200 segundos. Se han repetido las simulaciones para diferentes valores del tamaño de la caché: 0, 10, 20 y 100. Hemos seleccionado como extremo superior 100, porque es un valor suficientemente alto para que no se borren entradas debido a la limitación del tamaño de la caché, dado que en media existirán 20 servicios en la red.

El tiempo de disponibilidad con el que se anuncian los dispositivos está ajustado al valor de la media de su tiempo de vida, que sigue una distribución exponencial.

En la Figura 5.1, vemos que el número de mensajes por búsqueda no es sensible al tiempo de disponibilidad de los dispositivos, ya que prácticamente se mantiene constante. Lo más destacado que se observa en esta figura son las diferencias que existen para diferentes tamaños de caché, así para tamaño de caché 0, PDP se comporta como un modo *pull* puro, por lo que el número de mensajes es próximo a 5, a medida que aumentamos el tamaño de la caché el número de mensajes disminuye y se aproxima a 1 para tamaño 100, aunque ya a partir del valor 20 se obtiene una reducción importante (aproximadamente 1,4).

En las Figuras 5.2 y 5.3 se muestra la tasa de servicios descubiertos y servicios falsos, respectivamente. Observar que entre el valor máximo y mínimo en la escala de las ordenadas solamente existe una diferencia de un 1 %. Los resultados obtenidos son mejores en entornos más estáticos, aunque la diferencia no es muy significativa. Además, estos dos parámetros son menos sensibles al tamaño de la caché, ya que las diferencias que existen para los diferentes valores no son tan

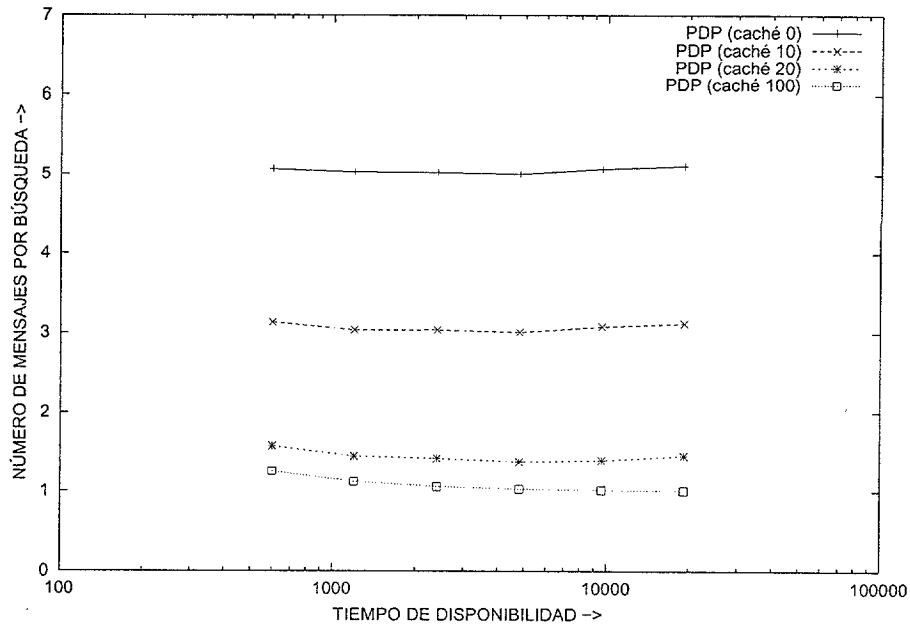


Figura 5.1: Número de mensajes por búsqueda en PDP respecto al tiempo de disponibilidad para cachés de tamaños 0, 10, 20 y 100.

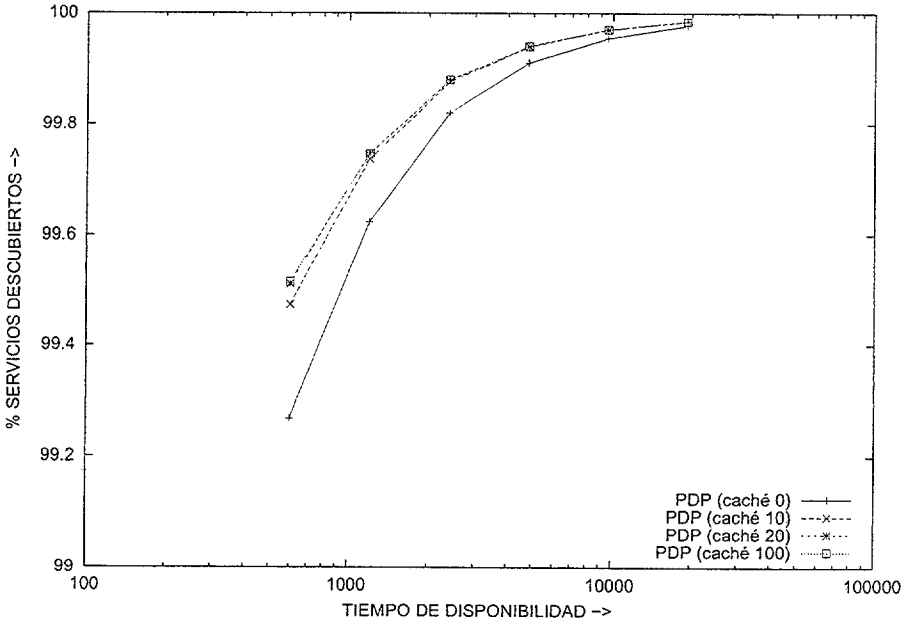


Figura 5.2: Tasa de servicios descubiertos en PDP respecto al tiempo de disponibilidad para cachés de tamaños 0, 10, 20 y 100.

significativas como en el caso del número de mensajes, aunque sí que se observa que para tamaños a partir de 20 las diferencias se reducen.

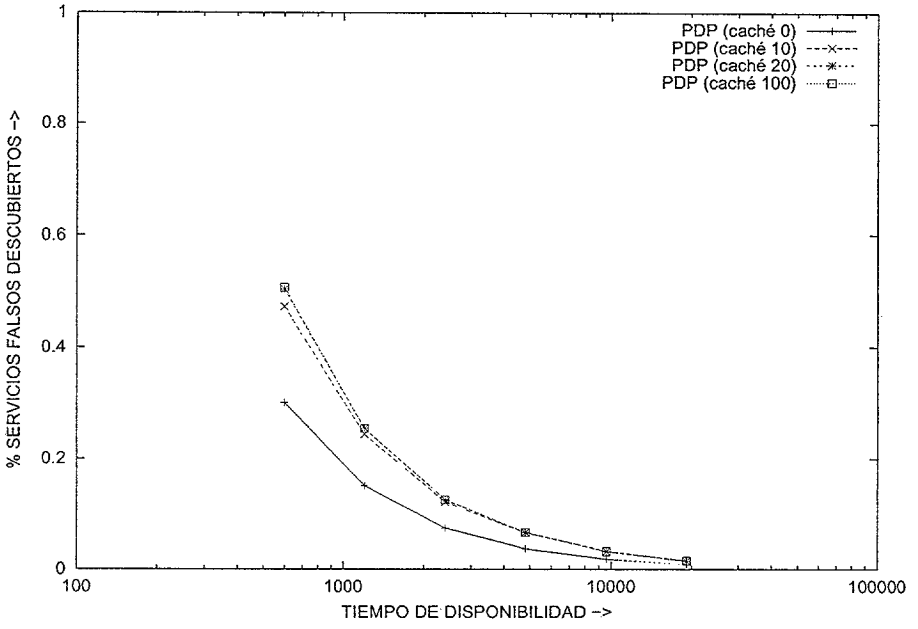


Figura 5.3: Tasa de servicios falsos descubiertos en PDP respecto al tiempo de disponibilidad para cachés de tamaños 0, 10, 20 y 100.

5.1.2. Prestaciones de PDP respecto al número de dispositivos

En las simulaciones realizadas para analizar las prestaciones de PDP respecto al número de dispositivos, hemos partido del siguiente escenario: el número medio de dispositivos que existen en la red toma los valores de 5, 10, 15, 20, 25 y 30, existen 5 tipos diferentes de servicios, y cada dispositivo busca un servicio de manera aleatoria, siguiendo una distribución exponencial de media 60 segundos y tienen un tiempo de disponibilidad de 600 segundos. Se han repetido las simulaciones para diferentes valores del tamaño de la caché: 0, 10, 20 y 100.

Como en el caso anterior, el tiempo de disponibilidad con el que se anuncian los dispositivos está ajustado al tiempo medio que permanecen los dispositivos en la red.

En la Figura 5.4, observamos como para caché 0 el valor del número de mensajes por búsqueda es directamente proporcional al número de servidores que ofrecen el mismo tipo de servicio, ya que su comportamiento es el mismo que un

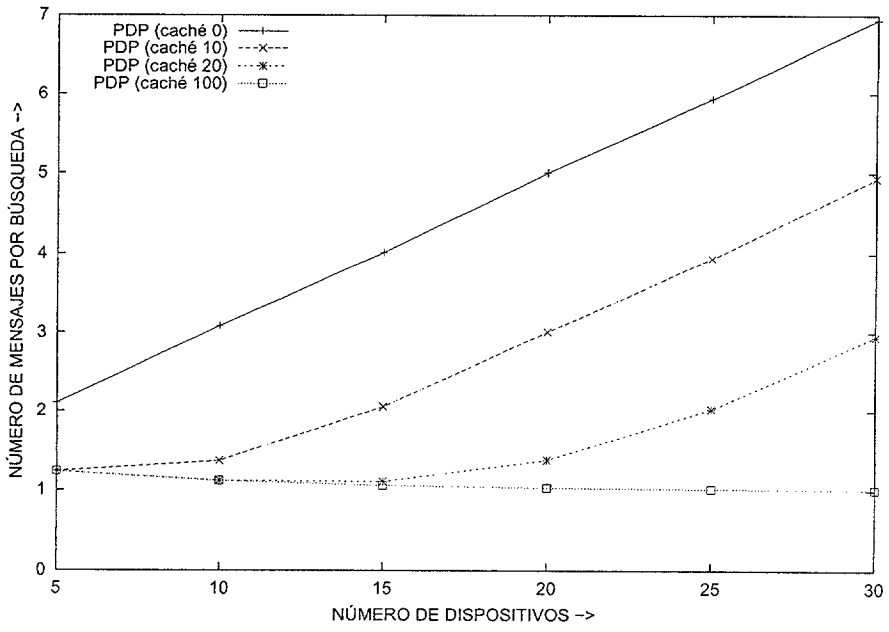


Figura 5.4: Número de mensajes por búsqueda en PDP respecto a número de dispositivos para cachés de tamaños 0, 10, 20 y 100.

modo *pull* puro. A medida que aumentamos el tamaño de la caché, este valor se reduce y se mantiene prácticamente constante en escenarios con un número bajo de dispositivos. Ésto es debido a que el tamaño de la caché es mayor que el número de servidores existentes en la red y por lo tanto, PDP alcanza sus mejores prestaciones. A medida que el número de dispositivos aumenta, la curva tiene un crecimiento similar a un modo *pull* puro, ya que la caché no tiene un tamaño lo suficientemente alto respecto al número de servidores existentes en la red.

Respecto a la tasa de servicios descubiertos y de servicios falsos, que se observa en las Figuras 5.5 y 5.6, cuando aumenta el número de dispositivos en la red, el tamaño de la caché es lo suficientemente bajo respecto al número de servidores existentes, para que las prestaciones del protocolo se aproximen a las obtenidas para caché 0. Observar, como en el apartado anterior, que entre el valor máximo y mínimo en la escala de las ordenadas solamente existe una diferencia de un 1 %, y por lo tanto, las diferencias obtenidas en los diferentes curvas no son muy significativas.

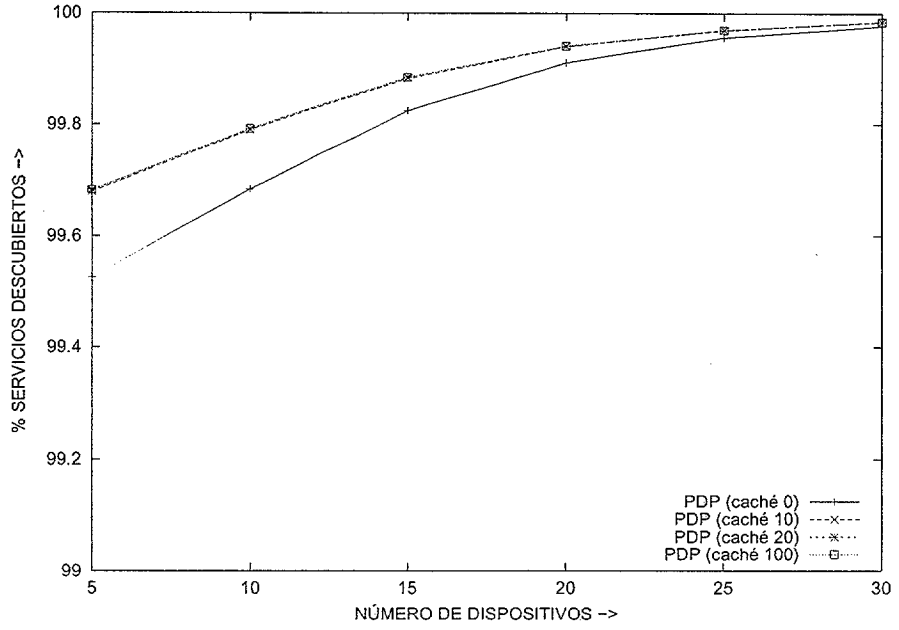


Figura 5.5: Tasa de servicios descubiertos en PDP respecto a número de dispositivos para cachés de tamaños 0, 10, 20 y 100.

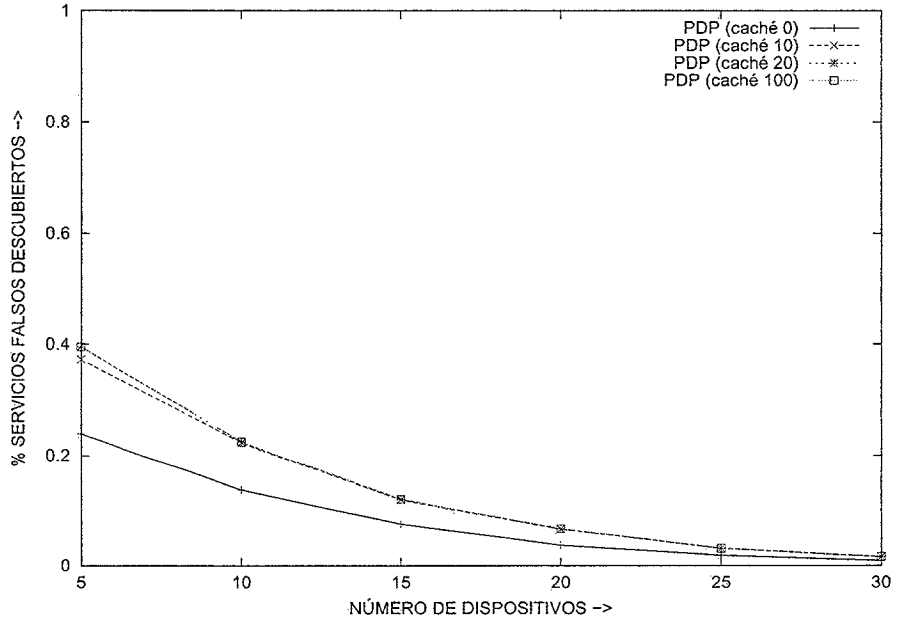


Figura 5.6: Tasa de servicios falsos descubiertos en PDP respecto al número de dispositivos para cachés de tamaños 0, 10, 20 y 100.

5.1.3. Prestaciones de PDP respecto al tamaño de la caché

De los resultados obtenidos en los estudios realizados en los apartados anteriores, podemos concluir que el tamaño de la caché es un parámetro decisivo en las prestaciones del protocolo PDP. Se observa que para tamaños de caché próximos al número de servicios que existen en la red, en los escenarios anteriores 20, el número de mensajes por búsqueda ya se reduce considerablemente, a un valor inferior a 1,5, mientras que una vez sobrepasado este valor la mejora obtenida no es significativa, por lo que un tamaño excesivamente alto no mejora las prestaciones y además, consume un mayor número de recursos en el dispositivo.

A la hora de seleccionar el tamaño de la caché debe tenerse en cuenta en primer lugar las restricciones de memoria y capacidad de proceso del dispositivo, y en principio, tendrá un valor por defecto adaptado a estas limitaciones. Los dispositivos personales, y por lo tanto móviles, emplearán tamaños de caché pequeños no sólo por ser dispositivos limitados en memoria y con tiempos de disponibilidad bajos, sino también, porque el hecho de que tener cachés mayores, hace que respondan a más peticiones y por lo tanto, consuman más baterías (ver sección 5.2.2). En los dispositivos de función específica, normalmente con tiempos de disponibilidad mayores, el valor seleccionado dependerá fundamentalmente, de sus limitaciones de memoria y de baterías, en caso de que las tengan. En dispositivos fijos tipo PC, las cachés empleadas deberían tener un tamaño elevado, de manera que sean los que respondan a un mayor número de peticiones y de esta forma, reduzcan el consumo de los dispositivos limitados que existen a su alrededor y que buscan servicios.

En la Figura 5.7, observamos la tasa de servicios falsos[1] en el mismo escenario que para las figuras de la sección 5.1.1, pero eliminando de PDP los mecanismos de consistencia de cachés. Vemos que el hecho de tener cachés altas provoca que aumente la tasa de servicios falsos hasta valores muy elevados, aproximadamente del 50 %, pero que en cuanto la caché empleada está próxima o por debajo del número de servidores existentes en la red, este valor se reduce hasta el 15 %. Por lo tanto, es importante seleccionar un valor no muy elevado del tamaño de la caché para obtener las mejores prestaciones de PDP, incluso en el caso de no emplear los mecanismos de consistencia de cachés definidos en el protocolo.

Por lo tanto, podemos concluir que en PDP existe un tamaño óptimo de caché, que es un valor próximo al número de servicios que existen en la red. Así, como trabajo futuro y como línea de continuación de esta tesis, se podrían

[1]El número de mensajes por búsqueda y la tasa de servicios descubiertos son similares a las obtenidas en PDP con mecanismos de consistencia de cachés.

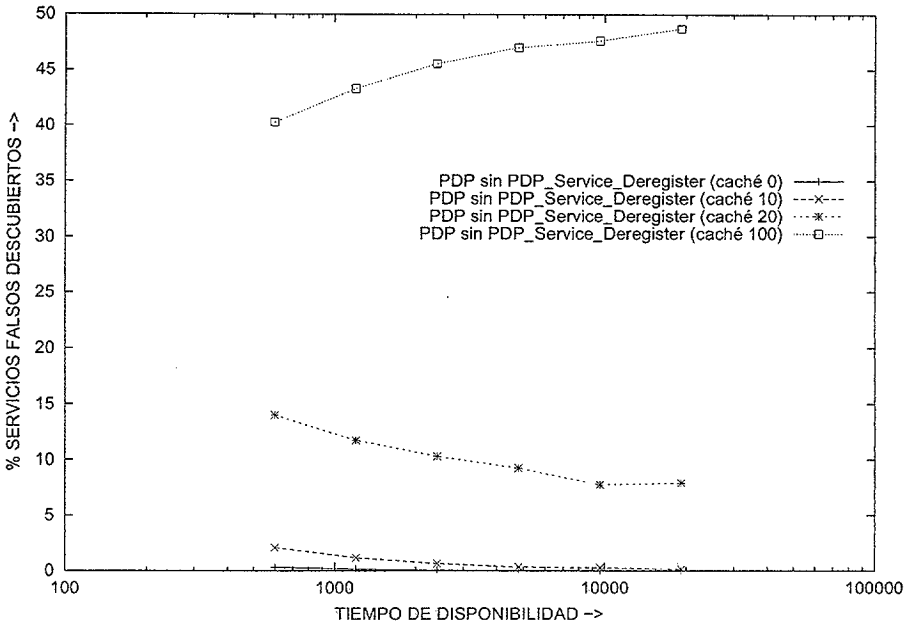


Figura 5.7: Tasa de servicios falsos descubiertos en PDP sin mecanismos de consistencia de caché respecto al tiempo de disponibilidad para cachés de tamaños 0, 10, 20 y 100.

introducir mecanismos que permitieran que el tamaño de la cachés se adaptase dinámicamente dependiendo del número de servidores existentes en el entorno, sobre todo en el caso de dispositivos limitados, y con tiempo de disponibilidad bajo.

5.2. Comparativa de PDP con otros protocolos

Una vez revisadas las prestaciones del protocolo propuesto respecto a diferentes parámetros: tiempo de disponibilidad, número de dispositivos y tamaño de la caché, en esta sección, realizamos un estudio comparativo de las prestaciones de PDP respecto al modo *push*, al modo *pull*, al protocolo SLP, al protocolo SSDP y a las soluciones basadas en directorio (cuando sea aplicable) en varios escenarios.

En el modo *push* consideramos dos versiones, con tiempo entre anuncios de 60 segundos y de 12 segundos. Respecto al protocolo SSDP, recordar que soporta un modo de funcionamiento *pull* y *push* y no especifica cuándo se emplea uno u otro, se deja abierto a las aplicaciones que lo utilicen, es por ello que sólo se hace referencia explícita a este protocolo en la comparativa realizada para el soporte

a aplicaciones de tipo "buscador", en la sección 5.2.4.

5.2.1. Escenarios homogéneos

En las simulaciones que analizamos en este apartado comparamos nuestro protocolo en un escenario homogéneo, es decir, en el que todos los dispositivos tienen aproximadamente las mismas características de movilidad. Esta situación se daría por ejemplo, en escenarios abiertos, tipo salas de espera de aeropuertos, salas de reuniones, etc. en las que se forma una red espontáneamente, sin infraestructura fija, entre los dispositivos personales de los usuarios, como PDAs, teléfonos móviles o portátiles.

Los parámetros seleccionados para estas simulaciones son: en la red existen una media de 20 dispositivos, con un tiempo medio de disponibilidad variando entre 600 y 19200 segundos, un tamaño de caché de 100 entradas, con 5 tipos diferentes de servicios y en los que cada dispositivo solicita la búsqueda de un tipo de servicio cada 60 segundos, el tipo se selecciona de forma aleatoria entre los 5 existentes.

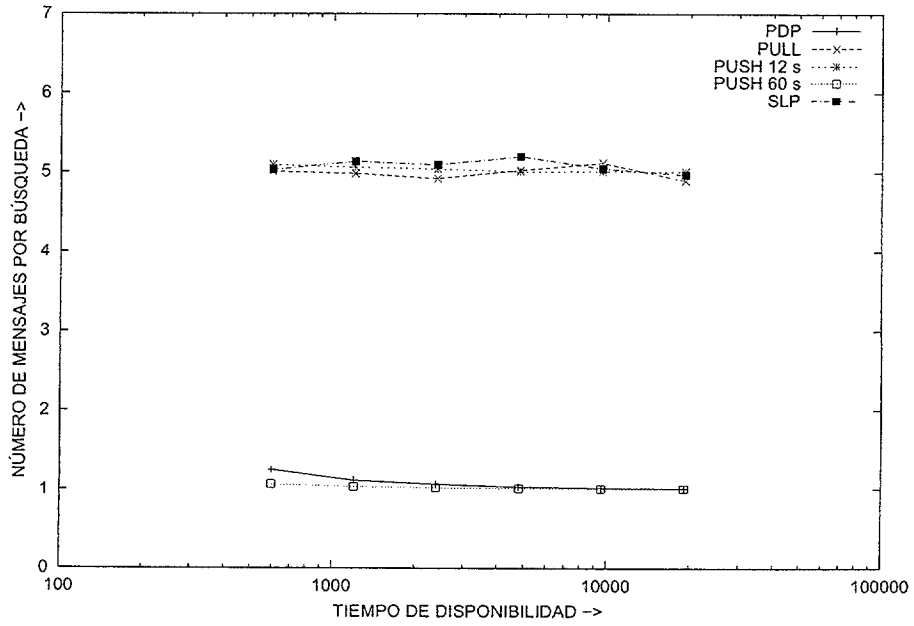


Figura 5.8: Comparativa del número de mensajes transmitidos por búsqueda en un escenario homogéneo.

En la Figura 5.8, se observa que el número de mensajes por búsqueda es menor en PDP y en el modo *push*(60 segundos), en ambos casos se aproxima a 1 mientras

que el obtenido para los demás protocolos está próximo a 5. Sin embargo, en la Figura 5.9 observamos que respecto a las tasa de descubrimiento de servicios, PDP se aproxima al 100% junto con los protocolos SLP, *push*(12 segundos) y *pull*, y en cambio el *push*(60 segundos) tiene una tasa de descubrimiento menor, sobre todo cuanto menor es el tiempo de disponibilidad de los dispositivos.

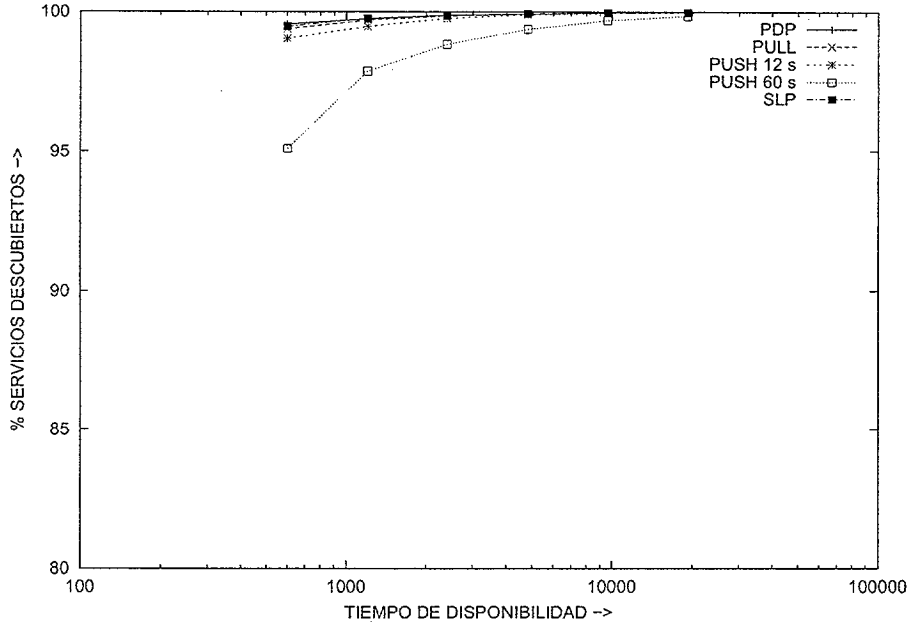


Figura 5.9: Comparativa de la tasa de descubrimiento de servicios en un escenario homogéneo.

En cuanto a la tasa de servicios descubiertos falsos, vemos como en PDP se obtienen resultados equiparables al modo *pull* y a SLP, mientras que los modos *push* presentan unas peores prestaciones.

A la vista de los resultados obtenidos podemos decir que en escenarios homogéneos, PDP presenta unas prestaciones similares al modo *pull* y a SLP en cuanto a tasa de descubrimiento de servicios y de servicios falsos, mientras que reduce considerablemente el número de mensajes por búsqueda, pasando, aproximadamente de 5 a 1.

Indicar que las diferencias entre el modo *pull* y SLP se deben a que en este último protocolo aunque no exista directorio, los UAs deben realizar búsquedas de DAs, lo que aumenta un poco el tráfico generado, como se puede observar en las figuras anteriores.

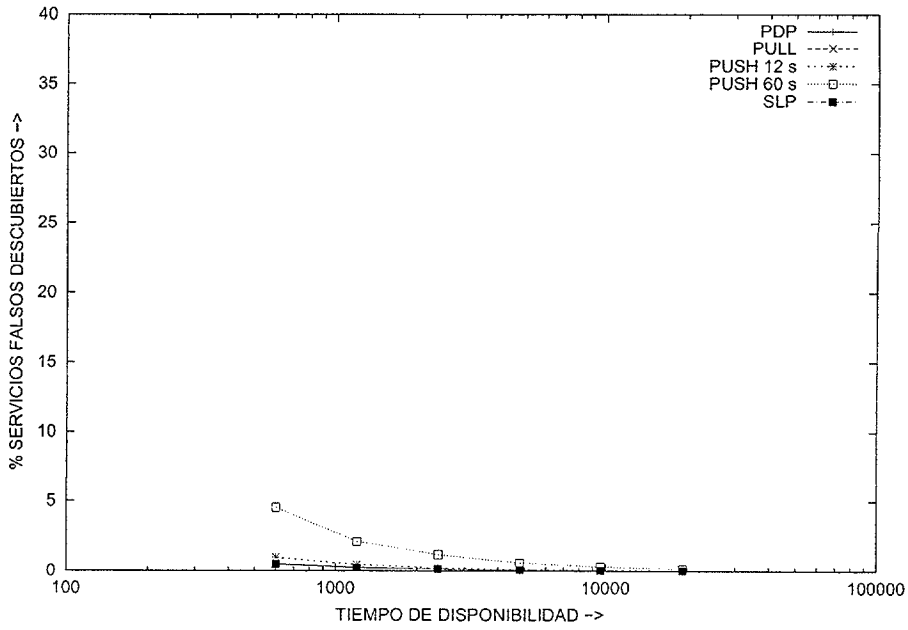


Figura 5.10: Comparativa de la tasa de descubrimiento de servicios falsos en un escenario homogéneo.

5.2.2. Escenarios heterogéneos

En las simulaciones que analizamos en este apartado comparamos nuestro protocolo en un escenario heterogéneo, en el que los dispositivos tienen características distintas de movilidad. Esta situación se daría por ejemplo, en el hogar o en oficinas en las que los usuarios tienen dispositivos móviles que interaccionan con dispositivos fijos embebidos en el entorno físico que les rodea. Los servicios que se buscarían en estos escenarios son, por ejemplo, control del aire acondicionado, de iluminación, obtención de información de sensores de temperatura, envío de documentos a impresoras o a faxes, etc.

Los parámetros seleccionados para estas simulaciones son los mismos que en el escenario anterior, pero el 20 % de los dispositivos tienen un tiempo medio de disponibilidad 10 veces mayor que los demás. Como se puede observar en las Figuras 5.11, 5.12 y 5.13, las prestaciones obtenidas se mantienen respecto al escenario homogéneo.

Como describíamos en la sección 4.4.4, en PDP un Agente de Servicio antes de responder a un mensaje de búsqueda, genera un retardo aleatorio que es inversamente proporcional a su tiempo de disponibilidad y al número de servicios que conoce. El objetivo de introducir este mecanismo, es que en entornos heterogéneos

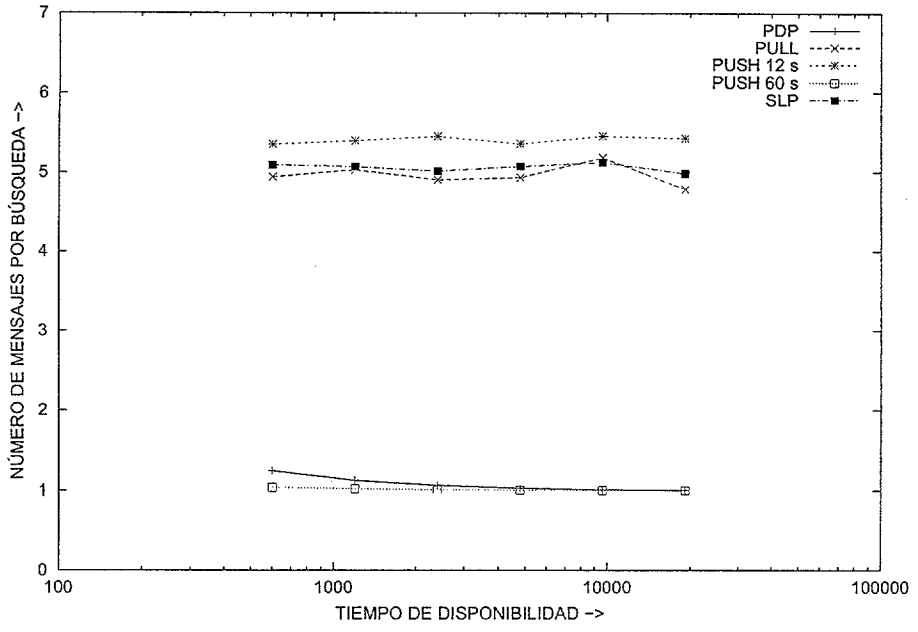


Figura 5.11: Comparativa del número de mensajes transmitidos por búsqueda en un escenario heterogéneo.

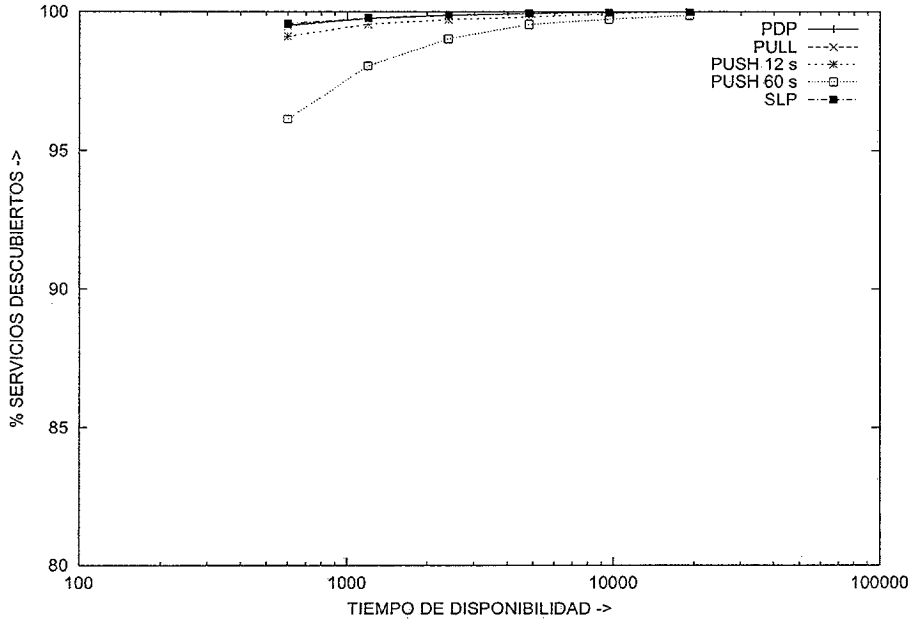


Figura 5.12: Comparativa de la tasa de descubrimiento de servicios en un escenario heterogéneo.

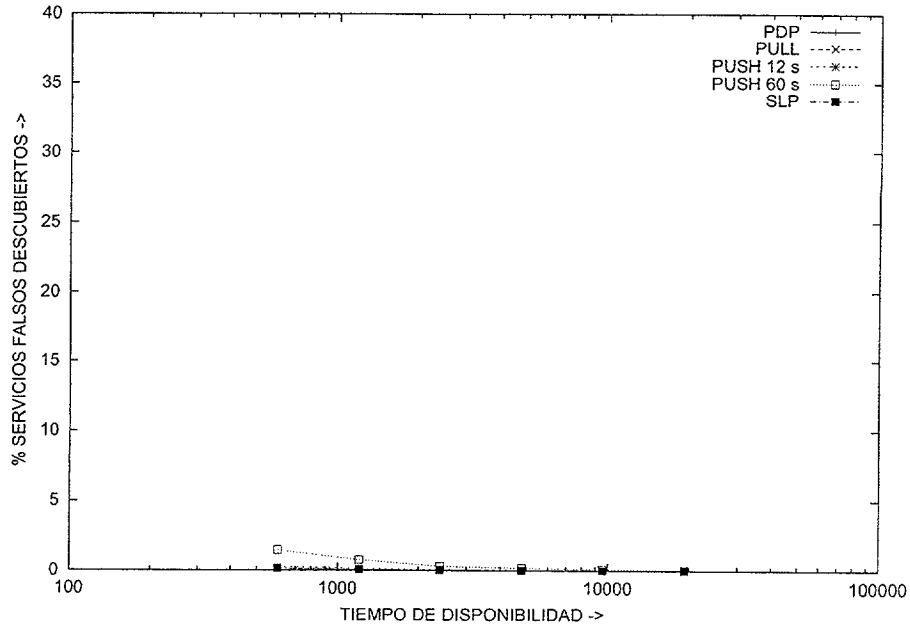


Figura 5.13: Comparativa de la tasa de descubrimiento de servicios falsos en un escenario heterogéneo.

los dispositivos que tienen un tiempo de disponibilidad mayor respondan a más peticiones que los que tienen un menor tiempo de disponibilidad. Lo que revierte en un menor consumo de baterías en los dispositivos que tienen un menor tiempo de disponibilidad, y por lo tanto, más restricciones de consumo.

Hemos realizado una simulación para observar qué porcentaje de respuestas transmiten los dispositivos dependiendo de su tiempo de disponibilidad en un escenario heterogéneo. Así, hemos considerado un escenario en el que existen en media 40 dispositivos con cinco tiempos de disponibilidad distintos: 500, 2500, 4500, 6500 y 9500 segundos. El porcentaje de dispositivos de cada tipo es uniforme, por lo tanto, existirán en media un 20 % de cada tipo. Los demás parámetros de la simulación son los mismos que en la simulación anterior, excepto el tamaño de la caché, que para los dispositivos con tiempo de disponibilidad 500 y 2500 segundos es 10, para los de tiempo de disponibilidad 4500 y 6500 segundos es 40 y para los de tiempo de disponibilidad 9500 segundos es 100. En la Figura 5.14 se observa el resultado de esta simulación.

La primera conclusión que se obtiene al observar esta gráfica es que el objetivo marcado con esta característica de PDP se cumple, ya que los dispositivos con mayor tiempo de disponibilidad responden a más mensajes de búsqueda. Si el retardo para responder a un mensaje se calculase independientemente del tiempo

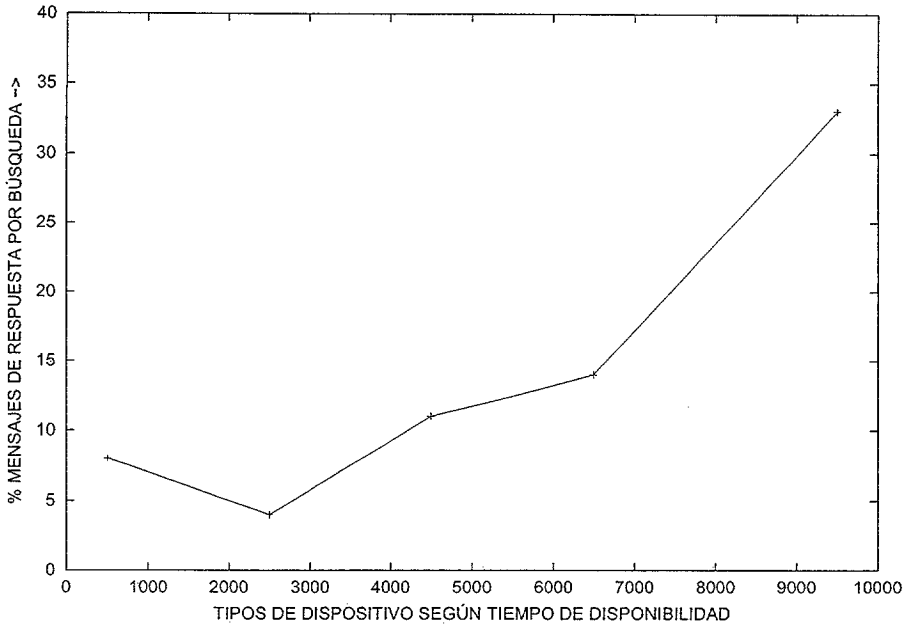


Figura 5.14: Porcentaje de mensajes de respuesta transmitidos por búsqueda respecto a tipos de dispositivos (según su tiempo de disponibilidad) en un escenario heterogéneo.

de disponibilidad, el porcentaje de mensajes de respuesta se distribuiría de forma uniforme entre los diferentes tipos de dispositivos, de tal forma, que los dispositivos con menores tiempos de disponibilidad responderían el mismo porcentaje de mensajes de búsqueda que los dispositivos con mayores tiempos de disponibilidad. Observar que la suma de los porcentajes obtenidos es aproximadamente del 70 %, esto es así, porque en PDP algunas peticiones de búsqueda no generan mensajes de repuesta[2], por lo tanto los dispositivos con un tiempo de disponibilidad de 9500 segundos, porcentualmente, responden a casi el 50 % de las peticiones de búsqueda.

También se observa en la gráfica, que los dispositivos con menor tiempo de disponibilidad responden más que los que tienen valores intermedios, este resultado era esperable, ya que al cambiar continuamente de red, en cada nuevo entorno tienen que responder a peticiones de sus propios servicios para darlos a conocer, ya que el resto de los dispositivos que existen a su alrededor todavía no los tienen almacenados en su cachés.

En la implementación realizada de PDP para los estudios de simulación, el mecanismo concreto que se ha empleado para que el retardo de las respuestas sea

[2]La lista de servicios conocidos incluidos en el mensaje de petición está actualizada.

inversamente proporcional al tiempo de disponibilidad y al número de servicios conocidos, es el indicado en la fórmula 5.1 (ver también anexo B). Como línea futura de investigación se podrían estudiar nuevos mecanismos y ver las mejoras que se obtienen en cada caso.

$$\text{Retardo} = \text{Uniforme}(0, \quad 3 * \frac{3600}{3600 + T_D * \text{Service\_Entries\_Number}}) \qquad (5.1)$$

Siendo,

- T_D: el tiempo de disponibilidad asociado al dispositivo.
- Service_Entries_Number: el número de servicios conocidos que se van a incluir en la respuesta.
- El valor de 3 se corresponde con el tiempo máximo que espera un dispositivo antes de transmitir un `PDP_Service_Reply`. Hemos limitado este valor para que no se produzcan grandes retardos en las respuestas a los mensajes de búsqueda.

5.2.3. Escenario con un dispositivo fijo

En este apartado consideramos un escenario en el que existe un dispositivo fijo, que, en el caso de protocolos que soportan funcionamiento centralizado (SLP), actúa como directorio. En las gráficas se incluye también una solución teórica basada en directorio. Los parámetros de la simulación son los mismos que los considerados en un entorno homogéneo (ver apartado 5.2.1).

En las Figuras 5.15 y 5.16 se muestran los resultados obtenidos para los diferentes protocolos respecto al número de mensajes por búsqueda y a la tasa de servicios descubiertos. Como podemos observar las soluciones con directorio, SLP y directorio teórico, disminuyen el número de mensajes transmitidos respecto a las mayoría de soluciones distribuidas.

En el caso de PDP, se sigue obteniendo un número menor de mensajes por búsqueda que en las soluciones con directorio. Además, como hemos analizado en el apartado anterior, la manera de generar el retardo antes de responder a un mensaje de petición de servicio, hace que en PDP el dispositivo fijo sea el que asuma el mayor porcentaje de respuestas, por lo que su comportamiento es muy similar a las soluciones de directorio pero disminuyendo aún más el número de mensajes transmitidos.

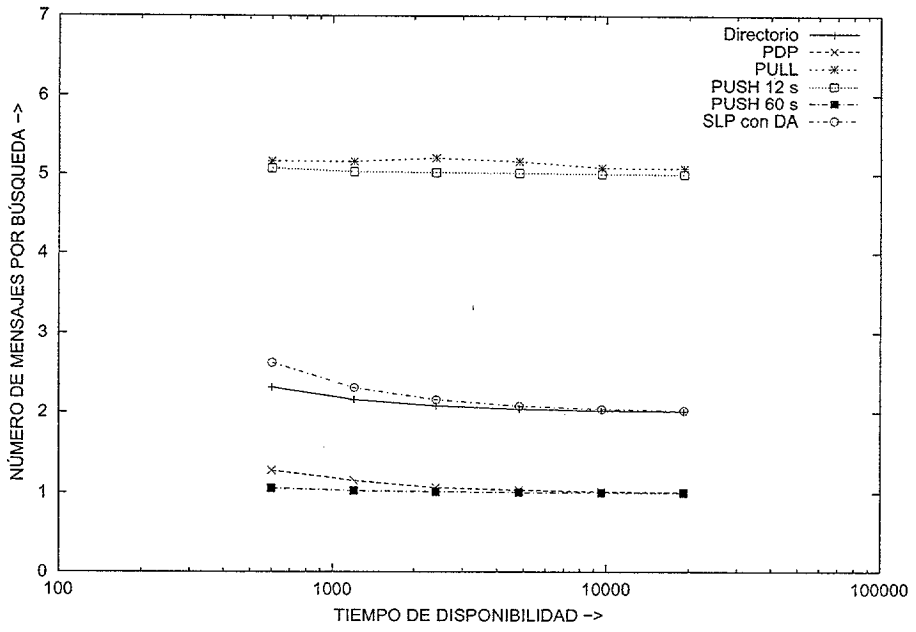


Figura 5.15: Comparativa del número de mensajes transmitidos por búsqueda en un escenario con un dispositivo fijo.

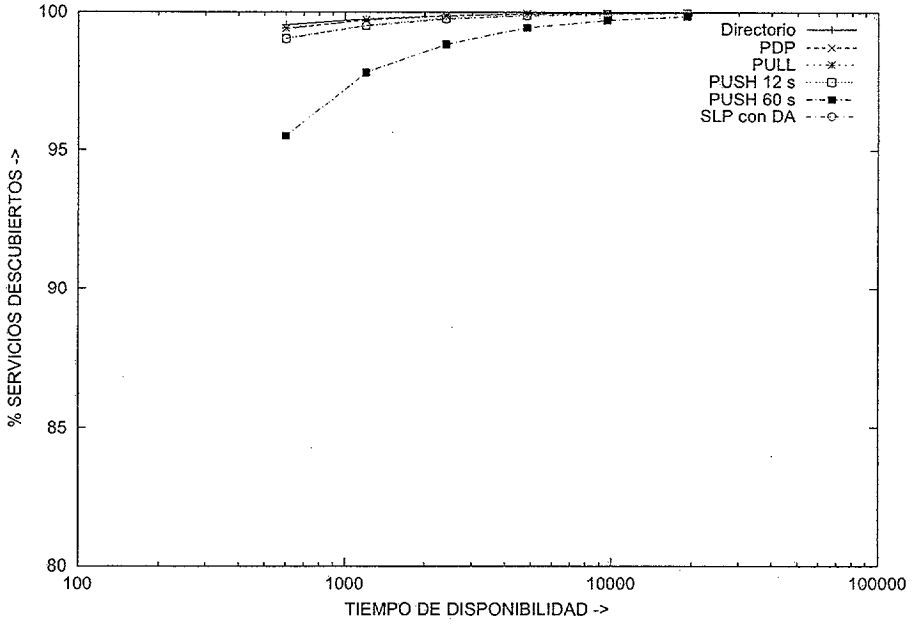


Figura 5.16: Comparativa de la tasa de descubrimiento de servicios en un escenario con un dispositivo fijo.

Notar, que las diferencias existentes entre SLP y la solución teórica de directorio, se deben a que en el segundo caso todos los dispositivos conocen el directorio, mientras que en SLP los dispositivos deben localizar el DA existente en la red, mediante mensajes especiales de búsqueda. Así, para escenarios con dispositivos con tiempo de disponibilidad bajos, SLP tiene un número de mensajes transmitidos por búsqueda mayor que un directorio teórico.

5.2.4. Soporte a aplicaciones de tipo "buscador"

Algunos de los protocolos de descubrimiento de servicios incluyen mecanismos para soportar aplicaciones tipo "buscador", es decir, aplicaciones que permiten mostrarle al usuario todos los servicios disponibles en el entorno que lo rodea. SLP y SSDP son dos de los protocolos que incluyen este soporte. En el caso de SLP, esta búsqueda se realiza en dos pasos, en primer lugar es necesario descubrir todos los tipos de servicios existentes en la red y después, localizar todos los servidores que ofrecen cada tipo de servicio. En el caso de SSDP, se define un nuevo tipo de servicio `ssdp:all` que se incluye en un mensaje de búsqueda transmitido por difusión a la red (modo *pull*), y al que todos los servidores responden indicando el servicio que ofrecen.

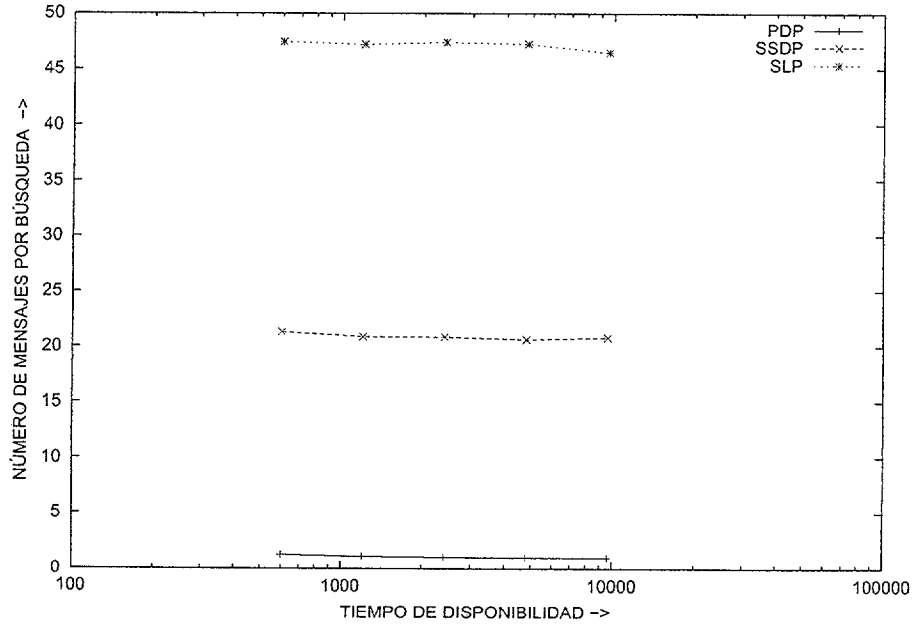


Figura 5.17: Comparativa del número de mensajes transmitidos por búsqueda de todos los servicios disponibles en la red.

En PDP también se ha dado este soporte introduciendo un tipo de servicio especial denominado ALL en las peticiones de búsqueda de tipo 1/n. Hemos realizado una serie de simulaciones para comparar las prestaciones de PDP, respecto a SLP y a SSDP, cuando se realizan búsquedas periódicas de todos los servicios existentes en la red. El escenario empleado es el siguiente: existen una media de 20 dispositivos en la red, con 5 tipos diferentes de servicios, cada dispositivo transmite una búsqueda de todos los servicios existentes en la red siguiendo una distribución exponencial de media 60 segundos, los tiempos de disponibilidad de los dispositivos toman los valores 600, 1200, 2400, 3600 y 9200 segundos.

En la Figura 5.17 se observa el número de mensajes por búsqueda de todos los servicios existentes en la red, para los tres protocolos estudiados. El que genera un mayor número de mensajes es SLP, en torno a 46, porque para cada petición de búsqueda primero es necesario buscar todos los tipos de servicios existentes en la red, y en media se obtienen tantas respuestas como dispositivos existen, por lo tanto 20 y después, por cada tipo de servicio se obtienen 4 respuestas, que es el número medio de servidores que ofrecen un determinado tipo de servicio ya que existen 5 tipos distintos en la red. En el caso de SSDP, se obtienen en media tantas respuestas como servidores existen, por lo tanto 20 y sumando el mensaje de petición, se obtienen en media 21 mensajes transmitidos por petición de búsqueda. En PDP, el resultado obtenido es mucho menor que en los casos anteriores, manteniéndose en un valor próximo a 1, ya que al realizarse búsquedas periodicas de todos los servicios existentes en la red, las cachés estarán muy actualizadas, por lo que los mensajes de petición generan ocasionalmente mensajes de respuesta.

Respecto a la tasa de servicios descubiertos y servicios falsos, tanto PDP, SLP como SSDP obtienen las mismas prestaciones, muy próximas al 100 % y al 0 %, respectivamente.

5.2.5. Soporte a aplicaciones de tipo "una petición–una respuesta"

En PDP hemos diferenciado entre estos dos tipos de peticiones de búsqueda, una "petición–una respuesta" y "una petición–varias respuestas", con el objetivo de minimizar el ancho de banda consumido, la idea es que si sabemos que a la aplicación que solicita el servicio le da igual qué servidor se lo ofrece, el protocolo se comporta de tal manera que sólo responde un servidor. Como se puede observar en la simulación realizada en la sección 4.4.7, la reducción obtenida en el número de mensajes transmitidos por búsqueda es muy significativa.

Ninguno de los protocolos estudiados realiza este tipo de distinciones a la hora de modificar el comportamiento del protocolo, por lo que no se realiza ninguna comparativa al respecto.

5.3. Conclusiones

En este capítulo realizamos un estudio detallado de las prestaciones de PDP en diferentes escenarios, e incluimos un estudio comparativo de PDP respecto a otros protocolos de descubrimiento de servicios.

En la sección 5.1, hemos analizado cómo se comporta el protocolo PDP dependiendo del tiempo de disponibilidad de los dispositivos, del número de dispositivos que existen en la red y del tamaño de su caché. Hemos visto que un parámetro decisivo en PDP es el tamaño de la caché, y que existe un tamaño óptimo en torno al número de servicios existentes en la red, a partir de este valor las mejoras obtenidas con PDP no son demasiado significativas y en cambio, esto nos lleva a que se consuma un mayor número de recursos en los dispositivos.

En la sección 5.2, hemos realizamos la comparativa de PDP respecto a otros protocolos tanto teóricos (*push*, *pull* y directorio) como de Internet (SLP y SSDP), en escenarios homogéneos, heterogéneos y con un dispositivo fijo. Hemos visto en todos los casos, que con PDP se obtiene el menor número de mensajes por petición, manteniendo las mismas tasas de descubrimiento de servicios y servicios falsos que los demás protocolos. En [Jones et al., 2001], se proporcionan una serie de valores de consumo energético de una tarjeta de red en modo transmisión y recepción, en general, en modo transmisión se consume más que en modo recepción, llegando en algunas tarjetas hasta el doble de diferencia. Por lo tanto, con la reducción en el número de mensajes por petición, unido al hecho de que en PDP los dispositivos con mayor tiempo de disponibilidad transmiten el mayor número de respuestas, nos llevan a concluir que PDP es el protocolo, de los estudiados, que realiza un menor consumo energético en los dispositivos que tienen mayores restricciones, alcanzando así uno de los objetivos principales planteados al diseñar este nuevo protocolo.

En esta última sección también hemos realizado un estudio de las prestaciones de PDP respecto a SLP y a SSDP, en el soporte a aplicaciones tipo "buscador". El resultado obtenido es muy favorable a PDP, ya que mantiene los mismos resultados que cuando se solicita un tipo concreto de servicio, mientras que en los otros dos protocolos, el valor del número de mensajes por búsqueda aumenta proporcionalmente al número de dispositivos existentes en la red. También hemos destacado el soporte de PDP a aplicaciones del tipo "una petición–una respuesta",

que no existe en los demás protocolos estudiados y que permite una reducción importante en el ancho de banda consumido al realizar este tipo de búsquedas.

A través del estudio realizado en este capítulo también hemos visto que se abren dos nuevas líneas de investigación futuras relacionadas con esta tesis, por una parte, la introducción de mecanismos dinámicos que nos permitan adaptar el tamaño de la caché al número de servicios existentes en la red, y por otra, el estudio de nuevos mecanismos para el cálculo de retardo en las respuestas dependiendo del tiempo de disponibilidad de los dispositivos, que nos permite reducir el consumo energético de los dispositivos más limitados. Además este último mecanismo podrá incluirse como mejora en otros protocolos de descubrimiento de servicios distribuidos.

Capítulo 6

Service Discovery Agent: servicio de páginas amarillas para agentes en entornos ubicuos

El primer objetivo con el que se comenzó la realización de esta tesis doctoral era contribuir a la adaptación del paradigma de agentes, para su uso como middleware de desarrollo de servicios en entornos de computación ubicua. Además, la propuesta debía ser conforme con los estándares de facto en agentes, los estándares de FIPA, o limitar al máximo los cambios que pudiesen tener en ellos.

El principal reto que se nos plantea para alcanzar este objetivo, es adaptar las plataformas de agentes para que se puedan ejecutar en los nuevos dispositivos con limitaciones de memoria y procesamiento, que se comunican mediante protocolos inalámbricos y que pueden formar una red sin necesidad de infraestructura. Estos retos se resumen en los ámbitos de investigación bajo el título de adaptación del paradigma de agentes a entornos ad-hoc.

Prueba de la importancia que está teniendo este tema en la investigación sobre agentes, es que el propio FIPA a principios de 2002 creó un comité técnico para la realización de especificaciones que aportasen soluciones en este ámbito, este comité se denomina FIPA Ad-Hoc[1], y la autora de esta tesis es miembro activo del mismo.

Al igual que el trabajo del FIPA Ad-Hoc en estos últimos dos años, nuestro trabajo para adaptar las plataformas de agentes a entornos ad-hoc, se ha centrado en proporcionar una solución a la búsqueda eficiente de servicios proporcionados por agentes remotos, en terminología de agentes: la adaptación del

[1]http://www.fipa.org/activities/ad_hoc.html

servicio de páginas amarillas. El análisis de este problema nos llevó a investigar la validez de los protocolos de descubrimiento de servicios existentes en la literatura. Debido a las carencias encontradas en ellos para su aplicación directa en los escenarios ad-hoc, propusimos un nuevo protocolo, PDP, pasando a ser éste una de las principales contribuciones de esta tesis. El análisis, motivación, definición y evaluación de este protocolo han sido abordados con detalle en los capítulos 4 y 5. En este capítulo, basándonos en estos resultados, realizamos una propuesta sobre la adaptación de la funcionalidad del servicio de páginas amarillas de FIPA, denominado Directory Facilitator, a entornos ad-hoc.

La estructura del capítulo es la siguiente, en la sección 6.1, justificamos cuál es la motivación de centrarnos en proporcionar una solución alternativa al servicio de páginas amarillas existente actualmente en FIPA, y describimos algunos de los trabajos de investigación relacionados con la adaptación de FIPA a entornos móviles. A continuación, sección 6.2, enumeramos los objetivos que nos hemos impuesto a la hora de proponer una nueva solución al problema. En la sección 6.3, analizamos varias soluciones alternativas e indicamos cuál es la que proponemos, solución que formalizamos en la sección 6.4. En la sección 6.5, comparamos nuestra propuesta con las especificaciones preliminares del comité técnico FIPA Ad-Hoc. Antes de finalizar con las conclusiones, en la sección 6.7, realizamos algunas consideraciones sobre la viabilidad de implementar una plataforma de agentes en dispositivos limitados reales y los trabajos que hemos llevado a cabo en esta línea, sección 6.6.

6.1. Motivación

Los agentes se definen como entidades autónomas, que se comportan según los objetivos que deben alcanzar, reaccionan ante eventos externos y pueden comunicarse y colaborar con otros agentes. Además si son móviles, pueden migrar entre dos nodos de una red, por lo que se adaptan mejor que las aplicaciones clásicas cliente/servidor, cuando las redes tienen conectividad intermitente y calidad cambiante, como es el caso de las redes inalámbricas [Glitho y Pierre, 2002].

La validez de aplicar la tecnología de agentes a la computación móvil se ha demostrado estos últimos años en varios proyectos europeos, *Lightweight Extensible Agent Platform*[2] (LEAP) y *Creation of User-friendly Mobile services Personalised for Tourism*[3] (CRUMPET) y en trabajos realizados en algunas universidades

[2]http://leap.crm-paris.com/

[3]http://www.ist-crumpet.org/

norteamericanas, destacando los del grupo de investigación Ebiquity[4] de la Universidad de Maryland, Baltimore County (UMBC). El propio FIPA, a través de varios comités técnicos y grupos de trabajo, realizó estudios en este campo, y sus resultados se reflejan en una serie de especificaciones que abordan principalmente los retos relacionados con el acceso de usuarios móviles a Internet mediante protocolos inalámbricos, es decir, los problemas de variaciones en la calidad, el retardo, la latencia, las tasas de error,... dependiendo de la localización del usuario; y las limitaciones de procesamiento y display del dispositivo que emplean para su acceso, normalmente teléfonos móviles y agendas electrónicas. A continuación, resumimos brevemente las principales aportaciones de estas especificaciones:

- *FIPA Nomadic Application Support Specification* [FIPA00014, 2002]: en esta especificación se añaden dos elementos nuevos al modelo de referencia de plataforma de FIPA, y que sirven como middleware de soporte a aplicaciones móviles:

  - *Monitor Agent* (MA), que se encarga de recolectar información sobre la calidad de servicio del *Message Transport Protocol* (MTP) empleado para el transporte de mensajes.
  - *Control Agent* (CA), que se encarga de gestionar la conexión establecida, *Message Transport Connection* (MTC) y que a partir de los datos proporcionados por el MA, selecciona el MTP más adecuado en cada momento.

- *FIPA Device Ontology Specification* [FIPA00091, 2001]: en esta especificación se define una ontología para expresar las capacidades, tanto hardware como software, de los dispositivos.

- *FIPA Message Buffering Service Specification* [FIPA00092, 2001]: en esta especificación se describe un mecanismo para que una plataforma de agentes pueda realizar *buffering* de mensajes ACL cuando existen pérdidas de conectividad. Aunque se definió para comunicaciones inalámbricas, también puede ser empleado en redes fijas.

- *FIPA Messaging Interoperability Services Specification* [FIPA00093, 2001]: en esta especificación se define un nuevo servicio que realiza conversiones entre distintos MTPs y entre distintos tipos de esquemas de codificación de mensajes ACL. Se emplea para poder comunicar agentes que residen en plataformas que soportan distintos tipos de MTP.

[4] http://research.ebiquity.org/

Como indicamos en el capítulo 1, la computación móvil dio paso a lo que se denomina computación ubicua, cuando los dispositivos con capacidad de cómputo y comunicación no eran sólo dispositivos personales, sino casi cualquier dispositivo físico que rodea al usuario. Esto, junto al desarrollo de nuevos protocolos de comunicación de corto alcance, permite que estos nuevos dispositivos, de forma espontánea, formen una red sin ningún tipo de infraestructura y se puedan componer sus servicios para ofrecer a los usuarios nuevas aplicaciones de forma transparente.

Las soluciones proporcionadas a nivel agentes para computación móvil, no abordan alguno de los nuevos retos que la computación ubicua plantea. En computación móvil, las redes son inalámbricas pero con infraestructura, típicamente redes de telefonía móvil, y las soluciones proporcionadas tienen en cuenta este hecho, así los dispositivos móviles no poseen una plataforma de agentes completa, sino que parte de sus servicios básicos residen en elementos situados en la parte fija de la red, y sin ellos no pueden interaccionar de forma directa con agentes residentes en otros dispositivos remotos.

En computación ubicua es necesario que los dispositivos tengan plataformas completas de manera que puedan actuar de manera autónoma, sin necesidad de acceder a una red con infraestructura que no siempre estará a su alcance, por conectividad o coste, y además, que no precisan para acceder a los servicios que ofrece el entorno más cercano con el que se puede comunicar de forma directa.

La validez del uso del paradigma de agentes en computación ubicua está avalada por el hecho de que los agentes, por definición, se adaptan a las limitaciones de estos entornos, en los que necesitamos:

- aplicaciones proactivas y autónomas que sean capaces de alcanzar los objetivos que quiere el usuario, sin necesidad de interaccionar continuamente con él;

- aplicaciones que cooperan con otros sistemas para poder obtener información o ejecutar tareas que las limitaciones del dispositivo no les permiten realizar de forma local;

- aplicaciones móviles para poder ejecutarse en los sistemas que tengan la información localmente, de forma que se minimice el número de transmisiones, se compensen las limitaciones de los protocolos inalámbricos y no se dependa de protocolos de encaminamiento multisalto.

  Además en entornos tan cambiantes, el usuario no podrá tener en su dispositivo personal toda la variedad de aplicaciones que puede necesitar para

interactuar con el entorno, por ello, será el propio entorno el que le proporcione bajo demanda estas aplicaciones en forma de código móvil.

Uno de los principales retos que supone la adaptación del paradigma de agentes a entornos ubicuos, es la adaptación de las plataformas de agentes a dispositivos limitados. Este reto se debe abordar desde dos vertientes, por una parte su diseño genérico y adaptado al modelo de referencia de FIPA [FIPA00023, 2002] y por otra, el análisis de la viabilidad de realizar su implementación en dispositivos reales, que por los motivos explicados en la sección 2.4.5, es ya abordable en la actualidad gracias a la existencia de J2ME.

6.1.1. Modelo de referencia FIPA para plataformas de agentes

Una plataforma de agentes debe proporcionar la infraestructura básica para ejecutar y gestionar agentes, FIPA no especifica el diseño interno de la plataforma, sino que sus estándares se centran en garantizar la interoperabilidad entre plataformas. Sin embargo, para alcanzar este objetivo han definido una serie de servicios básicos que deben existir en todas las plataformas de agentes, y que se conoce como modelo de referencia FIPA, ver Figura 6.1, que se compone de tres elementos:

- **Agent Management System** (AMS): que gestiona el ciclo de vida de los agentes, los recursos locales de la plataforma, y los canales de comunicación. Proporciona además, un servicio de páginas blancas, que permite localizar agentes por su nombre.
- **Directory Facilitator** (DF) : que proporciona un servicio de páginas amarillas, que permite localizar agentes por sus capacidades (servicios que ofrecen) y no por su nombre.
- **Message Transport System** (MTS): que gestiona el envío de mensajes entre agentes de la misma plataforma o de plataformas distintas, y permite la migración de agentes.

En las especificaciones actuales, los tres componentes son obligatorios, aunque como explicaremos en la sección 6.5 por petición del comité FIPA Ad-Hoc, el DF ha pasado a ser opcional, pero a día de hoy, esto todavía no se ha actualizado en las especificaciones FIPA.

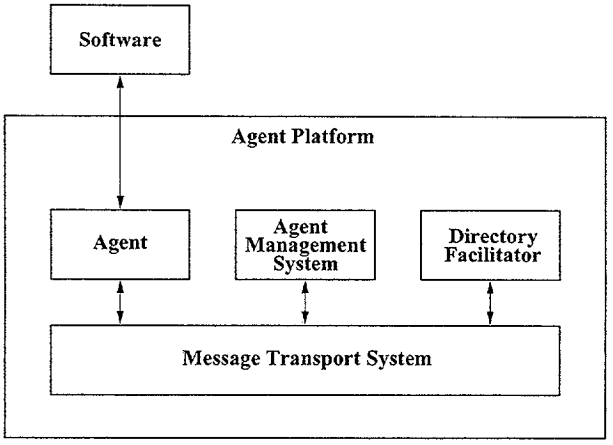


Figura 6.1: Modelo de referencia FIPA.

Los servicios que ofrecen estos elementos son imprescindibles dentro de una plataforma de agentes, pero debemos mejorar su diseño para que sea viable implementarlos en dispositivos limitados y para que se adapten a las características de las redes ad-hoc. En esta tesis doctoral, hemos centrado nuestra investigación en la adaptación del servicio de páginas amarillas, DF, el motivo principal de esta elección es que el propio FIPA Ad-Hoc quería analizar primero este problema, que además es el que conlleva mayores retos porque la definición de mecanismos de descubrimiento de servicios genéricos en una red ad-hoc, es también una de las líneas de investigación más importante en el ámbito de definición de protocolos para redes ad-hoc, como hemos visto en capítulos anteriores.

En cuanto a la funcionalidad del AMS y el MTS, en la sección 6.6 realizamos un breve análisis de cómo se debe abordar su adaptación y en la sección 6.7 cuáles son las posibles líneas futuras en las que se debe seguir trabajando.

6.1.2. Federación de DFs en entornos ad-hoc

El DF en FIPA permite a un agente buscar otros agentes, residentes en su misma plataforma o en otra remota, que proporcionan un determinado servicio. Los agentes que quieren ser localizados por otros agentes, deben registrar sus servicios en algún DF[5] de su plataforma. Una vez registrados, pueden eliminar la información que mantiene el DF sobre ellos y del mismo modo, también pueden modificar los datos asociados al registro realizado.

Por lo tanto, un DF mantiene una lista de los servicios ofrecidos por los

[5]FIPA permite que existan varios DFs en una misma plataforma

agentes residentes en la plataforma y cuando alguien solicita la búsqueda de un servicio, el DF la realiza sobre esta lista local. Para buscar agentes remotos, FIPA ha definido un mecanismo que se denomina federación de DFs. La federación de DFs consiste en que un DF además de tener registrados agentes residentes en su misma plataforma, también puede tener registrados DFs de plataformas remotas, y extender a éstos la búsqueda de un determinado servicio, y así sucesivamente. FIPA especifica una serie de restricciones en la realización de estas búsquedas:

- Profundidad de la búsqueda, parámetro `max-depth`, que indica cuántos saltos a DF remotos pueden ser realizarse antes de devolverle un resultado al agente que solicitó la búsqueda. Su valor por defecto es 0, que indica que sólo se buscan agentes locales. Un valor negativo indica que se puede propagar a todos los DFs posibles.

- Número máximo de resultados, parámetro `max-results`, que indica el número máximo de respuestas a una búsqueda. Su valor por defecto es 1 y un valor negativo indica que el agente acepta todas las respuestas posibles.

La federación de DFs es un mecanismo válido en redes fijas, y está siendo empleado en las plataformas de agentes compatibles con FIPA desarrolladas hasta la actualidad. La pregunta que nos debemos plantear ahora es, si este mecanismo sigue siendo válido para la búsqueda de servicios en redes ad-hoc. Desde nuestro punto de vista, la respuesta es que no es válido, fundamentalmente por dos razones:

- La búsqueda de un servicio remoto concreto siempre implica una comunicación con este sistema, porque a priori sólo conocemos que en esa plataforma existe un DF y no los servicios que están registrados en él. Por lo tanto, siempre es necesario realizar una consulta a los DFs remotos para saber si existe el servicio en la plataforma correspondiente, lo que implica realizar transmisiones sin garantía de éxito.

- Debido al carácter dinámico de los entornos, es necesario introducir mecanismos para que las federaciones de DFs se realicen de forma dinámica, es decir, se descubran las plataformas y automáticamente se federen los DFs de cada una de ellas entre sí. Por lo tanto, deben definirse nuevos mecanismos para el descubrimiento de plataformas y por lo tanto de DFs, cuyo coste es similar al descubrimiento directo de agentes remotos.

Por lo tanto, consideramos que es necesario definir un nuevo mecanismo de descubrimiento de servicios ofrecidos por agentes situados en plataformas remo-

tas. En la siguiente sección enumeramos los objetivos que debe cumplir, desde nuestro punto de vista, una solución a este problema.

6.2. Objetivos

Los objetivos que nos hemos planteado a la hora de proporcionar una solución al servicio de páginas amarillas en plataformas de agentes que operan en redes ad-hoc, son los siguientes:

- **Mantener el DF como servicio de páginas amarillas**

  El servicio que proporciona el DF, desde el punto de vista de su funcionalidad, sigue siendo imprescindible en entornos ad-hoc. Por lo tanto, nuestra propuesta mantendrá como servicio de páginas amarillas de los agentes, el DF, lo que modificará es la forma en la que el DF realiza las búsquedas de servicios en plataformas remotas.

  El DF debe mantener, por lo tanto, las mismas funciones que se le atribuyen en la especificación [FIPA00023, 2002], de tal forma que los agentes no necesitan modificar la manera en la que interaccionan con él, ni tienen que ser conscientes si se encuentran en una red con infraestructura o una red ad-hoc para poder descubrir servicios.

- **Eliminar la federación de DFs en entornos ad-hoc**

  El DF debe proporcionar mecanismos de búsqueda de servicios flexibles que abarquen tanto los servicios locales a la plataforma como los remotos. Para ello no empleará mecanismos de federación, sino que se basará en otros sistemas de descubrimiento que permitan localizar directamente los servicios ofrecidos por agentes remotos.

- **Emplear un mecanismo adaptado a las características de los entornos ad-hoc**

  El mecanismo de descubrimiento empleado debe adaptarse a las características del entorno en el que se va a aplicar, por lo tanto, se debe tener en cuenta los objetivos que ya mencionamos como requisitos para la definición del protocolo PDP, ver capítulo 4, es decir:

  - Minimizar el número de transmisiones, porque los dispositivos que forman la red pueden funcionar con baterías y por lo tanto, la reducción del consumo energético es prioritaria.

- Funcionar sin infraestructura fija, porque en una red ad-hoc, por definición, los dispositivos entran y salen de forma espontánea, y el funcionamiento de la red no puede depender de ningún elemento central.
- Adaptarse tanto a entornos dinámicos como estáticos, para que funcione de manera eficiente en redes ad-hoc, pero que se pueda emplear también en redes fijas.
- Simple y poco costoso computacionalmente, para que pueda implementarse sin consumir un gran número de recursos.

6.3. Posibles soluciones

Teniendo en cuenta el problema que pretendemos abordar a nivel agentes y el análisis realizado de protocolos de descubrimiento de servicios en los capítulos 4 y 5, vamos a analizar varias soluciones posibles, basadas algunas de ellas en el algoritmo de descubrimiento propuesto en esta tesis doctoral, PDP.

El primer objetivo que nos planteamos es mantener el DF como elemento obligatorio de una plataforma de agentes, esto nos lleva a introducir un nuevo elemento funcional en la plataforma que permitirá el descubrimiento de servicios remotos en entornos ad-hoc y que el DF utilizará como mecanismo alternativo a la federación. Este nuevo elemento lo hemos denominado **Service Discovery Agent**, a partir de ahora **SDA**.

En los siguientes apartados describiremos dos tipos de soluciones, una en la que se define un mecanismo de descubrimiento a nivel agentes que implementan internamente los SDA y otra, en la que el SDA interacciona con una implementación subyacente de un mecanismo de descubrimiento.

6.3.1. SDA implementa su propio mecanismos de descubrimiento

La primera opción que nos hemos planteado ha sido que el nuevo elemento, el SDA, se comporte según el algoritmo PDP. Para ello, un SDA implementaría las funciones de "agente de usuario" y "agente de servicio" PDP[6]. Cada SDA se comunicaría con los SDAs residentes en las plataformas de agentes remotas, para descubrir en ellas agentes que proporcionen los servicios solicitados.

[6]Recordar que en este caso la palabra "agente" es una manera de nombrar los módulos de los que se compone PDP

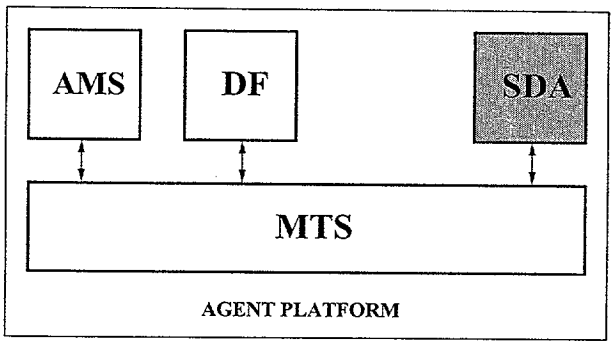


Figura 6.2: Primera propuesta de arquitectura para FIPA Ad-hoc.

Los SDAs son agentes y su comunicación debe estar basada en mensajes ACL, por lo tanto, se deberían especificar mensajes ACL equivalentes a los diferentes mensajes definidos en PDP. Además, al igual que los mensajes en PDP, se deberían transmitir por difusión, algo que, inicialmente, la semántica de ACL no contempla.

La interacción entre DF y SDA sería la siguiente (ver esquema en el Cuadro 6.1):

- Cuando un agente se registra en el DF, éste además de almacenar este registro, debe solicitar que esta descripción se registre también en el SDA, que internamente se manejará como un servicio local.

  Los agentes registran `df-agent-description`, definidas en [FIPA00023, 2002], en las que se incluye, junto información especifica del agente, un tiempo de vida asociado al registro, que se denomina `lease-time`. El SDA ajustará el valor de `lease-time` al mínimo entre el indicado por el agente y el tiempo de disponibilidad asociado al dispositivo. La modificación del valor `lease-time` debe comunicarse al DF, y éste, a su vez, se lo comunicará al agente para que, si procede, cuando este tiempo se agote realice un nuevo registro.

- Cuando un agente modifica un registro en el DF, éste debe realizar esta misma operación sobre el SDA, que internamente modificará la descripción del servicio y enviará un mensaje ACL equivalente a un `PDP_Service_Reply` gratuito[7] que incluye la nueva descripción del servicio, de forma que los SDAs remotos que tienen almacenado en su caché este servicio puedan actualizarlo.

[7]No se produce como respuesta a un mensaje de búsqueda

- Cuando un agente elimina un registro en el DF, éste debe realizar esta misma operación sobre el SDA, que internamente eliminará este servicio de sus servicios locales y enviará un mensaje ACL equivalente a un `PDP_Service_Deregister`, que permitirá que los SDAs remotos eliminen este servicio de sus cachés.

- Cuando un agente solicita una búsqueda al DF, éste debe comprobar los criterios de búsqueda y si está permitida la búsqueda de servicios en plataformas remotas ($\texttt{max} - \texttt{depth} \neq 0$), se solicita esta búsqueda al SDA que la realiza siguiendo el algoritmo PDP.

  Si $\texttt{max} - \texttt{results} = 1$, el SDA enviará un mensaje ACL equivalente a un `PDP_Service_Request (1/1)`. Si $\texttt{max} - \texttt{depth} > 1$, el SDA enviará un mensaje ACL equivalente a un `PDP_Service_Request (1/n)` y responderá al DF con tantas descripciones del tipo `df-agent-description` como indique el valor de `max-results`. Si $\texttt{max} - \texttt{depth} < 1$, el SDA enviará un mensaje ACL equivalente a un `PDP_Service_Request (1/n)` y responderá al DF con todas las descripciones del tipo `df-agent-description` como ha obtenido.

El principal inconveniente de esta solución es la necesidad de definir mensajes ACL de difusión. En el FIPA Ad-Hoc se llegaron a discutir este tipo de implementaciones, incluso Siemens [Berger et al., 2003] llegó a realizar una implementación basada en el protocolo de descubrimiento de servicios SSDP, pero se desestimó por prestaciones, el envío de mensajes ACL por difusión introducía una gran carga computacional en la plataforma.

6.3.2. SDA emplea un mecanismo de descubrimiento existente

Otra alternativa posible es que el SDA emplee un mecanismo de descubrimiento de servicios genérico para descubrir los servicios ofrecidos por agentes en la red ad-hoc. En este sentido deberá seleccionarse un mecanismo que se adapte a la restricciones expuestas en el sección 6.3, para garantizar una solución eficiente en redes ad-hoc.

Esta posible solución la dividimos a su vez en dos, la primera que emplea cualquiera de los mecanismos de descubrimiento de servicios de los existentes en la actualidad y la segunda, que emplea el protocolo que hemos propuesto en esta tesis doctoral, PDP.

Función DF	DF	Función SDA	SDA
register	Registra `df-agent-description`.	register	Registra `df-agent-description` como servicio local y modifica `lease-time`.
modify	Modifica `df-agent-description`.	modify	Modifica `df-agent-description` y envía mensaje ACL equivalente a `PDP_Service_Reply`.
deregister	Elimina `df-agent-description`.	deregister	Elimina `df-agent-description` y envía mensaje ACL equivalente a `PDP_Service_Deregister`.
search	Si $\texttt{max} - \texttt{depth} = 0$ busca en sus registros.		
	Si $\texttt{max} - \texttt{depth} \neq 0$	search	Si $\texttt{max} - \texttt{results} = 1$ transmite mensaje ACL equivalente a `PDP_Service_Request (1/1)`. Si $\texttt{max} - \texttt{results} > 1$ transmite mensaje ACL equivalente a `PDP_Service_Request (1/n)`. Responde al DF con tantas descripciones como se indique en `max-results`. Si $\texttt{max} - \texttt{results} < 1$ transmite mensaje ACL equivalente a `PDP_Service_Request (1/n)`. Responde al DF con todas las descripciones obtenidas.

Cuadro 6.1: Esquema de correspondencias entre las funciones del DF y el SDA (primera propuesta).

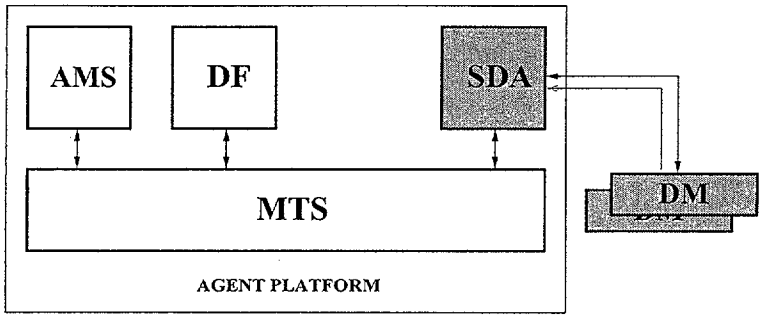


Figura 6.3: Segunda propuesta de arquitectura para FIPA Ad-hoc.

SDA emplea cualquier mecanismo de descubrimiento

El SDA para realizar búsquedas de servicios remotos ofrecidos por agentes emplea un mecanismo de descubrimiento existente. Este mecanismo de descubrimiento debe verificar al menos las siguientes condiciones:

- Ofrecer la misma funcionalidad al SDA que el DF ofrece a los agentes, es decir, registrar, eliminar registros, modificar registros y realizar búsquedas de servicios ofrecidos por agentes.
- Permitir integrar descripciones de servicios ofrecidas por agentes, `df-agent-description`, en el motor de búsquedas empleado.

El primer requisito no es demasiado restrictivo, porque las funcionalidades requeridas son las básicas que ofrece cualquier mecanismo de descubrimiento de servicios. El segundo es abordable, pero introduce una mayor complejidad en la solución, debido a que muchos mecanismos definen su propio formato de descripción de servicios, y por lo tanto, es necesario realizar correspondencias entre las descripciones a nivel agentes (`df-agent-description`) y el formato correspondiente del mecanismo de descubrimiento.

Esta correspondencia entre descripciones la llevaría a cabo el SDA y para cada mecanismo que se quiera integrar debe especificarse un procedimiento concreto, esto puede suponer en algunos casos una carga computacional elevada. Además, si permitimos que el propio SDA pueda emplear varios mecanismos de descubrimiento para realizar las búsquedas que le solicita el DF, esta solución sería muy compleja y no sería realista aplicarla en un entorno en el que existen dispositivos con limitaciones de memoria y procesamiento importantes.

Otro problema a tener en cuenta, es que si el SDA realiza búsquedas a través de un mecanismo de descubrimiento concreto, sólo se descubren agentes que están en

plataformas en las que el SDA local pueda utilizar ese mismo mecanismo. Como indicamos en la sección 2.3.7, existen pasarelas entre mecanismos de descubrimiento que podrían emplearse proporcionando una solución similar a la realizada para plataformas de agentes que emplean distintos MTP [FIPA00093, 2001], pero como en ese caso, necesitaríamos pasarelas intermedias que introducirían todavía mayor complejidad en el sistema, y la red precisaría de cierta infraestructura para su funcionamiento.

Además, en el análisis realizado en el capítulo 4 hemos visto que no todos los mecanismos de descubrimiento son óptimos para su funcionamiento en redes ad-hoc. Una manera de mejorar las prestaciones, independientemente del sistema empleado, sería introducir una caché de agentes remotos en el SDA, de tal manera que si el DF solicita una búsqueda de un agente al SDA, éste busque inicialmente en esta caché y en caso de encontrarlo, responda con estas descripciones de agentes al DF y no reenvíe la búsqueda al sistema de descubrimiento subyacente. Las entradas de agentes remotos registrados en el SDA deben tener un tiempo de vida asociado, indicado en el parámetro `lease-time` de su `df-agent-description`. Además, para maximizar la consistencia de esta caché, debería permitirse que cuando un agente detecte que un agente descubierto no está disponible realmente, se lo indique al DF para que éste a su vez elimine este registro en el SDA, y se borre de su caché la entrada correspondiente.

Teniendo en cuenta otro de los mecanismos en los que se basó el diseño del PDP, el SDA puede tener asociado un tiempo de disponibilidad, dependiendo de las características del dispositivo, de manera que cada vez que registre un servicio local en el sistema de descubrimiento, modifique el `lease-time` de la descripción, para que su valor sea el mínimo entre el valor indicado por el agente y el tiempo de disponibilidad. Del mismo modo, cuando se almacena en la caché un agente remoto, debe modificar el tiempo de `lease-time` al mínimo de los dos valores. Estos cambios en el valor de `lease-time` deben ser notificados a los agentes, de manera que finalizado este tiempo, puedan volver a registrarse.

La interacción entre DF, SDA y DM[8] en este caso, incluyendo las mejoras propuestas de uso de cachés y tiempo de disponibilidad, sería la siguiente (ver esquema en el Cuadro 6.2):

- Cuando un agente se registra en el DF, éste además de almacenar este registro, debe solicitar que esta descripción se registre también en el SDA, que internamente lo registrará como un servicio local en el/los mecanismo/s de descubrimiento subyacente/s.

  Los agentes registran `df-agent-description` en la que se incluye, junto a

[8]Discovery Module.

Función DF	DF	Función SDA	SDA	DM
register	Registra `df-agent-description`.	register	Mapea `df-agent-description` al formato del DM	Registra el servicio como local.
modify	Modifica `df-agent-description`.	modify	Mapea `df-agent-description` al formato del DM.	Modifica la descripción del servicio local y si el sistema lo permite anuncia este cambio en la red.
deregister	Elimina `df-agent-description`.	deregister	Mapea `df-agent-description` al formato del DM.	Elimina servicio local y si el sistema lo permite anuncia este cambio en la red.
search	Si $\texttt{max} - \texttt{depth} = 0$ busca en sus registros.			
	Si $\texttt{max} - \texttt{depth} \neq 0$	search	Realiza el mapeo al formato DM. Mapea respuestas a `df-agent-description` y las almacena en caché.	Realiza la búsqueda solicitada.

Cuadro 6.2: Esquema de correspondencias entre las funciones del DF, el SDA y DM (segunda propuesta).

la información especifica del agente, un tiempo de vida asociado al registro, que se denomina `lease-time`. El SDA ajustará el valor de `lease-time` al mínimo entre el indicado por el agente y el tiempo de disponibilidad asociado al dispositivo. Posteriormente, realizará la correspondencia entre la descripción del agente y el formato específico del mecanismo de descubrimiento a emplear. La modificación del valor `lease-time` debe comunicarse al DF, y éste, a su vez, se lo comunicará al agente para que, si procede, cuando este tiempo se agote realice un nuevo registro.

- Cuando un agente modifica un registro en el DF, éste debe realizar esta misma operación sobre el SDA, que solicita a su vez esta modificación del correspondiente servicio local en el/los mecanismo/s de descubrimiento subyacente/s. Como en el caso anterior, debe realizarse la correspondencia entre descripciones de servicios.

- Cuando un agente elimina un registro en el DF, éste debe realizar esta misma operación sobre el SDA, que realizará la correspondencia entre descripciones, para solicitar esta misma operación al mecanismo/s de descubrimiento subyacente/s.

- Cuando un agente solicita una búsqueda al DF, éste debe comprobar los criterios de búsqueda y si está permitido propagarla a plataformas remotas ($\texttt{max} - \texttt{depth} \neq 0$), se solicita esta búsqueda al SDA, que realiza las siguientes pasos:

  - Consulta su caché de agentes remotos y si tiene descripciones que cumplan el criterio de búsqueda, responde con ellas al DF. El número de respuestas proporcionadas al DF se limitará al valor indicado por el parámetro `max-results`.
  - Si la búsqueda en la caché es infructuosa, entonces le solicita la realización de esta búsqueda al mecanismo/s de descubrimiento subyacente/s, realizando previamente la correspondencia entre descripciones de servicios necesaria.
  - Si el mecanismo de descubrimiento le devuelve alguna descripción de servicio, debe volver a realizar la correspondencia para convertirla en una descripción de agentes y almacenarla en su caché para posteriores búsquedas, actualizando el `lease-time` al mínimo entre el indicado por el sistema de descubrimiento y el tiempo de disponibilidad asociado al dispositivo. Finalmente, responderá al DF con tantas respuestas como indique el parámetro `max-results`.

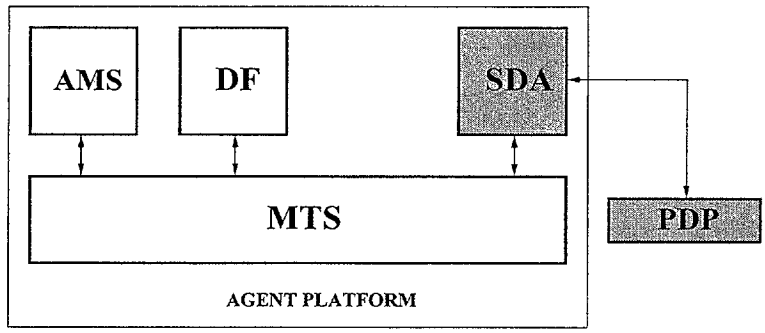


Figura 6.4: Tercera propuesta de arquitectura para FIPA Ad-hoc.

SDA emplea el Pervasive Discovery Protocol

Una solución tan genérica como la anterior, en la que el SDA pueda emplear cualquier sistema de descubrimiento de servicios, por los motivos explicados anteriormente, no cumple los objetivos que nos hemos impuesto. Así llegamos a la tercera solución, que es similar a la anterior pero estableciendo como sistema de descubrimiento el protocolo que hemos propuesto en esta tesis, PDP.

En esta solución la interacción entre el DF, el SDA y el PDP sería la siguiente (ver esquema en el Cuadro 6.3):

- Cuando un agente se registra en el DF, éste además de almacenar este registro, debe solicitar al SDA realizar esta misma operación. El SDA realizará una petición de registro del correspondiente `df-agent-description` al PDP_SA, que lo incluirá como un servicio local.

  Igual que en las propuestas anteriores, el SDA ajustará el valor del `lease-time`, incluido en `df-agent-description` proporcionado por el agente, al mínimo entre el indicado por el agente y el tiempo de disponibilidad asociado al dispositivo. La modificación del valor `lease-time` debe comunicarse al DF, y éste, a su vez, se lo comunicará al agente para que, si procede, cuando este tiempo se agote realice un nuevo registro.

- Cuando un agente modifica un registro en el DF, éste debe solicitar esta misma operación al SDA, que realizará una petición de modificación de la descripción del correspondiente servicio al PDP_SA. A nivel PDP, se enviará un mensaje `PDP_Service_Reply` gratuito, que incluye la nueva descripción del servicio, de forma que PDP_UAs remotos que tienen almacenado en su caché este servicio, lo actualicen.

- Cuando un agente elimina un registro en el DF, éste debe solicitar esta

Función DF	DF	Función SDA	SDA	PDP
register	Registra df-agent-description.	register	Solicita registro df-agent-description a PDP_SA.	Incluye el df-agent-description como servicio local.
modify	Modifica df-agent-description.	modify	Solicita modificación df-agent-description a PDP_SA	Modifica descripción del servicio local. Envía PDP_Service_Reply gratuito.
deregister	Elimina df-agent-description.	deregister	Solicita eliminar df-agent-description a PDP_SA.	Borra servicio local. Envía un PDP_Service_Deregister.
search	Si $max - depth = 0$ busca en sus registros.			
	Si $max - depth \neq 0$.	search	Si $max - results = 1$ solicita búsqueda tipo 1/1 al PDP_UA.	Envía PDP_Service_Request (1/1).
			Si $max - results > 1$ solicita búsqueda tipo 1/n al PDP_UA y responde con tantas descripciones como indica max-results. Si $max - results < 1$ solicita búsqueda tipo 1/n al PDD_UA y responde con todas las descripciones obtenidas.	Envía PDP_Service_Request (1/n).

Cuadro 6.3: Esquema de correspondencias entre las funciones de DF, SDA y PDP (tercera propuesta).

misma operación sobre el SDA, que realizará la petición correspondiente al PDP_SA, que eliminará este servicio de sus servicios locales y enviará un mensaje `PDP_Service_Deregister`, que permitirá que los PDP_UAs remotos eliminen este servicio de sus cachés.

- Cuando un agente busca un servicio a través del DF, éste debe comprobar los criterios de búsqueda y si está permitido extenderla a plataformas remotas ($\texttt{max} - \texttt{depth} \neq 0$), solicitar esta búsqueda al SDA, que la realiza a través del PDP_UA.

  Si la restricción de búsqueda, $\texttt{max} - \texttt{results} = 1$ se realiza una petición de búsqueda PDP del tipo `1/1`. En cambio, si $\texttt{max} - \texttt{results} > 1$ se realiza un petición de búsqueda PDP del tipo `1/n` y en este caso, aunque a nivel PDP se obtengan más respuestas, es el SDA el que al responder al DF, las limita al valor indicado en `max-results`. Sólo si $\texttt{max} - \texttt{results} < 1$ se responde con todas las respuestas obtenidas.

Desde nuestro punto de vista, esta solución es más eficiente para entornos ad-hoc que permitir que el SDA interaccione con cualquier mecanismo de descubrimiento de servicios, no sólo por la complejidad que introduce en la plataforma y que dificulta su implementación en dispositivos reales, sino también por los problemas de interoperabilidad entre plataformas que emplean distintos mecanismos de descubrimiento y que en cambio, comparten el mismo tipo de MTP y por lo tanto, pueden comunicarse pero no descubrir los servicios que ofrecen. Así, consideramos que es necesario establecer un mecanismo concreto y nosotros proponemos que sea PDP, decisión que se apoya en los resultados de prestaciones obtenidas para este protocolo respecto a otros sistemas de descubrimiento existentes en la literatura, descritos en el capítulo 5.

6.4. Service Discovery Agent

Como hemos explicado al final de la sección anterior, una vez evaluadas las diferentes opciones, hemos optado por la última de las tres propuestas. En esta sección, formalizamos la definición del Service Discovery Agent que emplea como mecanismos de descubrimiento el PDP, proporcionando una ontología para este agente y describiendo sus interacciones con el DF.

El **Service Discovery Agent (SDA)** es un nuevo agente, que proponemos introducir en el modelo de referencia FIPA para realizar búsquedas de servicios ofrecidos por agentes localizados en plataformas remotas y que operan en un

entorno ad-hoc. La funcionalidad que proporciona sustituye al mecanismo de federación de DFs en estos entornos.

6.4.1. Funcionalidad

El SDA es una concretización para el descubrimiento en redes ad-hoc del servicio de directorio de agentes genérico que se describe en [FIPA00001, 2002], *Agent Directory Service*. Proporciona la misma funcionalidad al DF, que éste a su vez proporciona a los agentes residentes en la plataforma, pero permite buscar de forma eficiente agentes que ofrecen un determinado servicio en plataformas remotas, localizadas en otros dispositivos que forman la red ad-hoc.

El SDA empleará una implementación del PDP para realizar estas búsquedas, esta implementación utilizará como formato de descripción de servicios el de agentes y su motor de búsquedas se basará en plantillas de descripción de agentes. El SDA solicitará al PDP_SA el registro, modificación y eliminación de registros de agentes, que han solicitado estas mismas operaciones al DF, y pedirá al PDP_UA realizar búsquedas en la red ad-hoc, siempre que las restricciones lo permitan. Si la búsqueda sólo se puede realizar localmente, será el propio DF el que la procese chequeando los registros de agentes locales.

Todas las funciones del SDA emplean para describir servicios ofrecidos por agentes, objetos del tipo `df-agent-description`, tal y como se definen en FIPA00023 como parte de la ontología de gestión de agentes *Agent Management Ontology*, cuyo identificador es `fipa-agent-management`. El SDA soporta funciones de registro (`register`), modificación (`modify`), eliminación (`deregister`) de registros y una función de búsqueda (`search`) de manera análoga a las funciones del DF.

6.4.2. Registro del SDA en el DF

El agente SDA proporciona un servicio que obligatoriamente tiene que estar registrado en el DF de la plataforma en la que reside, de esta forma en un DF que opere en una red ad-hoc siempre existirá una entrada correspondiente al SDA, que poseerá el siguiente AID (*Agent Identifier*) reservado:

```
(agent-identifier
  :name sda@hap[9]_name
  :addresses (sequence hap_transport_address))
```

[9] *Home Agent Platform*

Este agente se registra en el DF poniendo en el parámetro `:type` del `service-description` el valor `sda`.

El DF siempre que se realice una función de registro, modificación o borrado solicitará esta misma operación al SDA. Si la operación es de búsqueda, sólo se la reenviará si la restricción de búsqueda `max-depth` lo permite, es decir, si su valor es distinto de cero.

6.4.3. Ontología

El SDA es un agente y por lo tanto, deberá especificarse una ontología para establecer cómo se interacciona con él. Tal y como se ha planteado en la arquitectura propuesta, el DF es el único agente que interacciona con el SDA, y por lo tanto es el único con el que comparte esta ontología.

La ontología definida para el SDA la hemos denominado `service-agent-ontology`, y emplea parte de los objetos y funciones especificadas en la ontología `fipa-agent-management`, definida para el DF y el AMS. Notar que si el SDA se integrase en las especificaciones de FIPA, su ontología debería ser la misma que la del DF y la del AMS, ya que, al igual que ellos, es un agente que implementa un tipo concreto del servicio de directorio, especificado en la arquitectura abstracta de FIPA.

Descripción de objetos

El SDA reutiliza los siguientes objetos definidos en la ontología `fipa-agent-management` especificada en el FIPA00023:

- `df-agent-description`: que representa la descripción de un agente que se puede registrar en el SDA.
- `agent-identifier`: que representa la identificación de un agente. Éste es un parámetro de un objeto de la clase `df-agent-description` que identifica al agente que realiza el registro.
- `service-description`: que representa la descripción de cada servicio ofrecido por un agente. Un conjunto de `service-descriptions` se incluye como parámetro dentro de un objeto de la clase `df-agent-description` para indicar qué servicios registra el agente.

La ontología del SDA, `service-agent-ontology`, incluye además un nuevo objeto, `adhoc-search-constraints` que representa las restricciones que se imponen a la función de búsqueda del SDA.

Para describir este objeto, se emplea la misma terminología que la empleada para describir los objetos del dominio de la ontología `fipa-agent-management`, que es la siguiente:

- **Marco** (*Frame*): nombre obligatorio de esta entidad que debe ser empleado para representar cada instancia de la clase.
- **Ontología** (*Ontology*): nombre de la ontología, cuyo dominio de discurso incluye los parámetros descritos en el cuadro.
- **Parámetro** (*Parameter*): nombre obligatorio del parámetro de este marco.
- **Descripción** (*Description*): descripción en lenguaje natural del significado del parámetro.
- **Presencia** (*Presence*): indica si el parámetro es obligatorio u opcional.
- **Tipo** (*Type*): indica el tipo al que pertenecen los valores del parámetro: `Integer`, `Word`, `String`, `URL`, `Term`, `Set` o `Sequence`.
- **Valores reservados** (*Reserved values*): es una lista de constantes predefinidas que están reservadas como posibles valores del objeto.

El objeto `adhoc-search-constraints` es un tipo de objeto que representa el conjunto de restricciones que limitan la función de búsqueda en el SDA (ver Cuadro 6.4).

Notar que si el SDA se integrase en las especificaciones de FIPA, debería modificarse la descripción del parámetro `max-depth` del objeto `search-constraints` de la ontología `fipa-agent-management`. Su descripción en lenguaje natural en la especificación actual, dice que `max-depth` es "el parámetro que indica la profundidad de propagación de una búsqueda en directorios federados", debería reflejarse además que si existe un SDA registrado en el DF y el valor de $\texttt{max} - \texttt{depth} \neq 0$ se debe propagar la búsqueda al SDA.

Descripción de funciones

El SDA reutiliza las funciones `register`, `deregister` y `modify` tal y como se definen en la ontología `fipa-agent-management`. En la ontología definida para

Marco	adhoc-search-constraints			
Ontología	service-agent-ontology			
Parámetro	**Descripción**	**Presencia**	**Tipo**	**Valores reservados**
max-results	Indica el número de df-agent-descriptions que se le pueden devolver al DF como resultado de una búsqueda. Un valor negativo indica que se pueden proporcionar todos los resultados posibles. Su valor por defecto es 1	Opcional	integer	

Cuadro 6.4: Descripción de adhoc-search-constraints.

el SDA, sólo se define la función search, ya que emplea unas restricciones de búsqueda distintas.

Para describir esta función, se emplea la misma terminología que la empleada para describir las funciones del dominio de la ontología fipa-agent-management, que es la siguiente:

- **Función** (*Function*): símbolo que identifica la función en la ontología.
- **Ontología** (*Ontology*): nombre de la ontología, en este caso service-agent-ontology.
- **Soportado por** (*Supported by*): tipo de agente que soporta esta función.
- **Descripción** (*Description*): descripción en lenguaje natural de la semántica de la función.
- **Dominio** (*Domain*): indica el dominio sobre el que se define la función. Los argumentos que se le pasan deben pertenecer a este dominio.
- **Rango** (*Range*): indica el rango en el que la función traslada los símbolos del dominio. El resultado de la función debe pertenecer a este rango.
- **Grado** (*Arity*): indica el número de argumentos que toma la función. Si la función toma un número arbitrario de argumentos, el valor de este campo es indefinido.

Función	search
Ontología	service-agent-ontology
Soportado por	SDA
Descripción	El SDA busca en la red ad-hoc df-agent-descriptions según la plantilla de descripción de agentes proporcionada por el DF. Como resultado de esta función se obtienen un número menor o igual a max-results de df-agent-descriptions, y un resultado nulo en caso de que no se encuentre ninguna descripción de servicios que verifique la búsqueda. Internamente el SDA empleará la implementación del PDP para realizar estas búsquedas de forma eficiente.
Dominio	df-agent-description × adhoc-search-constraints (producto cartesiano)
Rango	Conjunto de objetos del tipo df-agent-description
Grado	2

Cuadro 6.5: Descripción de la función search.

En el Cuadro 6.5 se describe la función de búsqueda search definida en la ontología empleada por el SDA.

Excepciones

El patrón de interacciones entre DF y SDA sigue el protocolo de interacción fipa-request definido en [FIPA00026, 2002]. Bajo ciertas circunstancias se pueden producir algunas excepciones, como por ejemplo que el DF intente registrar un agente que ya ha sido registrado en el SDA.

En la ontología service-agent-ontology se reutilizan las excepciones definidas en [FIPA00023, 2002] para la ontología fipa-agent-management.

6.5. Contribuciones en el comité técnico FIPA Ad-Hoc

Siguiendo el procedimiento habitual de los comités técnicos en FIPA, el FIPA Ad-Hoc realizó en Febrero de 2002 un llamada abierta a toda la comunidad científica, solicitando propuestas para adaptar las plataformas de agentes FIPA

a entornos ad-hoc. Las propuestas debían abordar el problema que plantea tener plataformas de agentes en dispositivos que forman una red ad-hoc y por lo tanto, que son libres de moverse libremente, y de entrar y salir de ella de forma espontánea. Además, debían tener en cuenta las restricciones de memoria, procesamiento y baterías que caracterizan a los dispositivos móviles que forman habitualmente estas redes, como teléfonos móviles y agendas digitales.

Los objetivos principales de esta llamada eran intentar definir qué componentes de una plataforma de agentes FIPA debían residir en estos dispositivos, y cómo conseguir que se descubriesen entre si las plataformas para que los agentes residentes en cada una de ellas pudiesen compartir sus servicios y comunicarse.

Las propuestas realizadas se discutieron en la reunión FIPA 25, celebrada en Marzo de 2002. Los contenidos de estas contribuciones son de forma resumida los siguientes:

- Propuesta del grupo Ebiquity de la University Maryland, Baltimore County UMBC [Ratsimor et al., 2002]:

  Cada dispositivo móvil debía contener una versión reducida de los servicios básicos de una plataforma FIPA: DF y AMS. Además, proponían introducir una serie de nuevos servicios para permitir que las plataformas compartieran sus servicios:

  - Gestor de anuncios: responsable de anunciar mediante mensajes periódicos de broadcast los servicios ofrecidos por los agentes locales.
  - Gestor de caché: responsable de almacenar de forma local los servicios ofrecidos por agentes que residen en dispositivos remotos.
  - Gestor de reenvíos: responsable de reenviar peticiones de servicios de dispositivos vecinos, para que se propagasen a otros dispositivos de la red.
  - Gestor de políticas: responsable de controlar el comportamiento de la plataforma, por ejemplo, con qué frecuencia se anuncian los servicios locales, qué servicios se almacenan en la caché,...

- Propuesta de Siemens AG [Berger y Watzke, 2002]:

  Esta propuesta se centraba en cómo se descubren las plataformas de agentes en una red ad-hoc, para ello proponían un mecanismo de anuncio de plataformas empleando mensajes ACL enviados por difusión. Cuando dos

plataformas se descubren mediante este mecanismo, los DFs residentes en ambas se federan, de tal forma que los servicios ofrecidos por los agentes de cada una de las plataformas pueden descubrirse entre si. Cuando se detecte un error al intentar acceder a un servicio remoto, se elimina el registro correspondiente al DF de la plataforma donde reside ese servicio, y por lo tanto, se elimina la federación.

- Propuesta del Media Lab Europe [Lawrence, 2002]:

  En esta propuesta se plantea que los servicios de directorio, AMS y DF sean elementos opcionales en una plataforma de agentes en dispositivos limitados. Para suplir algunas de las funcionalidades proporcionadas por estos servicios, se propone la definición de un nuevo agente, denominado *Discovery Agent* (DA) que es el responsable de anunciar los servicios locales y de descubrir servicios remotos en ausencia de directorio.

  Si el DA descubre alguna plataforma remota que contenga un servicio de directorio, puede registrar los agentes de su plataforma local en él, estos registros se realizarán indicando un tiempo de vida para que no queden obsoletos. Si en la plataforma en la que reside el DA se activa un directorio, los agentes deben emplear el directorio para realizar búsquedas y para registrar servicios, el DA en este caso permitirá que plataformas remotas descubran este directorio.

  El DA puede manejar internamente varios protocolos de descubrimiento de servicios para realizar sus tareas. Las funciones que soportará este agente son similares a las del DF, pero se propone añadir algunas nuevas:

  - `register` y `deregister`: que permiten a los agentes registrar sus servicios en el DA o eliminarlos.
  - `subscribe` y `unsubscribe`: que permite a los agentes subscribirse a un determinado tipo de servicio, de manera que cuando se descubra en la red un agente que lo proporciona, el DA se lo notifique.
  - `get-directories`: que devuelve todos los directorios en los que el DA ha registrado a los agentes locales.
  - `get-advertisement`: que permite obtener todos los agentes localizados en una plataforma, si la plataforma tiene un servicio de directorio, se proporciona una referencia a éste.
  - `get-all-agent-descriptions`: que permite obtener todos los agentes localizaciones en una plataforma.

- Propuesta de Universidad Carlos III de Madrid [Campo et al., 2002b], que se incluye en el anexo C:

  Nuestra propuesta abordaba la adaptación del servicio de páginas amarillas, DF, para su funcionamiento en redes ad-hoc. Realizábamos un estudio detallado de las tecnologías existentes de descubrimiento de servicios en la literatura y proponíamos que siguiese existiendo un DF en cada plataforma, pero eliminábamos el mecanismo de federación. Estos DF deberían compartir los servicios registrados en ellos, implementando algún sistema de descubrimiento de servicios adaptado a redes ad-hoc. Proponiamos que este mecanismo fuese el Pervasive Discovery Protocol, y en el documento describíamos la primera versión de nuestro protocolo.

Después de esta primera discusión, el comité decidió centrar sus estudios en el problema del descubrimiento de servicios a nivel agentes, y con este objetivo se comenzó a redactar un *White Paper* titulado *Agents in Ad Hoc Environments* [WG-AdHoc, 2002], en el que aparece gran parte del análisis de los mecanismos de descubrimiento de servicios incluidos en nuestra propuesta. En este documento también se refleja la necesidad de eliminar el mecanismo de federación de DF, como habíamos propuesto.

El comité Ad-Hoc nos invitó a exponer nuestra propuesta en la siguiente reunión del FIPA, en Julio de 2002, para la que realizamos un nuevo documento en el que detallábamos más algunos de los aspectos de nuestra propuesta inicial. El contenido de este documento se incluye en el anexo D y en él se introduce un agente intermedio entre el DF y el protocolo de descubrimiento, denominado SDA, y se plantean las diferentes alternativas expuestas en la sección 6.3 de este capítulo.

A lo largo de este último año, el grupo ha seguido discutiendo sobre el servicio de páginas amarillas en entornos ad-hoc y ha tomado algunas decisiones. La primera de ellas, es solicitar al *FIPA Architecture Board*[10] que el DF no sea un elemento obligatorio de una plataforma compatible con FIPA, decisión que ha sido aceptada. El motivo es que el DF y el mecanismo de federación tal y como se define en FIPA00023 no son válidos en redes ad-hoc y por lo tanto, no existirá en las plataformas de agentes que operen en estos entornos. La segunda decisión, es que cualquier especificación propuesta por el comité debe ser lo suficientemente flexible, como para que se pueda emplear cualquier mecanismo de descubrimiento de servicios desarrollado para redes ad-hoc.

[10]Comité encargado de coordinar el trabajo de todos los comités y grupos de trabajo de FIPA

En Noviembre de 2003, se propone la primera versión de especificación FIPA, denominada *FIPA Agent Discovery Service Specification*. En esta especificación se define un nuevo agente, *Agent Discovery Service* (ADS), que sustituye la funcionalidad del DF en redes ad-hoc, manteniendo funciones equivalentes a las de `register`, `deregister` y `search` del DF e introduciendo dos nuevas: `subscribe`, que permiten a los agentes suscribirse en el ADS para recibir notificaciones cuando se descubren agentes que ofrecen un determinado servicio y `unsubscribe`, que permite eliminar esa suscripción. El ADS puede interaccionar con uno o varios mecanismos de descubrimiento, denominados en la especificación *Discovery Middleware* (DM). Si existe un DF en la plataforma, un agente debe emplear el ADS sólo para localizar servicios remotos ofrecidos en la red ad-hoc, los servicios locales se deben seguir buscando a través del DF. También se ha realizado una especificación, denominada *FIPA JXTA Discovery Middleware Specification*, que aborda cómo el ADS interacciona con el sistema de descubrimiento JXTA.

Nuestra propuesta difiere de la adoptada en el comité, fundamentalmente en dos aspectos:

- La eliminación del DF, que desde nuestro punto de vista presenta el inconveniente de que los agentes deben ser conscientes de si la plataforma en la que residen opera en una red ad-hoc o no.
- La posibilidad de que el ADS emplee cualquier mecanismo de descubrimiento, porque esto supone introducir una gran complejidad en el sistema, ya que el ADS debe realizar las correspondencias entre `df-agent-descriptions` y el formato de descripción de servicios empleado por el DM. Otro problema importante es el de la interoperabilidad entre plataformas que empleen distintos DMs.

  Desde nuestro punto de vista, aunque la especificación final permita utilizar cualquier DM, de facto en las implementaciones se establecerá un DM estándar y único.

Aunque algunas decisiones tomadas en el grupo difieren de nuestra solución al problema de la adaptación de los servicios de páginas amarillas en redes ad-hoc, la especificación final ha sido construida basándose en algunas de las ideas que defendíamos en nuestras contribuciones iniciales:

- Eliminar el mecanismo de federación de DF en plataformas que operan en redes ad-hoc.
- Realizar el descubrimiento a nivel agentes, no a nivel plataforma.

- Basar el descubrimiento de servicios remotos en un mecanismo de descubrimiento existente.

6.6. Viabilidad de implementación en dispositivos reales

A lo largo de este capítulo hemos realizado un estudio del modelo de referencia FIPA para adaptarlo a los requisitos que imponen los entornos ad-hoc. Esta tarea es fundamental para proporcionar unas especificaciones que permitan la interoperabilidad de plataformas de agentes en estos entornos, y de esta forma poder desarrollar servicios ubicuos basados en el paradigma de agentes. Pero el reto que se plantea para que esto se haga realidad no es sólo definir este estándar de interoperabilidad entre plataformas, sino también poder implementar una plataforma de agentes en dispositivos limitados con la tecnología existente en la actualidad. En esta sección realizamos un análisis de este segundo reto y describimos algunos de los trabajos que hemos realizado en esta línea.

Como comentábamos en el capítulo 2, a día de hoy podemos decir que aunque se han definido lenguajes específicos, Java ha sido el lenguaje en el que más plataformas de agentes se han desarrollado, debido a las siguientes ventajas: independencia de la plataforma, ejecución segura y serialización. Además junto a otras características como la carga dinámica de clases y la reflexión, que facilitan el soporte a la movilidad de los agentes.

Con la definición en 1999 de J2ME, versión de Java para dispositivos limitados, y la aparición cada vez más creciente en el mercado de teléfonos móviles y agendas digitales, que implementan el perfil MIDP de J2ME, la tarea de implementar una plataforma de agentes en estos dispositivos pasó a ser más fácil de abordar. De hecho, la plataforma desarrollada en el proyecto LEAP fue la primera en integrar dispositivos MIDP como parte de un sistema de agentes.

La implementación realizada en LEAP está basada en la plataforma de agentes J2SE JADE[11] y en la actualidad se distribuye como una parte integrada dentro de ésta. En LEAP, los dispositivos MIDP tienen soporte para la ejecución y comunicación de los agentes que residen en el dispositivo, pero su funcionamiento está asociado a una plataforma de agentes situada en un dispositivo tipo PC, en terminología de JADE, un contenedor principal. En el dispositivo MIDP no existe por lo tanto, una plataforma de agentes compatible con FIPA, ya que el AMS y DF del que dependen sus agentes, están en el contenedor principal.

[11] http://sharon.cselt.it/projects/jade/

Además de LEAP, han aparecido otra serie de plataformas de agentes compatibles con FIPA que se pueden ejecutar en agendas electrónicas, que han sido desarrolladas para la versión Java denominada Personal Java, como son [Micro FIPA-OS, 2001] y Grasshopper MicroEdition [Grasshopper MicroEdition, 2001]. Las ventajas de realizar la implementación en esta versión, es que soporta casi todas las características de J2SE, al contrario que el perfil MIDP de J2ME. El inconveniente es que los tipos dispositivos en los que se puede ejecutar se reduce a agendas digitales de altas prestaciones y no a cualquier dispositivo móvil.

La dificultad de desarrollar plataformas de agentes en dispositivos MIDP, es que parte de las características que convertían a Java en un buen lenguaje de programación de plataformas de agentes han desaparecido, o se han visto limitadas por motivos de seguridad, en concreto, no se da soporte ni a la carga dinámica de clases, ni a la serialización, ni a la reflexión. Además, como MIDP es un perfil diseñado para dispositivos limitados, su soporte a la comunicación no está pensado para que dos dispositivos se puedan comunicar de forma directa, requisito básico si las plataformas de agentes operan en redes ad-hoc.

6.6.1. Implementación de nuestra propuesta para FIPA Ad-Hoc

LEAP se desarrolló para desplegar servicios basados en agentes en una red de telefonía móvil, por lo tanto, una red con infraestructura. La limitación a nivel MIDP que necesitaron suplir fue la serialización, para que los agentes que se ejecutan en el dispositivo puedan enviar y recibir mensajes a otros agentes. Como la red era con infraestructura el envío de estos mensajes siempre se realiza a través del contenedor principal.

Construir una plataforma de agentes para entornos ad-hoc, por lo tanto, plantea nuevos retos todavía sin abordar. Durante estos últimos años hemos estudiado y alcanzado alguno de ellos. Nuestra primera contribución ha sido definir un nuevo perfil J2ME, construido sobre MIDP, que incluya una serie de clases adicionales que nos permitan desarrollar sobre él una plataforma de agentes. Este perfil lo hemos denominado Travel Agents Profile (TAgentsP) [Campo, 2002a], por analogía con la plataforma de agentes sobre J2SE, TAgents [Letsch, 2000], desarrollada en nuestro grupo de investigación. En la actualidad este perfil da soporte a:

- Un mecanismo de serialización semiautomático, muy similar al soportado en J2SE. En [Campo et al., 2003] se describe de forma detallada.

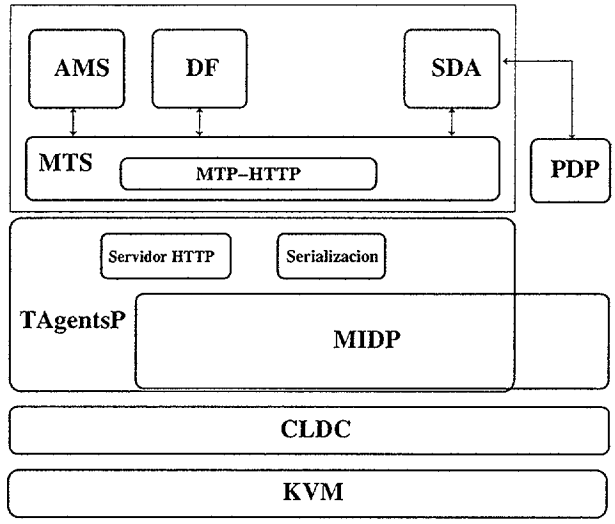


Figura 6.5: Perfil TAgentsP y plataforma propuesta para entornos ad-hoc.

- Un mecanismo de comunicación directa entre dispositivos basado en el protocolo HTTP 1.1. Para ello se ha desarrollado una serie de clases que nos permiten ejecutar un servidor HTTP sobre MIDP: clases para la creación de un sistema de archivos (no existente en MIDP), soporte a tipos MIME y opciones de configuración del servidor. En [Díez-Andino et al., 2003c] se describe de forma detallada.

Sobre el perfil desarrollado, en la actualidad se está abordando la construcción de una plaforma basada en nuestra propuesta para el modelo de referencia FIPA Ad-Hoc, ver Figura 6.5, que incluye:

- Un AMS limitado, que gestiona el ciclo de vida de los agentes pero no proporciona un servicio de páginas blancas.
- Un DF que interacciona con el SDA que emplea una implementación de PDP sobre MIDP.
- Un MTS que emplea el MTP HTTP estándar de FIPA, que se construye sobre el servidor HTTP desarrollado.

6.7. Conclusiones

En este capítulo hemos realizado un estudio sobre la adaptación de los estándares de FIPA a entornos ad-hoc, centrándonos en definir un mecanismo alternativo

a la federación de DFs para el descubrimiento de servicios ofrecidos por agentes remotos en este tipo de entornos.

En la sección 6.1 hemos planteado la motivación de realizar esta contribución y en la sección 6.2 hemos detallado los objetivos que nos hemos planteado para ello. Desde nuestro punto de vista, el DF debe seguir siendo el servicio de directorio que emplean los agentes para registrar sus servicios y para solicitar búsquedas, lo que se debe modificar es la manera en la que el DF debe localizar agentes remotos, ya que el mecanismo de federación no es válido en estos entornos. Para ello hemos introducido un nuevo agente el SDA, que será el encargado de realizar búsquedas de agentes remotos de forma eficiente en una red ad-hoc.

En la sección 6.3, hemos analizado las ventajas e inconvenientes de dos tipos de soluciones: la primera de ellas, consiste en que el SDA implemente internamente un protocolo de descubrimiento de servicios y la segunda, que el SDA base su funcionamiento en una implementación de un mecanismo de descubrimiento ya existente. El principal inconveniente de la primera alternativa, es la necesidad de transmitir mensajes ACL por difusión que, por una parte supone una importante carga computacional en las plataformas y por otra, que semánticamente ACL no soporta difusión. La segunda alternativa, presenta la desventaja que si queremos que sea muy genérica, de manera que el SDA pueda interaccionar con cualquier protocolo de descubrimiento, introducimos una importante carga computacional en el SDA ya que debe realizar las correspondencias entre las descripciones de servicios a nivel agentes y el formato de descripción de servicios que emplee el mecanismo de descubrimiento. Además, esta solución plantea problemas de incompatibilidad si distintas plataformas en una misma red ad-hoc no emplean el mismo mecanismo de descubrimiento, ya que no se podrían descubrir servicios. Finalmente, la solución que proponemos, es que el SDA emplee un mecanismo de descubrimiento concreto, que por los estudios realizados en esta tesis y descritos en los capítulos 4 y 5, proponemos que sea el PDP. En la sección 6.4 formalizamos esta propuesta, definiendo una ontología para el SDA.

Las ideas planteadas en este capítulo han sido expuestas en el comité técnico FIPA Ad-hoc, que aborda este mismo problema. Algunas de ellas se han tenido en cuenta para realizar las versiones preliminares de las especificaciones que se han presentado en la reunión del FIPA celebrada en Noviembre de 2003, coincidiendo con la finalización de esta tesis doctoral. En la sección 6.5, resumimos el trabajo del comité técnico y nuestras contribuciones al mismo.

Además de abordar el reto de adaptar las especificaciones FIPA a entornos ad-hoc, hemos llevado a cabo una serie de estudios e implementaciones que permiten realizar el desarrollo de la plataforma propuesta en dispositivos reales que soportan el perfil MIDP de J2ME. En la sección 6.6 describimos el estado actual

de nuestros desarrollos y las contribuciones realizadas en este reto.

Este trabajo deja importantes líneas futuras abiertas: en la adaptación de las especificaciones FIPA debe estudiarse en detalle qué simplificaciones se deben realizar a nivel AMS y a nivel MTS. Creemos que el AMS debería sólo mantener funciones de gestión de ciclo de vida de los agentes, que son imprescindibles en una plataforma, pero quizás se podría prescindir del servicio de páginas blancas que ofrece. En cuanto al MTS, proponemos emplear el MTP estándar HTTP, por ser HTTP un protocolo lo suficientemente sencillo y fácil de implementar en dispositivos reales. A nivel comunicación de agentes creemos que se debería realizar un estudio en profundidad del lenguaje de comunicación de agentes ACL, y realizar simplificaciones al mismo, porque es lo que introduce una mayor carga computacional en la implementación de una plataforma de agentes compatible con FIPA en dispositivos limitados.

Capítulo 7

Conclusiones y trabajos futuros

La computación ubicua comienza a ser una realidad gracias fundamentalmente a los avances realizados en el campo de la microelectrónica, que permiten dotar de capacidades de cómputo y comunicación a cada vez más elementos del mundo físico. El reto ahora es aprovechar estas capacidades para desarrollar aplicaciones que permitan al usuario que ese mundo físico se adapte a sus necesidades, de manera transparente.

En esta tesis doctoral se ha abordado este nuevo reto, contribuyendo a la definición de tecnologías middleware para el desarrollo de nuevos servicios en entornos de computación ubicua. Inicialmente nuestra investigación se centró en la adaptación de la tecnología de agentes para utilizarla como middleware para el desarrollo de estos nuevos servicios. Su análisis basado en los estándares FIPA, nos llevó a descubrir y a abordar un nuevo problema, que fue el diseño de un protocolo de descubrimiento de servicios adaptado a las restricciones que imponen los entornos ubicuos. Los resultados obtenidos se aplicaron al problema inicial, en concreto a la adaptación del servicio de páginas amarillas definido en FIPA, denominado Directory Facilitator.

En este capítulo recopilamos las principales contribuciones de esta tesis, sección 7.1, exponemos las conclusiones extraidas de la realización de la misma, sección 7.2 y finalizamos con los trabajos futuros en la sección 7.3.

7.1. Resumen y principales contribuciones

Las principales contribuciones que se han realizado en esta tesis doctoral son las siguientes:

1. **Estudio de los nuevos retos impuestos por los entornos de computación.**

   La computación ubicua es una evolución de la computación móvil, pero plantea algunos nuevos retos que no se habían abordado hasta ahora: definición de nuevos protocolos de red inalámbrica para la comunicación entre dispositivos, nuevos protocolos que permitan que los dispositivos formen redes sin necesidad de infraestructura ni configuraciones de administradores, desarrollo de servicios adaptados al contexto y a las propias preferencias del usuario, para que el entorno se adapte a sus necesidades de manera transparente, estudio de nuevos mecanismos de interacción hombre-máquina,...

   En el capítulo 1, detallamos estos nuevos retos partiendo de una clasificación de los nuevos tipos de dispositivos existentes: personales, de función específica y sensores/actuadores, y describiendo algunos de los nuevos escenarios de aplicación, que ilustran las posibilidades de la computación ubicua.

2. **Estudio de mecanismos de descubrimiento de servicios.**

   El problema del descubrimiento de servicios no es algo nuevo, a lo largo de estos últimos años han aparecido diversas propuestas al respecto, más o menos extendidas en entornos de redes fijas. El objetivo principal con el que surgieron fue facilitar a un dispositivo móvil el descubrimiento, configuración y uso de los servicios que se ofrecían en la nueva red a la que se conectaba. En general, las diferentes propuestas abordan el problema de distintas formas, desde las que han sido diseñadas para su uso con un determinado protocolo de red, hasta las que están asociadas a un determinado lenguaje de programación. En el capítulo 2, se describen brevemente algunos de los más destacados mecanismos de descubrimiento de servicios definidos en los últimos años.

   En el capítulo 3, realizamos un estudio teórico y detallado del problema de descubrimiento, planteamos de forma genérica el problema y enumeramos las diferentes soluciones teóricas existentes, denominadas: modo *pull*, modo *push* y directorio. Los mecanismos de descubrimiento existentes basan su funcionamiento en alguna o en varias de estas soluciones teóricas, y para ilustrarlo, también en este capítulo, describimos de forma detallada cómo funcionan las propuestas existentes en Internet: SLP, SSDP y DNS-SD.

3. **Estudio analítico y mediante técnicas de simulación de las prestaciones de los protocolos de descubrimiento.**

   En el capítulo 4 y 5, abordamos el estudio de prestaciones de los mecanismos de descubrimiento de servicios, respecto a tres variables:

- Número de mensajes por búsqueda, que nos permite comparar los diferentes protocolos desde el punto de vista de su consumo energético.
- Tasa de descubrimiento de servicios, que nos permite comparar los diferentes protocolos desde el punto de vista de eficiencia, lo ideal es que el protocolo nos permita obtener tasas de descubrimiento de servicios del 100 %.
- Tasa de descubrimiento de servicios falsos, que nos permite comparar los diferentes protocolos desde el punto de vista de eficiencia, lo ideal es que el protocolo nos permita obtener tasas de descubrimiento de servicios falsos del 0 %.

En el caso de los protocolos teóricos, *push* y *pull*, obtuvimos mediante análisis matemático el valor de estos tres parámetros. Para algunos de los protocolos propuestos en Internet, como SLP y SSDP, el análisis se ha realizado empleando técnicas de simulación, para lo cual se ha utilizado el lenguaje de simulación MODSIM y se han aplicado métodos de medias por bloques para determinar el final de simulación. Todos los resultados se han realizado con un intervalo de confianza del 10 % y un nivel de confianza del 90 %.

4. **Propuesta de nuevos mecanismos para mejorar las prestaciones de protocolos de descubrimiento en entornos ubicuos.**

   El análisis realizado sobre las características de los entornos ubicuos, nos llevó a determinar una serie de requisitos que se deben imponer a la hora de definir protocolos de descubrimiento que operen en estos entornos. Estos requisitos se enumeran a continuación:

   - Minimizar número de transmisiones: para que reduzca el consumo energético y se adapte a las restricciones de los protocolos inalámbricos.
   - Funcionar sin infraestructura fija: para que pueda funcionar en redes ad-hoc en las que los dispositivos entran y salen de forma espontánea, y en las que el funcionamiento de la red no depende de ningún elemento central que precise administración.
   - Adaptarse tanto a entornos dinámicos como estáticos: para que se puedan obtener las mejores prestaciones adaptándose al entorno en el que se encuentra.
   - Adaptarse a las características de las aplicaciones: para mejorar las prestaciones teniendo en cuenta las necesidades de las aplicaciones que realizan las búsquedas.

- Maximizar la cooperación entre los dispositivos: para obtener un mayor beneficio común reduciendo el coste global.
- Simple y poco costoso computacionalmente: para que sea implementable en dispositivos con restricciones de memoria y capacidad de proceso.

Partiendo de los análisis teóricos del modo *push* y el modo *pull*, hemos propuesto, como mejor solución para entornos ubicuos, realizar un mezcla de ambos modos, beneficiándose de sus ventajas e intentando compensar sus inconvenientes. Así, hemos propuesto una serie de mecanismos que nos permiten mejorar las prestaciones de los protocolos de descubrimiento en entornos ubicuos y por lo tanto, verificar los requisitos enumerados anteriormente:

- No emplear servicios de directorio.

  El protocolo debe tener un funcionamiento totalmente distribuido de tal manera que pueda operar en una red sin infraestructura y sin administración.
- Introducir tiempo de disponibilidad en los dispositivos.

  En entornos ubicuos un gran número de dispositivos serán móviles, por lo tanto es necesario que anuncien sus servicios indicando el tiempo de vida asociado a ellos. En esta tesis proponemos emplear un tiempo de disponibilidad asociado al dispositivo, cuyo valor depende de sus características de movilidad.
- Introducir caché de servicios remotos.

  Se introduce una caché en los dispositivos, en la que se almacenan de forma local los servicios que se anuncian en la red. Cuando una aplicación solicita la búsqueda de un servicio se comprueba si existe en la caché y si es así, no se transmite ninguna petición de búsqueda. Con este mecanismo se consigue disminuir el número de mensajes por búsqueda. Por ejemplo, al introducir en un modo *pull* una caché de tamaño 35, el número de mensajes por búsqueda en un escenario con 20 dispositivos y 5 tipos distintos de servicios, se reduce de 4,8 a 1,8.

  El principal inconveniente al introducir una caché es que las tasas de servicios descubiertos y de servicios falsos empeora, siendo en el ejemplo considerado del 71 % y del 14 %, respectivamente.
- Mensajes de respuesta por difusión.

  Cuando un dispositivo responde a un mensaje de búsqueda envía su respuesta por difusión, de tal forma que todos los dispositivos pueden incluir en sus cachés este nuevo servicio.

Con este mecanismo se reduce el número de mensajes por búsqueda, ya que la caché de los dispositivos está más actualizada, y por lo tanto, se satisfacen más búsquedas con los servicios almacenados en la caché. Siguiendo con el ejemplo anterior, al permitir que los mensajes de respuesta se transmitan por difusión, el número de mensajes por búsqueda se reduce de 1,8 a 0,48.

Con este mecanismo se mejora la tasa de servicios descubiertos, ya que la caché está más actualizada, aunque la tasa de servicios falsos empeora ligeramente, en el ejemplo considerado los valores son del 86 % y del 28 %, respectivamente.

- Incluir servicios ya conocidos en los mensajes de respuesta.

  Con el objetivo de mejorar las tasas de servicios descubiertos y servicios falsos, cuando se solicita una búsqueda, el dispositivo transmite una petición a la red incluyendo los servicios que conoce, es decir, que están almacenados en su caché, de manera que sólo responden los dispositivos que ofrecen servicios que no están incluidos en el propio mensaje de petición.

  Con este mecanismo se consigue mejorar las tasas de servicios descubiertos y servicios falsos, obteniéndose en el ejemplo considerado un valor del 99,5 % y del 10 %, respectivamente.

- Responder con los mensajes almacenados en las cachés.

  Cuando se busca un servicio no sólo responde el dispositivo que ofrece el servicio, sino todos aquellos que tienen almacenado ese servicio en sus cachés. Y un dispositivo sólo responde si alguno de los servicios de los que conoce no ha sido anunciados previamente por otros.

  Este mecanismo, basado en maximizar la cooperación de los dispositivos, permite disminuir el número de mensajes por búsqueda. Además, si priorizamos que responda el dispositivo que conoce más servicios todavía se obtiene un mejor resultado. Para hacer esto, hemos propuesto que los dispositivos que tienen un mayor tiempo de disponibilidad y conozcan a un mayor número de servicios del tipo solicitado respondan antes, para ello retardan sus respuestas un tiempo aleatorio inversamente proporcional a estos dos valores.

  En el ejemplo considerado, al introducir este mecanismo se pasa de tener 0,49 mensajes por búsqueda a 0,36.

- Actualizar la caché con servicios incluidos en los mensajes de petición.

  Con este mecanismo los dispositivos no sólo almacenan en sus cachés servicios incluidos en los mensajes de respuesta, sino también en los

mensajes de petición. De esta forma se disminuye ligeramente el número de mensajes por búsqueda y se reduce la tasa de servicios falsos.

En el ejemplo considerado, al introducir este mecanismo se pasa de tener 0,36 mensajes por búsqueda a 0,16.

- Introducir mecanismos de consistencia de cachés.

  Al introducir una caché de servicios remotos, se consigue disminuir el número de mensajes por petición, pero provoca que se descubran servicios que no están disponibles realmente en la red, por ello es importante introducir mecanismos que permitan disminuir el número de servicios falsos que existen en la cachés. Los mecanismos que hemos propuesto son los siguientes:

  - Los dispositivos cuando anuncian un servicio local incluyen su tiempo de disponibilidad en la descripción del servicio. Los dispositivos remotos que escuchan este anuncio, almacenan el servicio en su caché durante un tiempo que es el mínimo entre el tiempo de disponibilidad del dispositivo que lo anuncia y el tiempo de disponibilidad del dispositivo que lo almacena.
  - Cuando los dispositivos detectan un cambio de red o apagado del dispositivo, envían un mensaje especial que indica a los demás dispositivos de la red que esos servicios dejan de estar disponibles, y por lo tanto, los borran de sus cachés.
    Del mismo modo, si un dispositivo al intentar acceder a un servicio que acaba de descubrir detecta que no está disponible, puede enviar un mensaje indicándoselo a los demás dispositivos de la red, para que del mismo modo borren estas entradas de sus cachés.

  Al introducir estos mecanismos en el protocolo, se consigue que las cachés estén actualizadas alcanzándose tasas de servicios falsos descubiertos próximas al 0 %, que es el valor deseable.

- Optimizar según la utilización que hacen las aplicaciones de los servicios descubiertos.

  Hemos realizado una clasificación de las aplicaciones según el uso que realizan de los servicios descubiertos, con el objetivo de mejorar la eficiencia de los protocolos, así hemos definido tres grupos:

  - Aplicaciones que quieren localizar algún dispositivo que ofrece un determinado servicio, pero sin importar cuál es el dispositivo que lo ofrece. Para ello definimos un tipo de petición denominada "una petición–una respuesta".
    En este caso, cuando se realiza una búsqueda si existe el servicio

solicitado en la caché se responde con él y en caso de realizar una consulta a la red sólo responde un dispositivo.

- Aplicaciones que quieren localizar todos los dispositivos que ofrecen un determinado servicio en el entorno. Para ello definimos un tipo de petición denominada "una petición–varias respuestas".
- Aplicaciones tipo buscador, que quieren localizar todos los servicios que se ofrecen en el entorno. Para ello definimos un nuevo tipo de servicio denominado ALL.

En los mensajes de petición se incluye el tipo de búsqueda que se realiza para que los dispositivos que conocen los servicios solicitados generen sus respuestas consecuentemente. A modo de ejemplo, al distinguir peticiones del tipo "una petición–una respuesta" se obtiene una reducción en el número de mensajes de hasta el 86 %.

5. **Diseño de un nuevo protocolo de descubrimiento de servicios: Pervasive Discovery Protocol.**

   Algunos de los mecanismos propuestos en el capítulo 4, explicados brevemente en el apartado anterior, pueden ser aplicados a los protocolos de descubrimiento de servicios existentes en la actualidad, siempre que soporten un modo funcionamiento distribuido, como por ejemplo SLP, SSDP y DNS-SD. De esta forma, se obtiene una versión del protocolo más eficiente para su aplicación en entornos de computación ubicua.

   Partiendo de un modo *pull* puro e introduciendo todos los mecanismos propuestos, hemos definido un nuevo protocolo, que hemos denominado Pervasive Discovery Protocol. En el capítulo 4, se ha realizado una formalización del mismo, describiéndolo en forma de algoritmo. Para facilitar su implementación en dispositivos reales y garantizar la interoperabilidad entre diferentes implementaciones hemos realizado una descripción detallada del mismo, que se incluye como anexo B.

6. **Estudio de prestaciones de PDP respecto a propuestas teóricas y de Internet.**

   En esta tesis se ha realizado un estudio de prestaciones del protocolo diseñado, PDP, respecto a una serie de parámetros:

   - Tiempo de disponibilidad.
     El tiempo de disponibilidad de los dispositivos es un parámetro importante dentro del protocolo PDP. Inicialmente, este tiempo se configura manualmente dependiendo de la características de movilidad del dispositivo, por ejemplo, dispositivos personales y móviles tendrán tiempos

de disponibilidad bajos, mientras que los dispositivos tipo PC tendrán tiempos más altos.

Hemos realizado una serie de simulaciones para comprobar qué efecto tiene que este tiempo de disponibilidad no esté ajustado realmente al tiempo de vida de los dispositivos. Las principales conclusiones es que si se introducen mecanismos de consistencia de cachés las desviaciones de este parámetro no afectan demasiado a las prestaciones del protocolo, y en caso de no existir estos mecanismos, es mejor estimar a la baja este valor.

- Número de dispositivos.

  En cuanto a las prestaciones del protocolo respecto al número de dispositivos, depende principalmente del tamaño relativo de la caché empleada.

- Tamaño de la caché.

  En las simulaciones realizadas en distintos tipos de escenarios, modificando tanto el tiempo de disponibilidad de los dispositivos como su número, hemos observado que las prestaciones del protocolo dependen fundamentalmente del tamaño de la caché de los dispositivos.

  De los análisis realizados mediante simulación, podemos concluir que existe un tamaño óptimo de caché en PDP que se corresponde con el número de servicios que existen en la red. Por lo tanto, un mayor tamaño de la caché no mejora las prestaciones del protocolo, lo cual es deseable debido a la restricciones de memoria que pueden presentar algunos de los dispositivos.

Además de este análisis, también hemos realizado un estudio de prestaciones del PDP respecto a otros protocolos de descubrimiento, tanto teóricos como algunos propuestos en Internet, en diferentes escenarios. En los escenarios homogéneos y heterogéneos, comparamos el protocolo respecto a mecanismos de descubrimiento distribuidos como el modo *pull*, *push* y SLP. En escenarios con un dispositivo fijo, añadimos además mecanismos de descubrimiento que hacen uso de directorios.

- Escenarios homogéneos.

  En escenarios homogéneos, en los que todos los dispositivos tienen el mismo tiempo de disponibilidad, se obtiene que las prestaciones de PDP en cuanto a número de mensajes están muy por debajo de los protocolos tipo *pull* como SLP, aproximadamente un 80 %, y que es sólo comparable con un modo *push* con una baja frecuencia de anuncios. En cuanto a la tasa de servicios descubiertos y servicios

falsos, PDP se aproxima a las obtenidas en los modos *pull*, es decir, muy próximas al 100 % y al 0 %, respectivamente, mientras que en el modo *push* con una baja frecuencia de anuncios, la tasa de servicios descubiertos es menor.

- Escenarios heterogéneos.

  En escenarios heterogéneos, en los que existe un porcentaje de dispositivos con distintos tiempos de disponibilidad, las prestaciones de PDP respecto a los otros protocolos se mantienen respecto a las obtenidas en escenarios homogéneos.

  La principal ventaja de PDP en estos entornos, es que los dispositivos con mayor tiempo de disponibilidad son los que responden a un mayor número de peticiones, lo que permite que los dispositivos más limitados y por lo tanto, con mayores restricciones de consumo energético, realicen menos transmisiones.

  Por ejemplo, en un escenario con 40 dispositivos con cinco tiempos de disponibilidad distintos: 500, 2500, 4500, 6500 y 9500 segundos, en el que el porcentaje de dispositivos de cada tipo es uniforme, por lo tanto un 20 %, los dispositivos con tiempo de disponibilidad de 9500 segundos, porcentualmente responden a casi el 50 % de los mensajes de búsqueda.

- Escenarios con un dispositivo fijo.

  En estos escenarios comparamos el protocolo con mecanismos de descubrimiento distribuidos y centralizados, como SLP con directorio o un directorio teórico. Los resultados obtenidos muestran que PDP consigue disminuir el número de mensajes por búsqueda incluso respecto a las soluciones centralizadas.

- Soporte a aplicaciones tipo "buscador"

  Hemos comparado las prestaciones de PDP respecto a mecanismos de descubrimiento que dan soporte a aplicaciones tipo buscador, como son SLP y SSDP. Los resultados obtenidos son mucho más favorables a PDP en cuanto a número de mensajes, ya que reduce su valor en aproximadamente un 97 % respecto a SLP y a un 95 % respecto a SSDP, manteniéndose las tasas de servicios descubiertos y servicios falsos obtenidas.

7. Implementación de referencia de PDP en J2ME.

Para validar el requisito impuesto de que el protocolo sea simple y poco costoso computacionalmente, hemos llevado a cabo una implementación del protocolo empleando el lenguaje de programación J2ME, [Perea, 2003], en

concreto empleando la máquina virtual J9 desarrollada por IBM [1] y sobre el Personal Profile para tener soporte multicast. El ejecutable ocupa 22 KB sin emplear ninguna técnica para optimizar su tamaño y se ha probado satisfactoriamente en un Pocket PC.

También se ha realizado una implementación de PDP en el simulador de red Network Simulator[2] ns–2, [Casillas, 2003].

8. **Adaptación de la especificaciones FIPA a entornos ad-hoc.**

En esta tesis, se propone emplear como tecnología middleware para el desarrollo de servicios en entornos ubicuos el paradigma de agentes, en el capítulo 6 se justifica la validez de esta aproximación, apoyada en la literatura por otros proyectos e iniciativas.

Uno de los objetivos marcados es que la contribuciones realizadas en esta temática estuvieran basadas en las especificaciones FIPA, estándares de facto en agentes en la actualidad. Para ello realizamos un estudio detallado de los servicios definidos en el modelo de referencia FIPA y propusimos las siguientes modificaciones:

- En cuanto al servicio de páginas amarillas, DF, proponemos eliminar la federación de DFs como mecanismo para descubrir agentes en plataformas remotas. Como mecanismo alternativo proponemos la introducción de un agente, el SDA, que emplea el DF para realizar búsquedas en redes ad-hoc. A su vez el SDA emplea un sistema de descubrimiento de servicios genérico, que después de analizar diversas alternativas, hemos propuesto que fuese el PDP.

  Esta propuesta se ha convertido en la contribución más importante en esta temática.

- En cuanto al AMS, en una primera aproximación, reducimos su funcionalidad a la gestión de agentes locales a la plataforma y eliminamos el servicio de páginas blancas.

- En cuanto al MTS, en una primera aproximación, proponemos que se utilice el MTP estándar HTTP, por ser un protocolo sencillo, fácil de implementar en dispositivos limitados y del que ya existe una especificación FIPA.

[1] http://www-306.ibm.com/software/wireless/wsdd/
[2] http://www.isi.edu/nsnam/ns/

9. **Participación en el comité técnico FIPA Ad-hoc.**

   Las propuestas realizadas en cuanto a la adaptación de las especificaciones FIPA a entornos ad-hoc, han sido llevadas al comité técnico del FIPA Ad-hoc, realizando las siguientes contribuciones:

   - Respuesta al Call For Technology FIPA Ad-hoc, donde exponíamos nuestras primeras ideas sobre la adaptación del servicio de páginas amarillas a entornos ad-hoc. Este documento se encuentra en `http://www.fipa.org/docs/input/f-in-00063/`.
   - Invitación a FIPA 26 celebrado en Helsinki para exponer ante el comité técnico nuestras contribuciones, para ello realizamos un documento más detallado que se encuentra en `http://www.fipa.org/docs/input/f-in-00070/`.
   - Contribuciones al White Paper: *Agents in Ad-hoc enviroments*, escrito con el objetivo de realizar un estudio sobre las diferentes alternativas propuestas y un estado del arte en las diferentes tecnologías existentes para el descubrimiento de servicios. En él se incluye parte de nuestro estudio realizado sobre algunos protocolos de descubrimiento de servicios, entre ellos el propuesto por nosotros, PDP, y también algunas consideraciones sobre los requisitos que debe verificar la propuesta realizada. Este documento se encuentra en `http://www.fipa.org/docs/input/f-in-00068/f-in-00068A.htm`.
   - Contribuciones a las especificaciones para descubrimiento de agentes en entornos ad-hoc, que se están discutiendo en la actualidad.

10. **Estudio de la implementación de plataformas de agentes en dispositivos limitados con J2ME.**

    Junto al diseño de la adaptación de las especificaciones de FIPA, hemos llevado a cabo una serie de desarrollos, con el objetivo de implementar la plataforma de agentes en dispositivos reales:

    - Implementación del perfil TAgentsP [Díez-Andino, 2002], que complementa al perfil estándar MIDP, pero que incluye soporte a:
      - Serialización.
      - Implementación de un servidor HTTP, para basar la comunicación de agentes y su migración en el protocolo HTTP.
    - Implementación de una plataforma de agentes según nuestra propuesta para FIPA Ad-hoc, basada en la modificación de la plataforma LEAP. Este desarrollo se está llevando a cabo en la actualidad tomando como base el perfil anterior.

7.2. Conclusiones

Las principales conclusiones que se extraen de la realización de esta tesis son las siguientes:

- En cuanto al estudio de protocolos de descubrimiento de servicios, hemos visto que las diferentes propuestas existentes en la actualidad no cubren las necesidades que imponen los entornos de computación ubicua, fundamentalmente por dos razones:

  - algunas basan su funcionamiento en servicios de directorio que no son válidos en algunos escenarios en los que no se puede depender de dispositivos concretos, ni de configuraciones realizadas por administradores;
  - algunas soluciones distribuidas hacen un uso excesivo de mensajes de difusión, por lo que los dispositivos limitados pueden tener un gran consumo energético.

  Existen entornos estáticos en los que las soluciones tradicionales pueden servir e incluso ser más eficientes, pero debido a la amplia variedad de escenarios en los que un dispositivo puede encontrarse, la solución al problema de descubrimiento en computación ubicua debe ser lo suficientemente flexible para que funcione de forma eficiente en todos los escenarios posibles. Por lo tanto, debemos dar una solución válida en aquellos escenarios en los que se imponen mayores restricciones como es el caso de las redes que se forman espontáneamente cuando un conjunto de dispositivos se aproximan entre sí, denominadas redes ad-hoc.

  Además, en computación ubicua los dispositivos tienen mayor necesidad de interactuar cuanto más próximos se encuentren unos a otros, por lo tanto el protocolo de descubrimiento tendrá un ámbito local para poder descubrir los servicios que se ofrecen en un entorno próximo, no a nivel global.

- En cuanto al protocolo propuesto, PDP, hemos visto mediante estudios de simulación que se adapta a las restricciones de los entornos ubicuos, ya que:

  - Es un protocolo totalmente distribuido, por lo tanto, no precisa ninguna configuración, ni administración para su funcionamiento.
  - Es un protocolo que reduce el consumo de baterías, ya que disminuye considerablemente el número de mensajes transmitidos por búsqueda, tanto respecto a soluciones distribuidas como centralizadas.

Además, en el caso de existir dispositivos con menos restricciones energéticas el funcionamiento del protocolo hace que estos dispositivos respondan a más peticiones de búsqueda, disminuyendo la carga de los más limitados.

- Es un protocolo eficiente, ya que alcanza tasas de descubrimiento de servicios altas, próximas al 100 % y de servicios falsos bajas, próximas al 0 %.
- Es un protocolo que se comporta bien tanto en escenarios dinámicos, como estáticos.
- Es un protocolo simple y poco costoso computacionalmente, y por lo tanto, fácil de implementar en dispositivos limitados.

El protocolo PDP tiene dos parámetros importantes de los que dependen sus prestaciones, por una parte, el tiempo de disponibilidad asociado a los dispositivos y por otra, el tamaño de la caché. En cuanto al tiempo de disponibilidad, hemos visto que es mejor estimar a la baja su valor, y que si existen mecanismos de consistencia de cachés no es demasiado crítico que este valor no esté perfectamente ajustado al tiempo de vida de los dispositivos. En cuanto al tamaño de la caché, hemos podido comprobar de manera experimental que existe un tamaño óptimo que se corresponde con el número de servicios existentes en la red, y aunque aumentemos su valor por encima de éste, no se obtienen mejores prestaciones.

- En cuanto a la adaptación de las especificaciones de FIPA a entornos ad-hoc, hemos visto que supone importantes retos, sobre todo la adaptación del servicio de páginas amarillas proporcionado, en terminología FIPA, por el Directory Facilitator.

  Hemos visto que el principal reto que se plantea es permitir realizar búsquedas eficientes en redes ad-hoc, ya que el mecanismo tradicional de federación de DFs no es válido. Nuestra propuesta se basa en no modificar la forma en la que los agentes realizan sus búsquedas y por lo tanto, mantener el DF, pero eliminando el mecanismo de federación y empleando un nuevo agente, que hemos denominado SDA, que realiza búsquedas eficientes en redes ad-hoc apoyándose en un protocolo de descubrimiento de servicios subyacente, que en nuestra propuesta será el PDP.

  Nuestra colaboración en el comité técnico del FIPA Ad-hoc nos ha permitido validar las contribuciones realizadas en este campo. Aunque algunas de nuestras ideas se han tenido en cuenta, las especificaciones preliminares difieren de nuestra propuesta en dos aspectos:

- La eliminación del DF, que desde nuestro punto de vista presenta el inconveniente de que los agentes deben ser conscientes de si la plataforma en la que residen opera en una red ad-hoc o no.
- La posibilidad de que el SDA emplee cualquier mecanismo de descubrimiento, porque esto supone introducir una gran complejidad en el sistema, ya que el SDA debe realizar las correspondencias entre descripciones a nivel agentes y el formato de descripción de servicios empleado por los mecanismos de descubrimiento subyacentes. Otro problema importante es el de la interoperabilidad entre plataformas que empleen distintos mecanismos.

- Durante todo el desarrollo de esta tesis doctoral, hemos estudiado la viabilidad de realizar implementaciones de nuestras propuestas en dispositivos reales, y de hecho hemos llevado a cabo algunas de ellas empleando la tecnología J2ME, que nos permite independizar, en gran medida, los desarrollos realizados del dispositivo concreto en el que se instale.

  En esta línea, hemos definido un nuevo perfil J2ME, construido sobre MIDP, que incluye una serie de clases adicionales que nos permiten desarrollar sobre él una plataforma de agentes. Este perfil lo hemos denominado Travel Agents Profile (TAgentsP) [Campo, 2002a], por analogía con la plataforma de agentes sobre J2SE, TAgents [Letsch, 2000], desarrollada en nuestro grupo de investigación. En la actualidad este perfil proporciona serialización y un mecanismo de comunicación directa entre dispositivos basado en el protocolo HTTP 1.1. Ver sección 6.6.1.

  Aunque podríamos pensar que con la evolución de la tecnología tanto hardware como software algunos de las restricciones que existen hoy, y a las cuales hemos dado respuesta, estarán resueltas en un futuro no muy lejano, desde nuestro punto de vista, siempre existirán nuevos dispositivos con limitaciones en los que introducir esta tecnología y que nos impondrán estos mismos retos.

No queremos concluir esta tesis sin analizar cómo las propuestas de esta tesis doctoral hacen que la vida de María, nuestra protagonista en la sección 1.1.2 de esta memoria, esté más cerca de ser una realidad.

En primer lugar vemos que a María le rodean en todas partes (la oficina, el hogar, el aeropuerto, el hotel) un gran número de dispositivos. En esta tesis, en primer lugar, hemos estudiado los retos que imponen los entornos de computación ubicua, y en particular hemos hecho una clasificación de los nuevos tipos de dispositivos existentes (contribución 1). Para que toda esta tecnología le ayude y

se haga transparente, es necesario que todos los dispositivos cooperen allí donde se encuentra María. María sólo lleva un dispositivo consigo, su dispositivo personal, una PDA, que se encargará de ocultarle los detalles de un entorno repleto de dispositivos, se anticipará a sus necesidades y solicitará interacción a María únicamente cuando sea imprescindible.

El escenario comienza cuando María se plantea que quiere ir de vacaciones a París dentro de un par de meses. No quiere perder tiempo en los detalles de buscar vuelo, hotel, etc. y por eso delega esas tareas a su PDA. Se trata éste de un problema clásico que se puede resolver de forma eficiente mediante tecnología de agentes gracias a sus características de autonomía, comunicación y cooperación. Un agente en la PDA podría encargarse de buscar la mejor oferta y ofrecérsela a María. Para ello saltaría a otras plataformas, haría la búsqueda en Internet sin consumir los recursos escasos de la PDA (CPU, batería, comunicaciones) y volvería a la PDA cuando tuviera los resultados. Esta solución era hasta ahora sólo viable si la PDA usaba una plataforma de agentes, tipo LEAP, que se apoyara en un dispositivo no limitado y fijo. Esto es así, como se ha visto a lo largo de esta memoria, por dos motivos:

- Por un lado, ciertos elementos de la arquitectura FIPA de plataformas de agentes no se adaptan bien al funcionamiento en PDAs con conexión a red inalámbrica, sino que están pensados para plataformas en PCs fijos conectados a LANs. Este es el caso por ejemplo del mecanismo de federación de DFs.

- Por otro lado, los lenguajes de programación para dispositivos limitados (como J2ME) no proporcionan cierta funcionalidad necesaria para construir plataformas de agentes completas, como la serialización o la reflexión.

Sin embargo, que la plataforma de la PDA dependa de otro dispositivo es poco flexible y transparente. En esta tesis doctoral (contribuciones 8, 9 y 10) hemos abordado y solucionado estos problemas y hemos hecho posible que la PDA de María aloje una plataforma de agentes FIPA que no dependa de otro dispositivo para su funcionamiento.

De esta manera, la PDA de María podría ejecutar una plataforma en la que se podría lanzar un agente que se encargase de navegar por Internet en busca de ofertas de viajes a París. Otro agente podría estar continuamente aprendiendo las preferencias de María (por ejemplo, sabría que suele visitar museos y exposiciones de pintores impresionistas). Un tercer agente, sabiendo que está planeando un viaje y conociendo sus preferencias, podría proactivamente buscar museos impresionistas en París para proponérselos a María. Finalmente, un cuarto agente

podría explorar la agenda de María para buscar amigos que viven en las proximidades de París.

Otra característica de la PDA y del resto de los dispositivos que acompañan a María es que son capaces de descubrirse unos a otros e interaccionar. Por ejemplo, la PDA descubre que tiene próximo un teléfono y lo usa para hacer una llamada. Esto es posible gracias a que la PDA usa algun protocolo de descubrimiento de servicios. En esta tesis doctoral hemos estudiado los protocolos de descubrimiento existentes, analíticamente y mediante simulación (contribuciones 2 y 3) y hemos visto que ninguno se adapta bien a un entorno como el que nos planteamos. Esto nos ha llevado a estudiar cómo se puede mejorar estos protocolos (contribución 4), proponer un nuevo protocolo, el PDP (contribución 5), estudiar sus prestaciones (contribución 6) e implementarlo en una PDA (contribución 7) para demostrar su viabilidad.

Usando un protocolo como PDP, la PDA descubre que tiene en su proximidad un teléfono y lo usa para hacer una llamada a los amigos de María en París. PDP se adapta además tanto a entornos estáticos (el hogar) como dinámicos (el aeropuerto), con o sin infraestructura. Además minimiza el consumo de baterías y permite una mayor autonomía de la PDA de María.

Gracias a un protocolo como PDP, cuando se aproxima la fecha del viaje un agente en la PDA descubre que tiene una nevera en el hogar y le pregunta qué alimentos tiene en su interior, descubre que hay una televisión encendida y la utiliza para mostrar un mensaje, descubre un sistema de climatización y lo programa para que esté a una determinada temperatura, descubre un sistema de gestión de las luces de la casa y lo programa para que se enciendan y apaguen, etc. Todas estas acciones las llevarán a cabo agentes que se ejecutan en la plataforma de la PDA de María. Será un agente, similar al que busca ofertas de viaje a París, el que a partir de la lista de alimentos en la nevera y de sus fechas de caducidad, hará búsquedas en Internet y sugerirá ciertos menús. También será un agente el que controlará en el hogar el sistema de climatización o el de luz mientras no esté María.

También un protocolo de descubrimiento como PDP es el que hace posible que la PDA de María pueda descubrir y usar la tarjeta inteligente que le da el recepcionista, el ascensor del hotel, la televisión de la habitación, el sistema de navegación del automóvil que alquila, o que reciba ofertas de las tiendas y restaurantes cuando pasea por las calles de París.

Muchos de los dispositivos que rodean a María en su viaje son dispositivos con los que se encuentra por primera vez la PDA y no sabe cómo manejar. La PDA se descargará dinámicamente el software necesario para controlar esos dispositivos. Este software será también un agente móvil que se descarga en la plataforma. De

esta manera, un agente se descargará en la PDA al llegar al aeropuerto (tanto en el de origen como en el destino) y guiará a María mientras está allí. Algo parecido ocurre cuando llega al hotel y se le descarga un agente que conoce las normas del hotel y permite controlar sus dispositivos. También un agente informará en el coche de las normas de tráfico de Francia, etc.

Aunque esta tesis doctoral ha supuesto un avance para hacer realidad el escenario del viaje de María, todavía son muchos los retos que hay que abordar, algunos de los cuales presentamos en el siguiente apartado.

7.3. Vías de investigación futuras

Para que la visión de Weiser sobre computación ubicua se haga realidad, y la tecnología embebida en el entorno físico ofrezca nuevos servicios adaptados a las necesidades del usuario de manera transparente al mismo, es necesario seguir investigando en muchos campos.

En esta tesis se han aportado soluciones al problema del descubrimiento de servicios, definiendo un nuevo protocolo adaptado a las nuevas necesidades y restricciones que imponen los entornos ubicuos. En esta misma temática se abren las siguientes líneas de investigación:

1. **Estudio y definición de formato de descripción de servicios.**

   El protocolo PDP define cómo es el mecanismo de descubrimiento pero no define un formato de descripción de servicios. En entornos ubicuos aparecen una amplia variedad de nuevos servicios que es necesario definir mediante algún formato determinado que permita la composición.

2. **Estudio y definición de mecanismos para establecer el tiempo de disponibilidad.**

   Como hemos visto en el estudio de prestaciones del protocolo PDP, se emplea internamente un tiempo de disponibilidad asociado a los dispositivos, que depende de sus características de movilidad.

   Inicialmente este tiempo estará preconfigurado con un valor por defecto, dependiendo del tipo de dispositivo. Como línea futura se podrían estudiar y definir mecanismos que permitiesen que este valor se ajuste dinámicamente según determinados factores (ver sección 4.5). Por ejemplo, en el caso de una PDA podría tenerse en cuenta los horarios indicados en la agenda del usuario, o en el caso de un teléfono móvil, empleando datos sobre su localización podrían obtenerse patrones de movimiento que nos permitiese

detectar cuánto tiempo permanece en los lugares más habituales, como es su trabajo o su hogar.

Este línea de investigación no sólo es válida para el protocolo PDP, sino que puede emplearse en otros mecanismos de descubrimiento en los que existe un tiempo de vida asociado a los servicios anunciados.

3. **Estudio y definición de mecanismos para determinar el tamaño óptimo de caché.**

   Como hemos visto en el estudio de prestaciones de PDP, existe un tamaño óptimo de caché, que empiricamente es igual al número de servicios disponibles en la red (ver sección 5.1.3).

   Inicialmente este valor estará preconfigurado en los dispositivos teniendo principalmente en cuenta sus restricciones de memoria. Como línea futura se podrían estudiar mecanismos que permitiesen adaptar el tamaño de la caché dinámicamente, por ejemplo, si se detecta que continuamente se borran entradas de la caché aumentar el tamaño de la misma, y si se observan que el tamaño de la caché es excesivo disminuirlo.

4. **Estudio y definición de nuevos mecanismos para generar el retardo antes de responder a un `PDP_Service_Request`.**

   Como hemos visto en el estudio de prestaciones de PDP en escenarios heterogéneos (ver sección 5.2.2), la forma de generar el retardo antes de responder a un mensaje `PDP_Service_Request` permite que los dispositivos con menos limitaciones respondan a un menor número de mensajes, y por lo tanto, su consumo energético sea menor. Este retardo se calcula de tal forma que sea inversamente proporcional al tiempo de disponibilidad del dispositivo y al número de servicios que va a incluir en su respuesta.

   En la definición detallada del protocolo hemos propuesto una ecuación para generar este retardo (ver ecuación 5.1). Como línea futura se podrían estudiar nuevos mecanismos para calcular este retardo y ver la mejoras que se obtienen en cada caso. Además, esta optimización podría emplearse en otros mecanismos de descubrimiento de servicios, en los que las respuestas a un mensaje de búsqueda se realizan por difusión y para evitar colisiones los dispositivos generan un retardo aleatorio, normalmente siguiendo una distribución uniforme.

5. **Estudio del problema de los nodos ocultos.**

   Las redes ad-hoc de un sólo salto presentan un problema que se conoce como el problema de los nodos ocultos, el planteamiento es el siguiente: dos dispositivos `A` y `B` pueden comunicarse entre sí, a su vez `B` puede comunicar

con un tercer dispositivo C, que no puede comunicarse con A. Si C solicita un servicio que ofrece A, es probable que B lo tenga almacenado en su caché y que le responda a C con él, cuando C detecte que el servicio A no está disponible, enviará un mensaje PDP_Service_Deregister para que se borre esta entrada en las cachés remotas y por lo tanto, borrará esta entrada en la caché de B, aunque para B el servicio ofrecido por A está disponible.

Este problema no ha sido considerado en la solución de PDP por varios motivos, el primero de ellos porque consideramos que el entorno en el que se aplica todos los nodos tienen visibilidad directa entre ellos bien porque el entorno es reducido, o bien porque existe un nodo maestro o un punto de acceso que reencamina el tráfico, o bien porque existe algún protocolo de encaminamiento multisalto; y por otra parte, porque con la tecnología de agentes móviles a la hora de acceder al servicio proporcionado por A, el nodo C podría enviar al nodo B un agente para acceder al servicio de A, para implantar esta solución en el protocolo se debería memorizar cuándo los servicios descubiertos son proporcionados directamente por el dispositivo que responde a la petición y cuándo son proporcionados por terceros.

De todas formas, consideramos que el problema que se plantea es muy interesante por lo que estamos trabajando en su resolución.

En esta tesis, se han realizado contribuciones para adaptar el modelo de referencia FIPA a entornos ad-hoc y de esta manera facilitar el desarrollo de servicios en entornos ubicuos basados en el paradigma de agentes. Nuestra propuesta se ha centrado en solucionar el problema que plantea la adaptación del servicio de páginas amarillas proporcionado por el DF, pero es necesario adaptar los demás servicios básicos de FIPA, AMS y MTS. En esta temática se abren las siguientes líneas de investigación (ver sección 6.7):

6. **Adaptación del AMS.**

   La tarea de gestión del ciclo de vida de los agentes es una tarea fundamental en una plataforma por lo que será necesario que el AMS mantenga esta funcionalidad. Sin embargo, el servicio de páginas blancas proporcionado por él puede ser opcional, aunque debería estudiarse en profundidad el impacto que esto supone.

7. **Adaptación del MTS.**

   Desde nuestro punto de vista, es importante que FIPA establezca un MTP único que se emplee en todas las plataformas que operen en entornos ad-hoc, ya que el uso de distintos MTPs provocaría, que para que existiese comunicación entre las plataformas, existieran pasarelas entre los diferentes MTPs,

con lo que introduciríamos infraestructura en la red lo cual es incompatible con la definición de una red ad-hoc. Por lo tanto, se abre una nueva línea de investigación en la que se debe definir qué protocolo establecer como estándar.

A nivel comunicación entre agentes, también es necesario realizar una reflexión sobre el lenguaje ACL, que desde nuestro punto de vista introduce una importante carga computacional en la plataforma. En este tema se abre una nueva línea de investigación en la que se debe estudiar y proponer una simplificación tanto del número de mensajes, como del número de parámetros de cada mensaje ACL. Una primera contribución en esta línea se ha presentado en [Díez-Andino et al., 2003b].

Una vez finalizado el diseño de la plataforma y su implementación, el siguiente paso es desarrollar servicios para entornos ubicuos basados en agentes móviles. Y a este nivel se abre una interesante línea de investigación que es:

8. **Desarrollo de aplicaciones basadas en contexto.**

   Uno de los principales retos de la computación ubicua es proporcionar al usuario nuevos servicios de manera transparente, para ello es necesario desarrollar aplicaciones basadas en contexto. Por contexto entendemos una información que puede utilizarse para caracterizar la situación de una entidad con la que el usuario o sus aplicaciones pueden interactuar, una entidad puede ser una persona, un lugar o un objeto, [Dey, 2000].

   Por lo tanto, las aplicaciones basadas en contexto deben en primer lugar modelar ese contexto dando respuesta a las preguntas ¿quién?, ¿dónde?, ¿cuándo? y ¿qué? de cada una de las entidades con las que se interacciona y después adaptar su ejecución a esta información obtenida. En este punto se abren una gran diversidad de líneas de investigación, las que modelan la localización del usuario (¿dónde? y ¿cuándo?) o sus preferencias (¿quién?). Nuestra propuesta es dar respuesta a estas preguntas empleando el paradigma de agentes.

Para finalizar, queremos destacar una línea de investigación futura muy importante y en cierta medida ortogonal a todas las líneas anteriores, que es:

9. **Seguridad en entornos ubicuos.**

   La seguridad es un tema muy importante en entornos ubicuos, en los que existen una gran variedad de nuevos dispositivos que forman redes de forma espontánea sin infraestructura subyacente. Desde nuestro punto de vista, las

soluciones tradicionales basadas en Infraestructuras de Clave Pública (PKI), ampliamente empleadas en redes fijas, no se pueden aplicar directamente en estos escenarios. Es por esto, que creemos que la seguridad debe basarse en nuevos esquemas basados en confianza distribuida y en este sentido existe en nuestro Departamento una tesis doctoral en curso realizada por Florina Almenárez en la que se propone un nuevo modelo de confianza denominado Pervasive Trust Model (PTM). En la actualidad, estamos trabajando en la integración de este modelo en PDP [Almenárez y Campo, 2003].

Apéndice A

Metodología de simulación

En esta tesis doctoral vamos a estudiar mediante simulación las prestaciones de los diferentes protocolos de simulación tanto teóricos, *pull*, *push* y directorio, como algunos de los protocolos definidos en el IETF, como SLP y SSDP, así como el propuesto en esta tesis doctoral, PDP.

En este apéndice justificamos la elección de técnicas de simulación para estudiar estos protocolos, planteamos la metodología que hemos seguido a lo largo de la tesis y las suposiciones de partida de nuestro estudio.

La simulación nace como una aplicación más de los ordenadores que se basa en su gran capacidad de cálculo y su potencia en términos de velocidad y memoria disponible.

Con su aparición fue posible la aplicación del cálculo numérico al estudio del comportamiento de los sistemas, permitiendo el estudio sistemático de problemas en aquellos casos en los que no se dispone del sistema real ni de una solución analítica fiable, bien porque su construcción sea inabordable (cosa normal en cuanto el sistema es mínimamente complejo), bien porque la fiabilidad del modelado esté cuestionada (a veces, es necesario hacer tales aproximaciones para obtener una solución analítica que el modelo final puede no recoger muchos de los aspectos de comportamiento del sistema modelado).

En nuestro trabajo, aquellos protocolos que son más sencillos los hemos estudiado de forma analítica, dejando el estudio mediante simulación para aquellos en los que la complejidad del sistema lo requería.

A.1. Suposiciones de partida

El objetivo de las simulaciones es estudiar el comportamiento del protocolo de descubrimiento de servicios en sí, por lo que las capas inferiores han sido simplificadas, así que tanto los detalles de la capa física como la capa MAC han sido eliminados.

Durante la simulación, los dispositivos se unen a la red de manera aleatoria, solicitan y ofrecen servicios de forma aleatoria y abandonan la red después de un tiempo aleatorio. El número de dispositivos que existen en la red, varía con el tiempo, pero su media permanece estacionaria. Los tiempos aleatorios siguen una distribución exponencial, mientras que la asignación de servicios ofrecidos a dispositivos sigue una distribución uniforme. Por simplificación, consideramos que cada dispositivo ofrece un único servicio.

A.2. Metodología de simulación en la tesis

A.2.1. Definición de objetivos

Con las simulaciones realizadas las variables que queremos medir para comparar los diferentes protocolos, son las siguientes:

- **Número de mensajes por búsqueda**: el número de mensajes transmitidos en la red, normalizado al número de búsquedas de servicios, que nos permite comparar los diferentes protocolos desde el punto de vista de su consumo energético.

- **Tasa de descubrimiento de servicios**: el porcentaje del número de servicios descubiertos, respecto al número total de servicios disponibles en la red, que nos permite comparar los diferentes protocolos desde el punto de vista de eficiencia, lo ideal es que el protocolo nos permita obtener tasas de descubrimiento de servicios del 100 %.

- **Tasa de descubrimientos falsos**: el porcentaje del número de servicios descubiertos que en realidad no están disponibles en la red, respecto al número total de servicios descubiertos, que nos permite comparar los diferentes protocolos desde el punto de vista de eficiencia, lo ideal es que el protocolo nos permita obtener tasas de descubrimiento de servicios falsos del 0 %.

A.2.2. Modelado del tiempo

Al hablar de simulación la primera elección suele darse en el mecanismo de avance del tiempo simulado. Este puede verse de modo continuo, discreto o en forma mixta.

Las simulaciones de esta tesis se corresponden a un avance del tiempo en **eventos discretos**. La granuralidad del detalle con que se implante el modelo definirá la densidad de los eventos. En este sentido deberán adoptarse los compromisos necesarios entre nivel de detalle del modelo y coste computacional de los resultados. En nuestras simulaciones hemos optado por la granuralidad a nivel de mensaje del protocolo bajo estudio.

Otra elección que se plantea, una vez definido que las simulaciones serán de eventos discretos, es el tipo de evolución temporal que se dará a los experimentos de simulación.

En este sentido las opciones son dos:

- Horizonte finito.
- Horizonte infinito.

Los experimentos de horizonte finito se corresponden a simulaciones en las que se persigue caracterizar comportamientos transitorios, evoluciones asociadas al agotamiento de recursos o cumplimiento de plazos.

Los de horizonte infinito se corresponden a simulaciones de sistemas cuyo comportamiento puede considerarse estacionario en el tiempo.

Los objetivos marcados en la sección anterior se corresponden a simulaciones de horizonte infinito. El modelo para este planteamiento no necesita incluir el comportamiento de los elementos asociados a funciones de gestión, de recuperación frente a errores o tratamiento de excepciones por cuanto consideramos la red en estado estacionario.

A.2.3. Selección de lenguajes

Las simulaciones se han realizado empleando programas específicos para este propósito, codificados empleando Modsim, un lenguaje de simulación modular y orientado a objetos. Seguimos el paradigma de simulación de eventos discretos orientado a proceso. Para calcular el final de las simulaciones se ha empleado el método de medias por bloques (batch-mean) con un nivel de confianza del 90 % y un intervalo de confianza del 10 %.

También se ha realizado una implementación de PDP empleando el simulador de red Network Simulator (ns–2). Este simulador permite validar implementaciones de protocolos mas que realizar estudios de simulación sobre sus prestaciones sobre todo en caso de horizonte infinito.

A.3. Implementación de los simuladores

Para el estudio de prestaciones, se han realizado los siguientes simuladores de protocolos:

- Para el modo *pull* se ha construido un simulador.
- Para el modo *push* se ha construido un simulador.
- Para el modo directorio se han construido dos simuladores:
  - dirwn: descubrimiento de servicios basado en directorio en el que la dirección de este directorio es conocida por los dispositivos.
  - dirdesc: descubrimiento de servicios basado en directorio en el que la dirección de este directorio debe descubrirse de forma dinámica, mediante mensajes periódicos de búsqueda por broadcast.
- Para el protocolo SLP se han construido varios simuladores:
  - slpsinDA: SLP sin caché en modo distribuido, sin DA.
  - slpDA: SLP sin caché en modo centralizado, con DA.
  - slpDAdereg: slpDA implementando mensajes SrvDereg.
  - slpcache: SLP con caché en modo distribuido, sin DA.
- Para el protocolo SSDP se ha construido solamente un simulador para comparar sus prestaciones en aplicaciones de tipo "buscador".
- Para el protocolo PDP, partiendo del simulador de *pull*, se han construido varios simuladores para explicar el diseño realizado:
  - pullcache: modo *pull* con caché.
  - pullcachemulticast: modo *pull* con caché y respuestas por difusión.
  - pdpversion1: modo *pull* con caché, respuestas multicast e incluyendo en los mensajes de búsqueda servicios ya conocidos.

- pdpversion2: pdpversion1 en el que se ha añadido que se responde con los servicios almacenados en la caché.
- pdpversion3: pdpversion2 en el que se ha añadido la actualización de la caché con los servicios incluidos en los mensajes de búsqueda.
- pdp: pdpversion3 con mecanismos de consistencia de cachés.

- Para medir diversos aspectos del comportamiento de PDP, se han construido también los siguientes simuladores:

  - pdperrortd: pdpversion3 introduciendo error en el tiempo de disponibilidad anunciado por los dispositivos.
  - pdphistograma: pdp en el que se realizan medidas del porcentaje de mensajes enviados por cada tipo de dispositivo dependiendo de su tiempo de disponibilidad.
  - pdpcachefalsa: pdp en el que los dispositivos se crean con cachés en las que se incluyen servicios falsos, que no están presentes en la red.

El código fuente de todos estos simuladores está accesible en la página web `http://www.it.uc3m.es/celeste/tesis/`.

A.4. Validación de los simuladores

La validación del simulador en los protocolos teóricos, *pull*, *push* y directorio, ha consistido en comprobar que los resultados de simulación coinciden con los resultados obtenidos mediante análisis. Para SLP, se ha comprobado que su comportamiento se ajustaba al de directorio o *pull*, según hubiera DA o no. Respecto al protocolo SSDP, recordar que soporta un modo de funcionamiento *pull* y *push* y no especifica cuando se emplea uno u otro, se deja abierto a las aplicaciones que lo utilicen, es por ello que sólo se hacen simulaciones de este protocolo en la comparativa realizada para el soporte a aplicaciones tipo "buscador". En cuanto al PDP, se ha comprobado que con caché 0 se comportaba como el *pull*. En cualquier caso, siempre se ha razonado sobre los resultados para explicar su comportamiento.

A.5. Cálculo de resultados

Los parámetros de entrada de las simulaciones realizadas son los siguientes:

- Número medio de dispositivos.
- Número de tipos de servicios.
- Para cada dispositivo:
  - Tiempo medio que permanece el dispositivo en la red.
  - Tiempo medio entre búsquedas de servicios.
  - Tamaño de la caché.

Como resultado de las simulaciones hemos seleccionado las siguientes variables:

- **Número de mensajes por búsqueda**: el número de mensajes transmitidos en la red, normalizado al número de búsquedas de servicios.
- **Tasa de descubrimiento de servicios**: el porcentaje del número de servicios descubiertos, respecto al número total de servicios disponibles en la red.
- **Tasa de descubrimientos falsos**: el porcentaje del número de servicios descubiertos que en realidad no están disponibles en la red, respecto al número total de servicios descubiertos.

El escenario de simulación base del que partimos en las simulaciones realizadas es el siguiente:

- Número medio de dispositivos: 20.
- Número de tipos de servicios: 5.
- Para cada dispositivo:
  - Tiempo medio que permanece el dispositivo en la red: 600 segundos.
  - Tiempo medio entre búsquedas de servicios: 60 segundos.
  - Tamaño de la caché: 100.

El tiempo de disponibilidad con el que se anuncian los dispositivos está ajustado al tiempo medio que permanecen los dispositivos en la red, en este caso 600 segundos.

Las simulaciones realizadas muestran como varían los resultados de simulación cuando se modifica alguno de sus parámetros, en cada caso se indica detalladamente el rango de valores para cada uno de ellos. Siempre que no se indique algún parámetro de simulación, se debe considerar el valor indicado en el escenario base.

A.5.1. Precisión de los resultados

De entre los métodos que existen para calcular la precisión con la que estamos estimando un parámetro en una simulación [Pawlikowski, 1990], en esta tesis hemos utilizado el Método de media de bloques (*batch means*), que pasamos a describir a continuación.

El problema de calcular, para un determinado nivel de confianza, que intervalo de confianza tiene el resultado que se está obteniendo en la simulación, viene de que la secuencia de valores de salida de la simulación están correlados.

El método de media de bloques se basa en la suposición de que las observaciones más separadas en el tiempo tienen menos correlación. De esta manera, para bloques de observaciones suficientemente largos, las medias de los bloques estarían prácticamente incorreladas.

En el método de media de bloques, la secuencia de n observaciones $x_1, x_2, \ldots, x_n$ es dividida en k bloques de m observaciones, con $k = n/m$.

Las medias de cada uno de estos bloques, $\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_k$, son usados como datos de salida secundarios para el análisis de estadístico de los resultados.

La media es estimada como

$$\hat{X} = \frac{1}{k} \sum_{i=1}^{k} \hat{X}_i \tag{A.1}$$

El intervalo de confianza para un nivel de confianza $100(1-\alpha)\,\%$ es

$$\hat{X} \pm t_{k-1,1-\alpha/2} \hat{\sigma}[\hat{X}] \tag{A.2}$$

donde $t_{k-1,1-\alpha/2}$ es el punto crítico superior $(1 - \frac{\alpha}{2})$ obtenido de la función t de Student con k-1 grados de libertad.

El problema es calcular el valor del tamaño de bloque (m) que asegure la incorrelación de las medias de los bloques.

En este método hay dos procedimientos: primero probar secuencialmente hasta encontrar un tamaño aceptable de bloque, y segundo probar secuencialmente la precisión de los resultados, hasta alcanzar el nivel prefijado, momento en que se detendrá la simulación.

Un determinado tamaño de bloque es aceptado como el correcto si todos los coeficientes de autocorrelación son estadísticamente despreciables para un nivel significativo, $\beta_k, 0 < \beta_k < 1$.

Los coeficientes de autocorrelación los estimamos mediante los llamados estimadores *jackknife*, que suelen ser los menos sesgados. Los estimadores *jackknife* del coeficiente de autocorrelación de retraso k para una secuencia de medias de bloques de tamaño m se calcula según la fórmula:

$$\hat{\hat{r}}(k,m) = 2\hat{r}(k,m) - \frac{\hat{r}'(k,m) + \hat{r}''(k,m)}{2} \tag{A.3}$$

donde los estimadores de la derecha se calculan como coeficientes de autocorrelación ordinarios, exceptuando que $\hat{r}(k,m)$ se estima sobre las k medias de bloques, mientras que $\hat{r}'(k,m)$ y $\hat{r}'(k,m)$ se estiman respectivamente sobre la primera y la segunda mitad del bloque analizado.

El método requiere varios parámetros:

- n_{max}: el número máximo permitido de observaciones para la simulación (el valor por defecto es $n_{max} = 10^7$).
- n_o: el número de observaciones del periodo transitorio inicial que han sido descartadas previamente (el valor por defecto es $n_o = 0$). Este valor se obtiene aplicando previamente un procedimiento para la detección del periodo transitorio inicial.
- $(1-\alpha)$: el nivel de confianza prefijado para los resultados finales ($0 < \alpha < 1$; el valor por defecto es $\alpha = 0,05$).
- e_{max}: el valor máximo aceptable de la precisión relativa del intervalo de confianza ($0 < e_{max} < 1$; el valor por defecto es $e_{max} = 0,05$).

Los resultados de simulación que se presentan en la tesis han sido todos obtenidos mediante este método con un nivel de confianza del 90 % y un intervalo de confianza del 10 %.

Apéndice B

Descripción detallada de Pervasive Discovery Protocol

En este anexo se proporciona una descripción detallada del protocolo de descubrimiento de servicios propuesto en esta tesis, para facilitar su implementación. Pervasive Discovery Protocol, es en sí un algoritmo de descubrimiento adaptado a entornos de computación ubicua, basado en una arquitectura distribuida, que propone un nuevo comportamiento de los clientes, a la hora de enviar peticiones de búsquedas de servicios, y de los servidores, a la hora de generar las correspondientes respuestas. Por lo tanto, este algoritmo es aplicable a otros protocolos de descubrimiento de servicios con arquitectura distribuida, sin necesidad de realizar cambios significativos en la estructura de los mensajes, y totalmente independiente de cómo se realice la descripción de los servicios y de cómo se detecta si el servicio solicitado corresponde con alguno de los servicios ofrecidos.

Para realizar esta descripción de PDP, nos hemos basado en la RFC del protocolo SLP v2, centrándonos en su funcionamiento distribuido e incluyendo los mínimos cambios en el formato de los paquetes y reutilizando la descripción de servicios propuesta, basada en URLs y especificada en la RFC 2609. Aclarar, que la especificación aquí proporcionada es autocontenida y no es necesario recurrir a la RFC de SLP para implementar el protocolo.

El motivo por el que hemos seleccionado SLP como base para nuestra descripción es que a día de hoy, es un estándar de Internet, el resto de propuestas que hemos estudiado en el capítulo 3 se encuentran en desarrollo o no han obtenido el consenso suficiente para pasar del estado de draft.

Para facilitar la implementación de PDP a partir de la especificación, se proporcionan diagramas de flujo para describir el comportamiento de clientes y ser-

vidores. Estos diagramas se han realizado basándonos en el lenguaje de especificación SDL, aunque en ningún momento se da una descripción completa del protocolo empleado este lenguaje, ya que no era uno de los objetivos marcados en esta tesis.

B.1. Terminología

- Agente de Usuario PDP (PDP_UA): Proceso encargado de localizar servicios a las aplicaciones del usuario. Los PDP_UAs obtienen esta información de los servicios locales del dispositivo, de su caché de servicios o a través de Agentes PDP de Servicio remotos.
- Agente de Servicio PDP (PDP_SA): Proceso encargado de anunciar los servicios conocidos por el dispositivo. Consideramos servicios conocidos tanto los servicios locales como los que se almacenan en la caché de servicios.
- Caché de servicios: Cada dispositivo tiene asociada una caché en la que almacena todos los servicios remotos conocidos. A lo largo del documento utilizaremos el termino caché para designar a este elemento.

  Si el dispositivo tiene varias interfaces de red, tendrá una caché asociada a cada interfaz de forma independiente, de manera que en una caché sólo se almacenan servicios que han sido anunciados en el interfaz de red asociado.
- Tiempo de disponibilidad: Cada dispositivo tiene asociado un tiempo de disponibilidad dependiendo de sus características de movilidad. Consideramos que es una estimación del tiempo que ese dispositivo permanecerá fijo en un determinado lugar. Un dispositivo anunciará los servicios que conoce con un tiempo de vida menor o igual a su tiempo de disponibilidad.
- Tiempo de expiración: Tiempo asociado a la entrada de un servicio en la caché. Será "t" segundos después de ser almacenado, siendo "t" el valor mínimo entre el tiempo de disponibilidad con el que se anuncia un servicio, y el tiempo de disponibilidad del dispositivo en el que se almacena su descripción. Una vez finalizado este tiempo, la entrada del servicio correspondiente debe eliminarse de la caché.
- Tipo de servicios: Cada tipo de servicio está definido por un único String.
- SLP: Service Location Protocol (RFC 2608).
- URL: Universal Resource Locator. Se emplearán para describir servicios, siguiendo el formato empleado en SLP: `Service URL`.

B.2. Descripciones de servicios

En esta versión de PDP se emplea el mismo formato de descripción de servicios que utiliza SLP, denominado `Service URL`, cuya sintáctica y semántica se definen en la RFC 2609. Los mensajes `PDPSrvRply` y `PDPDeReg` incluyen estas `Service URL` encapsulada dentro de lo que se denomina `Service Entry`.

En esta sección describimos brevemente el formato de `Service URL` proporcionando algún ejemplo ilustrativo y el de las `Services Entries` que emplea PDP.

B.2.1. Service URL

Una `Service URL` es de la siguiente forma:

```
``service:''<service-type>``://''<addrspec>
```

El tipo de servicio está contenido en `<service-type>` entre los Strings `service:` y `://`. Los tipos de servicios se definen según `Service Templates` (RFC 2609) que proporcionan el comportamiento, los atributos y sus posibles valores, de un servicio.

En la RFC 2609 se distinguen tipos abstractos que tienen la forma siguiente:

```
service:<abstract-type>:<concrete-type>
```

En los `PDPSrvRqst` las búsquedas de servicios se realizan indicando en `<service-type>` que puede ser un tipo concreto o un tipo abstracto. Así por ejemplo si buscamos el tipo de servicio abstracto `service:printer`, tanto el servicio:

```
service:printer:lpr://deviceaddress
```

como

```
service:printer:http://deviceaddress
```

serían servicios válidos como respuesta, mientras que si el servicio que buscamos debe ser además del tipo concreto `lpr`, `service:printer:lpr`, sólo el primero de ellos será válido.

El campo <addrspec> es el nombre del dispositivo (host) o en su defecto su dirección IP, seguido de forma opcional por ":" y el número de puerto. En este campo también pueden incluirse atributos del servicio en formato `attribute-id = value` separados por ";" siempre que éstos sean imprescindibles para acceder al servicio.

B.2.2. Service Entries

PDP almacena las descripciones de servicios en elementos del protocolo denominadas `Service Entries`, Figura B.1, que incluyen:

- `Flags`: `NO_CACHING` (`0x8000`) para indicar que el servicio no debe ser almacenado en caché, por defecto estará desactivado. El resto de bits reservados deben estar a 0.
- `Lifetime`: tiempo de vida asociado al servicio. La longitud de este campo es de 2 bytes y por lo tanto, su valor máximo es aproximadamente 18 horas[1].
- `Service Description Length`: longitud del campo `Service Description`. Tiene un tamaño de 2 bytes.
- `Service Description`: descripción del servicio. La descripción del servicio es una URL siguiendo el formato descrito en el apartado anterior. Este campo tiene un tamaño variable.

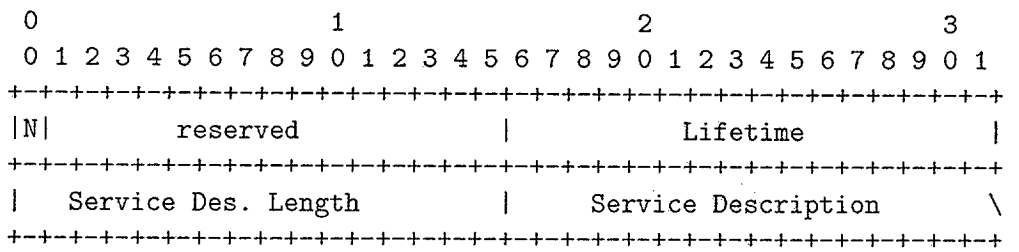


Figura B.1: Formato de Service Entries en PDP.

[1]Esto implica que el tiempo máximo que puede almacenarse un servicio remoto en la caché es 18 horas. Un dispositivo con un tiempo de disponibilidad mayor de 18 horas, anunciará sus servicios locales con un tiempo de vida igual a `0xFFFF`

B.3. Almacenamiento de servicios en cachés

Los agentes PDP tienen asociadas una caché con servicios remotos conocidos. Esta caché es actualizada por los PDP_UA a través de `PDPSrvRply` escuchadas en la red, y que corresponden con respuestas a `PDPSrvRqst` que se han realizado anteriormente en el entorno, tanto por el propio dispositivo, como por otros que le rodean.

Cada entrada en la caché está formado por los siguientes campos:

- Descripción del servicio: se corresponde con una `Service URL`, en la que se incluye información sobre el tipo de servicio y la localización del mismo.
- Tiempo de expiración: tiempo en el que la entrada en la caché debe borrarse. Este valor se calcula sumando al tiempo actual, el tiempo de vida asociado a esta entrada en la caché, que es el mínimo entre el `Lifetime` con el que se anuncia el servicio y el tiempo de disponibilidad del dispositivo en cuya caché se está almacenando la descripción.

Por ejemplo, un PDP_SA envía un `PDPSrvRply` con los siguientes datos de un servicio `net-transducer:thermometer`:

```
Service Description = service:net-transducer:
                      thermometer://v33.test/ports=3211

Lifetime = 10 horas
```

El PDP_UA de una PDA que tiene un tiempo de disponibilidad de 5 horas, al escuchar este mensaje, almacenará la siguiente entrada en su caché:

Descripción servicio	Tiempo de expiración
service:net-transducer: thermometer://v33.test/ports=3211	<tiempo sistema> + 5 horas

En cambio el PDP_UA de un aire acondicionado que tiene un tiempo de disponibilidad de 5 años, al escuchar este mensaje, almacenará la siguiente entrada en su caché:

Descripción del servicio	Tiempo de expiración
service:net-transducer: thermometer://v33.test/ports=3211	<tiempo sistema> + 10 horas

B.4. Descripción del protocolo

Pervasive Discovery Protocol es un protocolo de descubrimiento con arquitectura distribuida, que combina los métodos "push" y "pull", y en el que los clientes tienen una caché local para almacenar anuncios de servicios remotos.

La implementación de PDP consta de un PDP_UA o un PDP_SA o ambos. Será habitual que los dispositivos ofrezcan servicios y a su vez los demanden, de tal forma que la mayoría de ellos poseerán tanto un PDP_UA como un PDP_SA, Figura B.2.

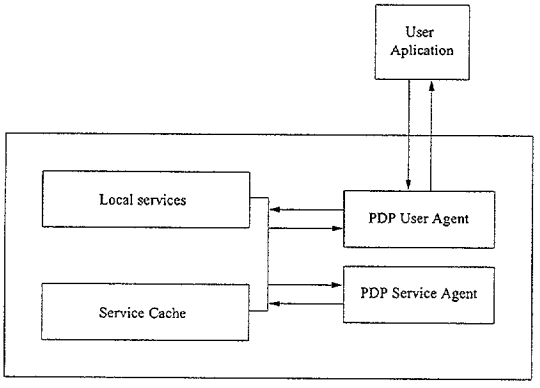


Figura B.2: Dispositivo PDP con PDP_UA y PDP_SA.

En dispositivos limitados y de función específica, como son los sensores y actuadores, lo habitual será que sólo posean un PDP_SA, que les permitirá anunciar sus servicios en el entorno que les rodea, Figura B.3.

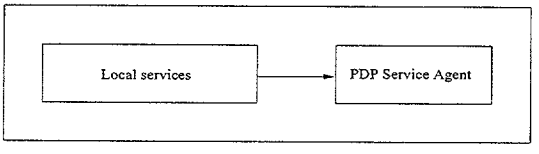


Figura B.3: Dispositivo PDP con PDP_SA.

Del mismo modo, podemos pensar que determinados dispositivos, como dispositivos personales del usuario, pueden poseer sólo un PDP_UA, ya que no tienen

o no quieren ofrecer servicios, Figura B.4.

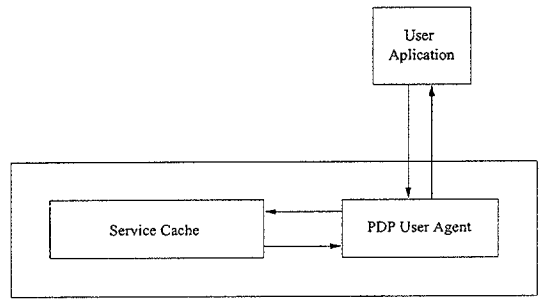


Figura B.4: Dispositivo PDP con PDP_UA.

Los PDP_UA envían mensajes `PDPSrvRqst` solicitando un tipo de servicio, y los PDP_SA que conocen servicios de este tipo responden enviando un mensaje `PDPSrvRply` de respuesta. Ambos mensajes se transmiten por multicast/broadcast, Figura B.5. En PDP los mensajes de respuesta `PDPSrvRply` se transmiten por difusión con dos objetivos: que se actualice la caché de servicios remotos en todos los dispositivos que escuchen el mensaje y que se minimice el número de mensajes de respuesta a una búsqueda, ya que un PDP_SA sólo responde a una búsqueda, si conoce algún servicio que no se ha incluido en respuestas anteriores, realizadas por otros PDP_SA.

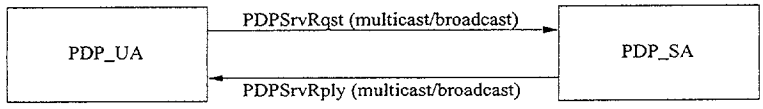


Figura B.5: Interacción entre PDP_UAs y PDP_SAs.

B.4.1. Búsquedas de un servicio concreto

En PDP existen dos tipos de búsquedas para localizar un servicio de un tipo determinado en la red. Dependiendo de qué tipo de búsqueda se realiza, el comportamiento de los PDP_UA y PDP_SA se modifica, con el objetivo de minimizar el ancho de banda consumido.

Una petición - una repuesta

En este tipo de búsqueda, a la petición de un tipo de servicio se responde, si existe, con un sólo servicio del tipo solicitado.

El PDP_UA busca primero el servicio entre los servicios locales, después entre los servicios almacenados en la caché y por último, si no lo ha encontrado, transmite un `PDPSrvRqst` por multicast/broadcast. Un PDP_SA que escuche este mensaje y ofrezca el servicio genera el correspondiente `PDPSrvRply`, escucha la red y si nadie ha respondido previamente a esta búsqueda, lo transmite por multicast/broadcast. Los demás PDP_SA, que iban a responder, descartan sus `PDPSrvRply` en cuanto escuchen esta respuesta.

Hemos introducido este tipo de búsquedas para aquellos tipos de servicio en los que el servicio obtenido es independiente del servidor que lo ofrece. Un ejemplo es obtener la temperatura.

Una petición - varias respuestas

A la petición de un tipo de servicio se responde, si existen, con todos los servicios de ese tipo encontrados en la red.

El coste de este tipo de búsquedas, en cuanto a consumo de ancho de banda, es mayor que el anterior. Para minimizarlo en PDP se introducen técnicas de supresión de respuestas duplicadas. El PDP_UA, al que se le solicita la búsqueda, construye una lista con los servicios del tipo solicitado conocidos, es decir, aquellos que existen en local o están almacenados en su caché, e incluye esta lista en un mensaje `PDPSrvRqst` que transmite por multicast/broadcast. Un PDP_SA que escuche este mensaje construirá una lista con los servicios de este tipo que tiene tanto de forma local como en la caché, y que no están en la lista incluida en el `PDPSrvRqst`, si nadie ha respondido previamente a esta búsqueda, lo transmite por multicast/broadcast. Los demás PDP_SA que conocen también nuevos servicios descartan sus `PDPSrvRply` si las respuestas que han dado previamente otros PDP_SA, ya incluyen los servicios que iba a anunciar él en su mensaje de respuesta.

Hemos introducido este tipo de búsqueda para aquellos tipos de servicio en los que el servicio obtenido puede ser distinto dependiendo del servidor que lo ofrece. Un ejemplo es un servicio de impresión: al usuario le gusta seleccionar, dependiendo del tipo de trabajo que va a imprimir, que tipo de impresora utilizar: a color o en blanco y negro, a doble cara o una sola cara, etc. Por ello es importante proporcionarle varias impresoras para que seleccione la apropiada.

B.4.2. Búsquedas de todos los servicios

En PDP se ha introducido un tipo especial de servicio denominado `service:all`, para facilitar que aplicaciones tipo "buscador" puedan solicitar a su PDP_UA una búsqueda de todos los servicios, de cualquier tipo, disponibles en el entorno.

Un PDP_UA al que se le solicita un búsqueda de `service:all` construye una lista con los servicios conocidos, es decir, aquellos que existen en local o están almacenados en su caché, e incluye esta lista en un mensaje `PDPSrvRqst` que transmite por multicast/broadcast. Un PDP_SA que escuche este mensaje construirá una lista con los servicios que tiene tanto de forma local como en la caché, y que no están en la lista incluida en el `PDPSrvRqst`, si nadie ha respondido previamente a esta búsqueda, lo transmite por multicast/broadcast. Los demás PDP_SA, que conocen también nuevos servicios, descartan sus `PDPSrvRply` si las respuestas que han dado previamente otros PDP_SAs ya incluyen los servicios que iba a anunciar él en su mensaje de respuesta.

B.4.3. PDP User Agent

El PDP_UA es el responsable de localizar servicios disponibles en la red y mantener actualizada la caché de servicios remotos asociada a un dispositivo. Debe generar mensajes `PDPSrvRqst` si procede, y procesar los `PDPSrvRply` que se transmiten en la red. Opcionalmente, puede procesar `PDPDeReg` para eliminar servicios almacenados en la caché que dejan de estar disponibles.

En este apartado describimos de forma detallada los procesos que lleva a cabo un PDP_UA dentro del protocolo PDP. Se incluye varios diagramas de flujo para facilitar la implementación.

Búsqueda de un servicio concreto

Describimos el comportamiento del PDP_UA dependiendo si el tipo de búsqueda es "una petición - una respuesta" o "una petición - varias respuestas".

- **Una petición - una respuesta**

  Cuando se le solicita al PDP_UA la búsqueda de un servicio concreto, del tipo "una petición - una respuesta" realiza los siguientes pasos, Figura B.6:

  - Busca si existe un servicio local del tipo solicitado, si es así, proporciona la descripción del servicio a la aplicación.

- Si no es un servicio local, busca si existe un servicio en la caché del tipo solicitado, si es así, proporciona la descripción del servicio almacenada a la aplicación.
- Si no es un servicio conocido, envía un `PDPSrvRqst` por broadcast/multicast y espera durante un tiempo (`CONFIG_WAIT_RPLY`) a que se produzcan los siguientes eventos:
  - Recepción de un `PDPSrvRply` correspondiente al `PDPSrvRqst` enviado: responde a la petición de búsqueda con el servicio incluido en este mensaje de respuesta.
  - Finalización del tiempo de espera: no se ha obtenido mensaje de respuesta, y por lo tanto responde a la petición de búsqueda indicando que el servicio no está disponible.

- **Una petición - varias respuestas**

  Cuando se le solicita al PDP_UA la búsqueda de un servicio concreto, del tipo "una petición - varias respuestas" realiza los siguientes pasos, Figura B.7:

  - Construye una lista con todos los servicios del tipo solicitado conocidos, es decir, tanto los que se proporcionan localmente como los que tiene almacenados en la caché.
  - Envía un `PDPSrvRqst` por broadcast/multicast incluyendo la lista de servicios conocidos y espera durante un tiempo (`CONFIG_WAIT_RPLY`) a que se produzcan los siguientes eventos:
    - Recepción de un `PDPSrvRply` correspondiente al `PDPSrvRqst` enviado: actualiza la lista de servicios conocidos con los servicios incluidos en este mensaje de respuesta y sigue esperando a que nuevos eventos se produzcan.
    - Finalización del tiempo de espera:
      - Si existe algún servicio en la lista de servicios conocidos, responde a la petición de búsqueda con esta lista.
      - Si no existe ningún servicio en la lista de servicios conocidos, responde a la petición de búsqueda indicando que el servicio no está disponible.

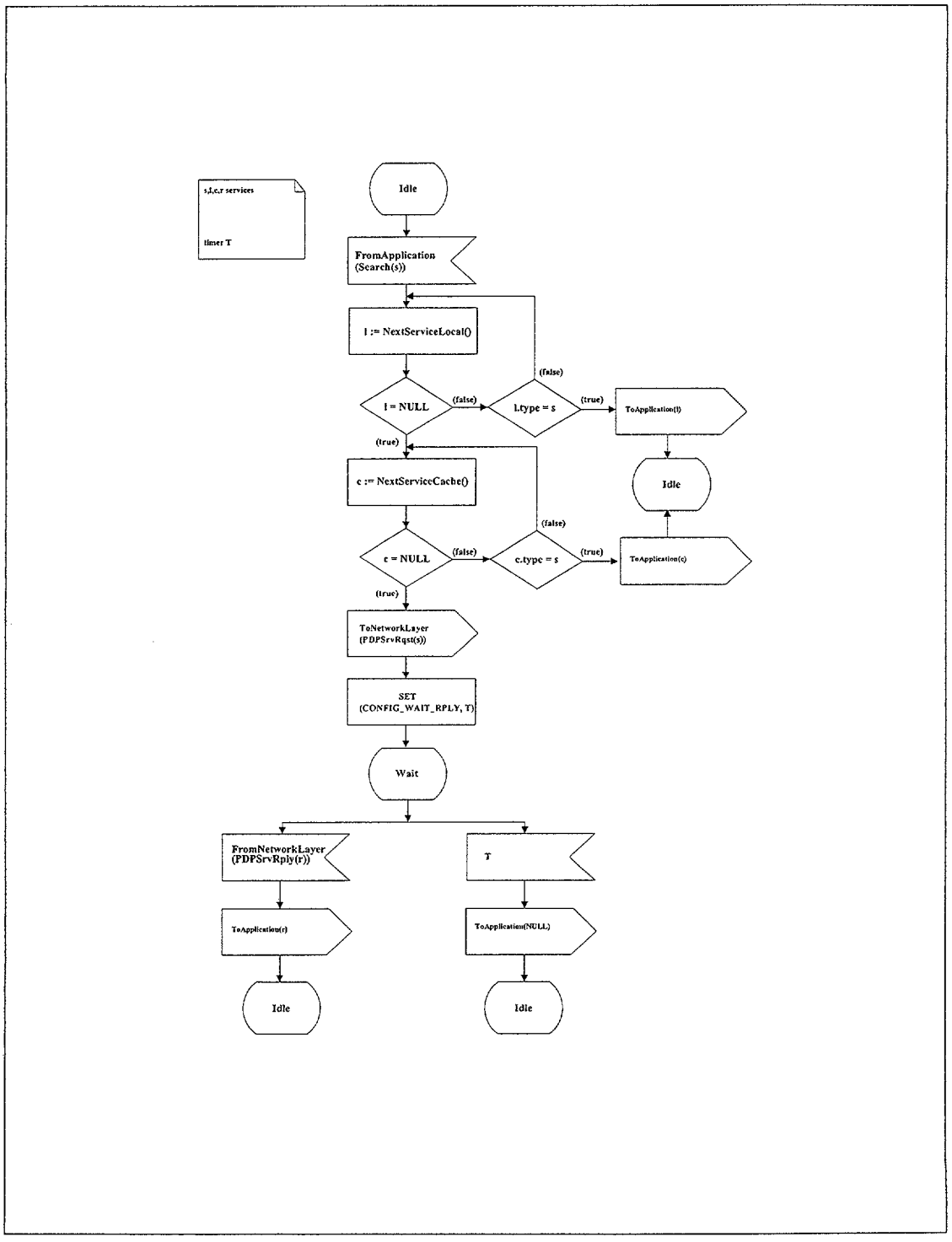


Figura B.6: Diagrama Agente de Usuario PDP: proceso de búsqueda una petición - una respuesta.

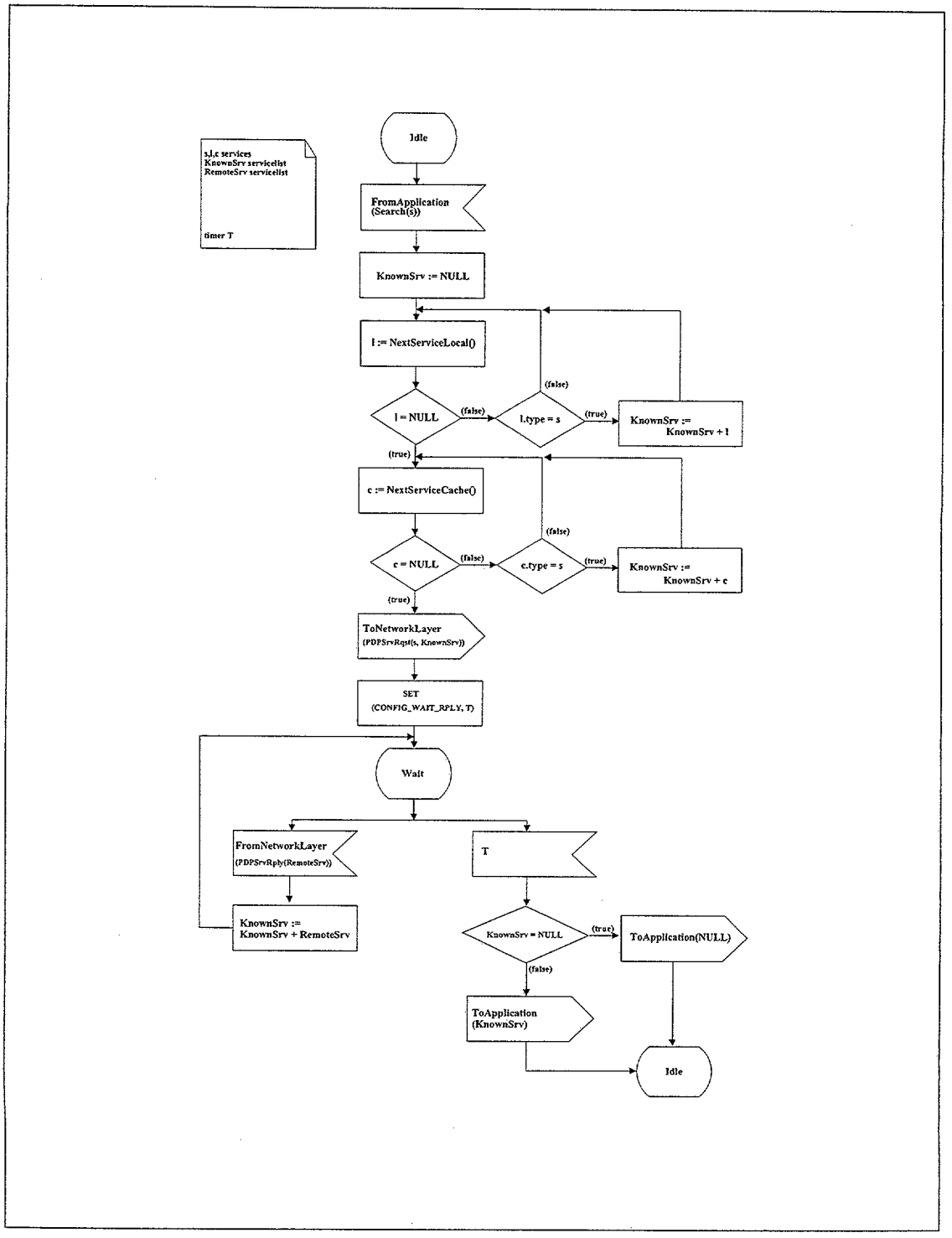


Figura B.7: Diagrama Agente de Usuario PDP: proceso de búsqueda una petición - varias respuestas.

Búsqueda de todos los servicios

Siempre que una aplicación le solicite una búsqueda de servicios de tipo `service:all`, el PDP_UA debe realizar los siguientes pasos, Figura B.8:

- Construye una lista con todos los servicios conocidos (de cualquier tipo), es decir, tanto los que se proporcionan localmente como los que tiene almacenados en la caché.
- Envía un `PDPSrvRqst (service:all)` por broadcast/multicast incluyendo la lista de servicios conocidos y espera durante un tiempo (`CONFIG_WAIT_RPLY`) a que se produzcan los siguientes eventos:
  - Recepción de un `PDPSrvRply` correspondiente al `PDPSrvRqst` enviado: actualiza la lista de servicios conocidos y sigue esperando a que nuevos eventos se produzcan.
  - Finalización del tiempo de espera:
    - Si existen servicios conocidos en la lista, responde a la petición de búsqueda con esta lista.
    - Si no, responde a la petición de búsqueda indicando que no existen servicios disponibles.

Actualización de caché

El PDP_UA además de realizar búsquedas de servicios, debe procesar todas las `PDPSrvRply` que recibe, aunque NO se correspondan a `PDPSrvRqst` transmitidas por él. Con las descripciones de servicios almacenadas en estos mensajes actualiza los contenidos de su caché.

La caché tiene un tamaño limitado, que puede ser configurado en el dispositivo dependiendo de sus restricciones de memoria. El PDP_UA debe gestionar que no se sobrepase este tamaño máximo, de forma que cuando se intenta almacenar una nueva descripción de servicio en la caché y está llena, se debe eliminar la entrada con un tiempo de expiración menor para introducir esta nueva entrada. El PDP_UA debe también eliminar las entradas de la caché que han expirado.

B.4.4. PDP Service Agent

El PDP_SA es el responsable de anunciar los servicios disponibles y conocidos por un dispositivo. Y debe interpretar mensajes `PDPSrvRqst` y generar el corres-

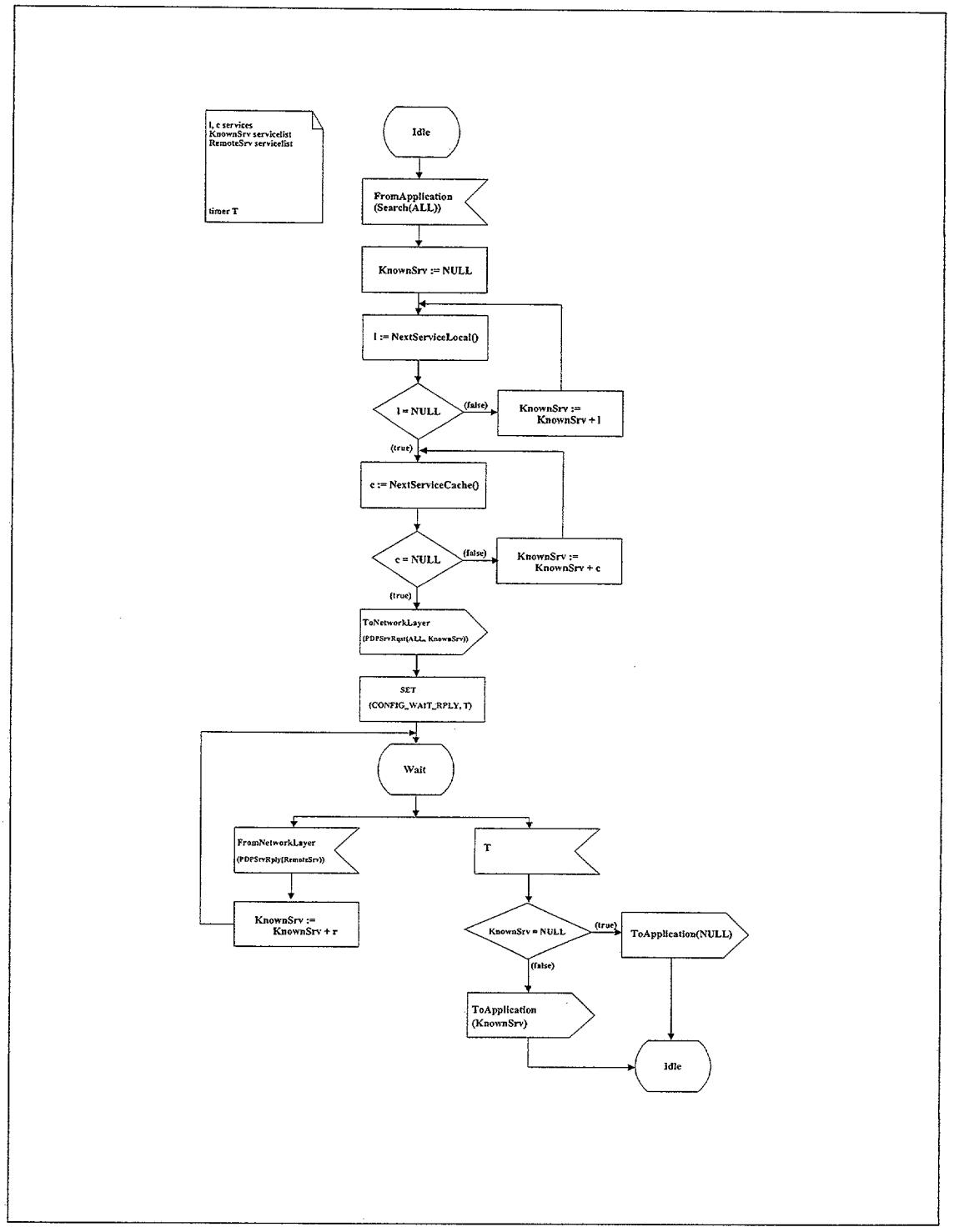


Figura B.8: Diagrama Agente de Usuario PDP: proceso de búsqueda de todos los servicios.

pondiente PDPSrvRply si procede. Opcionalmente, puede generar PDPDeReg para indicar que sus servicios locales dejan de estar disponibles.

En este apartado describimos de forma detallada los procesos que lleva a cabo un PDP_SA dentro del protocolo PDP. Se incluye varios diagramas de flujo para facilitar la implementación.

Respuesta a una búsqueda de un servicio concreto

Describimos el comportamiento del PDP_SA dependiendo si el tipo de búsqueda que recibe es "una petición - una respuesta" o "una petición - varias respuestas".

- **Una petición - una respuesta**

  Cuando un PDP_SA recibe un PDPSrvRqst correspondiente a la búsqueda de un servicio concreto, del tipo "una petición - una respuesta", realiza los siguientes pasos, Figura B.9:

  - Comprueba si el servicio solicitado es uno de sus servicios locales:
    - Si es así, genera de forma aleatoria un tiempo T, inversamente proporcional al tiempo disponibilidad del dispositivo; de tal forma que responderá con mayor probabilidad, aquel dispositivo que puede ofrecer el servicio durante un mayor periodo de tiempo.
    - Si no es, no responde a la búsqueda y finaliza.
  - Espera durante el tiempo T a que se produzca alguno de los siguientes eventos:
    - Recepción de un PDPSrvRply correspondiente al PDPSrvRqst : descarta su mensaje de respuesta.
    - Finalización del tiempo de espera: envía un PDPSrvRply con el servicio local proporcionado.

- **Una petición - varias respuestas**

  Cuando un PDP_SA recibe un PDPSrvRqst correspondiente a la búsqueda de un servicio concreto, del tipo "una petición - varias respuestas", realiza los siguientes pasos, Figura B.10:

  - Construye una lista de respuesta con todos los servicios del tipo solicitado conocidos, es decir, tanto los que se proporcionan localmente

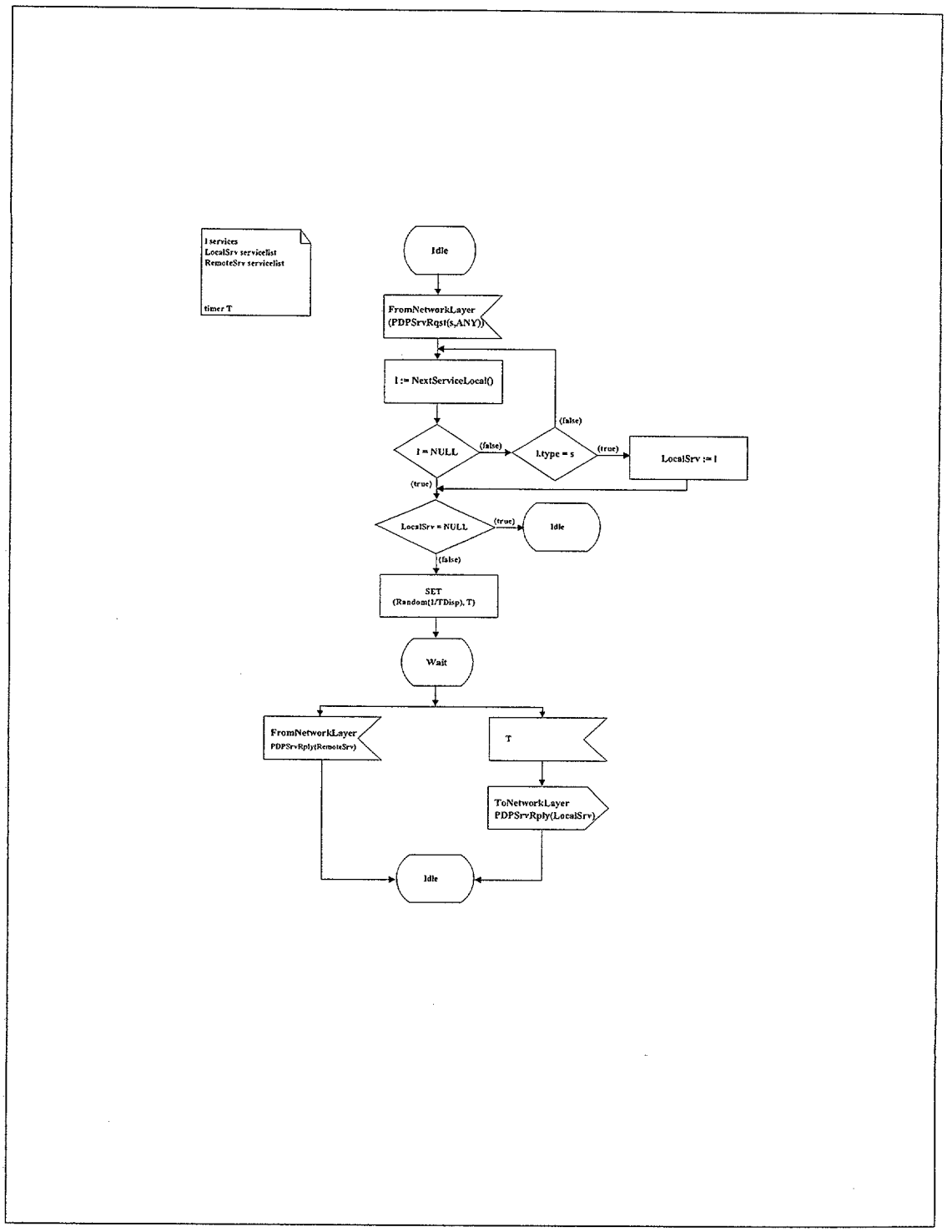


Figura B.9: Diagrama Agente de Servicio PDP: respuesta búsqueda una petición - una respuesta.

como los que tiene almacenados en la caché. De esta lista de respuesta se eliminan los servicios que se envían como conocidos en el mensaje `PDPSrvRqst` recibido:

- Si la lista de respuesta contiene servicios, genera de forma aleatoria un tiempo `T`, inversamente proporcional al tiempo de disponibilidad del dispositivo y al tamaño, en número de entradas, de la lista de respuesta.
- Si esta lista está vacía, no responde a la búsqueda y finaliza.

- Espera durante el tiempo `T` a que se produzca alguno de los siguientes eventos:
  - Recepción de un `PDPSrvRply` correspondiente al `PDPSrvRqst` : comprueba si los servicios incluidos en esta respuesta están en su lista de respuesta, si es así los elimina de ella. Si su lista de respuesta se queda vacía descarta su propia `PDPSrvRply` y finaliza.
  - Finalización del tiempo de espera: envía un `PDPSrvRply` con la lista de respuesta y finaliza.

Respuesta a una búsqueda de todos los servicios

Cuando un PDP_SA recibe un `PDPSrvRqst` correspondiente a la búsqueda de un servicio del tipo `service:all`, realiza los siguientes pasos, Figura B.11:

- Construye una lista de respuesta con todos los servicios conocidos, es decir, tanto los que se proporcionan localmente como los que tiene almacenados en la caché. De esta lista de respuesta se eliminan los servicios que se envían como conocidos en el mensaje `PDPSrvRqst` recibido:
  - Si la lista de respuesta contiene servicios, genera de forma aleatoria un tiempo `T`, inversamente proporcional al tiempo de disponibilidad del dispositivo, y al tamaño, en número de entradas, de la lista de respuesta.
  - Si esta lista está vacía, no responde a la búsqueda y finaliza.
- Espera durante el tiempo `T` a que se produzca alguno de los siguientes eventos:
  - Recepción de un `PDPSrvRply` correspondiente al `PDPSrvRqst` : comprueba si los servicios incluidos en esta respuesta están en su lista de respuesta, si es así los elimina de ella. Si su lista de respuesta se queda vacía descarta su propia `PDPSrvRply` y finaliza.

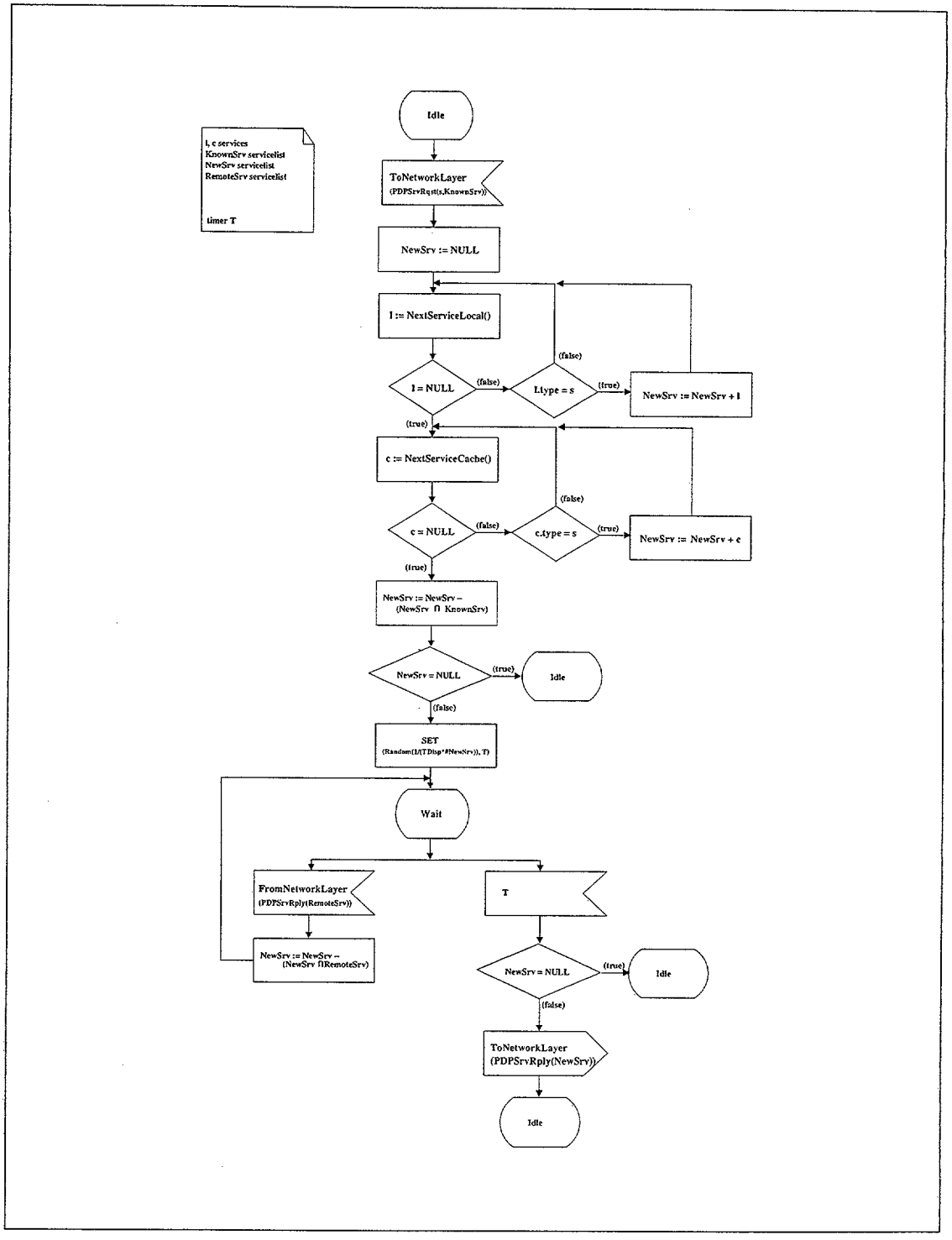


Figura B.10: Diagrama Agente de Servicio PDP: respuesta búsqueda una petición - varias respuestas.

- Finalización del tiempo de espera: envía un `PDPSrvRply` con su lista de respuesta y finaliza.

Anuncio de servicio almacenados en la caché

Cuando un PDP_SA responde a un `PDPSrvRqst` incluyendo descripciones de servicios almacenadas en la caché, el valor del campo `Lifetime` de la `Service Entry` construida a partir de la descripción de servicio almacenada en la caché, contiene el resultado de restar al tiempo de expiración de esta entrada el tiempo actual.

B.5. Consideraciones sobre múltiples interfaces

Si un dispositivo tiene interfaces a distintas redes, y la pila IP no soporta el encaminamiento entre estas redes, los servicios que se descubren por un determinado interfaz no serán accesibles desde la red conectada a los demás interfaces. Por ejemplo, si una PDA tiene dos interfaces, una Bluetooth y otra 802.11b, los servicios que se descubran en la red Bluetooth no son accesibles para otros dispositivos que estén en la 802.11b. El grupo MANET del IETF está estudiando el encaminamiento entre redes ad-hoc móviles, pero éste es un problema todavía no resuelto.

Por lo tanto, si un dispositivo tiene interfaz a dos o más redes, debe separar las cachés en la que almacena los servicios descubiertos en cada una de ellas, y responder a los `PDPSrvRqst` sólo con sus servicios locales y los que existan en la caché asociada a ese interfaz.

B.6. Uso de puertos, UDP y multicast

Los PDP_UAs deben transmitir peticiones por multicast y recibir respuestas por multicast. Los PDP_SAs deben aceptar peticiones multicast y transmitir respuestas por multicast.

Todos los mensajes PDP se transmiten usando el protocolo de transporte UDP. Si un mensaje de respuesta PDP excede el tamaño máximo de un datagrama UDP, se debe fragmentar en tantos mensajes PDP como sea necesario y enviarlo en los correspondientes datagramas.

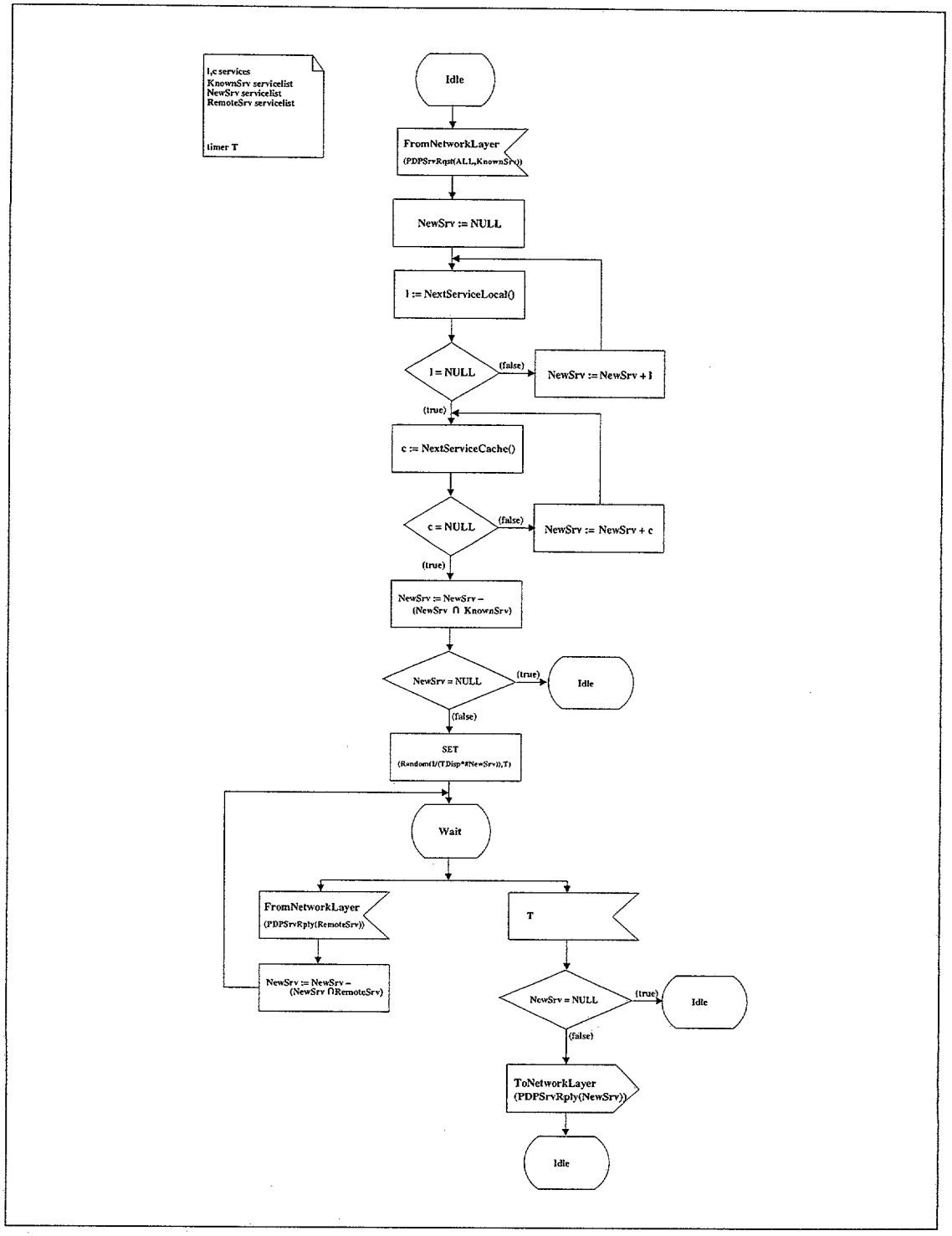


Figura B.11: Diagrama Agente de Servicio PDP: respuesta búsqueda todos los servicios.

Todos los mensajes de petición PDP deben llevar de puerto destino, el puerto reservado para PDP, el puerto que se está utilizando actualmente es el 3000 (sin asignar por el IANA). Todos los mensajes PDP deben enviarse a la dirección multicast 239.255.255.253. El TTL por defecto es 255.

En redes aisladas se puede utilizar broadcast en lugar de multicast.

B.7. Errores

En PDP todos los mensajes se envían por multicast/broadcast, tanto si son peticiones de servicios como si son respuestas a estas peticiones, y además todos los PDP_UA deben analizar todos los `PDPSrvRqst` para actualizar la caché de servicios.

Éste es el motivo por el que tanto los PDP_SA como los PDP_UA deben descartar mensajes erróneos, sin enviar ningún mensaje en el que se indique el error producido. Del mismo modo, un PDP_SA siempre debe incluir algún servicio en un `PDPSrvRply`, si no es así no debe enviar la respuesta.

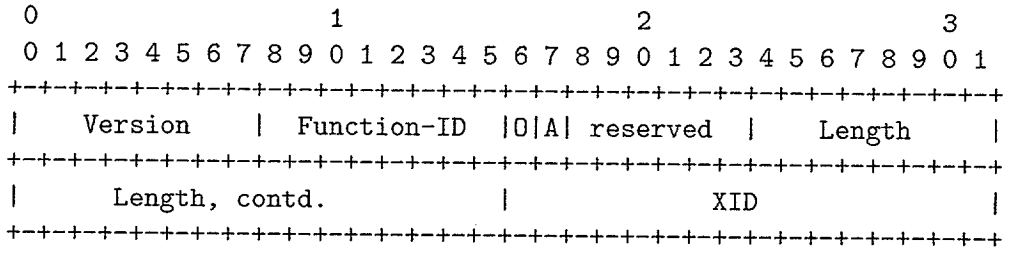
La idea de PDP es que los dispositivos aprendan de lo que se produce a su alrededor y que este aprendizaje les sirva para minimizar el número de transmisiones que deben realizar para localizar un nuevo servicio. El hecho de que no exista un servicio en un momento determinado, que el mensaje tenga un formato erróneo o que no se haya incluido un campo obligatorio, no es una información que interese al resto de dispositivos de la red y por lo tanto, no debe transmitirse.

B.8. Mensajes PDP obligatorios

En este sección describimos de forma detallada el formato de mensajes de PDP. Las implementaciones de PDP_UA y PDP_SA deben soportar obligatoriamente los mensajes `PDPSrvRqst` y `PDPSrvRply`.

Todos los mensajes PDP tienen una cabecera común con los siguientes campos, Figura B.12:

- `Version`: número de versión del protocolo. En la actualidad 1.0. Tamaño del campo un 1 byte.
- `Function ID`: identificador de tipo de mensaje. Tamaño del campo 1 byte.



Tipo de Mensaje	Abreviatura	Function-ID
PDP Service Request	PDPSrvRqst	1
PDP Service Reply	PDPSrvRply	2
PDP Service Deregister	PDPDeReg	3

Figura B.12: Cabecera de mensajes PDP.

- Flags: Tamaño del campo 1 byte. Los bits que no están reservados deben estar puestos a 0. Existen dos bits reservados en PDP:
  - OVERFLOW (0x80) se activa cuando la longitud del mensaje es mayor que lo que puede contener un datagrama.
  - ANY (0x60) se activa cuando el tipo de búsqueda es "una petición - una respuesta". En los mensajes PDPSrvRply este bit debe estar a 0 así como en los mensajes PDPSrvRqst en el que el tipo de servicio buscado es service:all.
- Length: longitud en bytes del mensaje PDP completo, incluida la cabecera. Tamaño del campo 3 bytes.
- XID: tiene un valor único para cada petición enviada. Los mensajes de respuesta incluyen el mismo xid que la petición correspondiente. El mensaje PDPDeReg se envía con un XID de 0. Tamaño del campo 2 bytes.

B.8.1. PDP Service Request

El PDP_UA envía un mensaje PDP Service Request para realizar búsquedas de servicios en la red.

Los campos contenidos en este mensaje son los siguientes, Figura B.13:

- Known Service Entry count: número de Known Service Entries contenidas en el mensaje. Tamaño del campo 2 bytes.

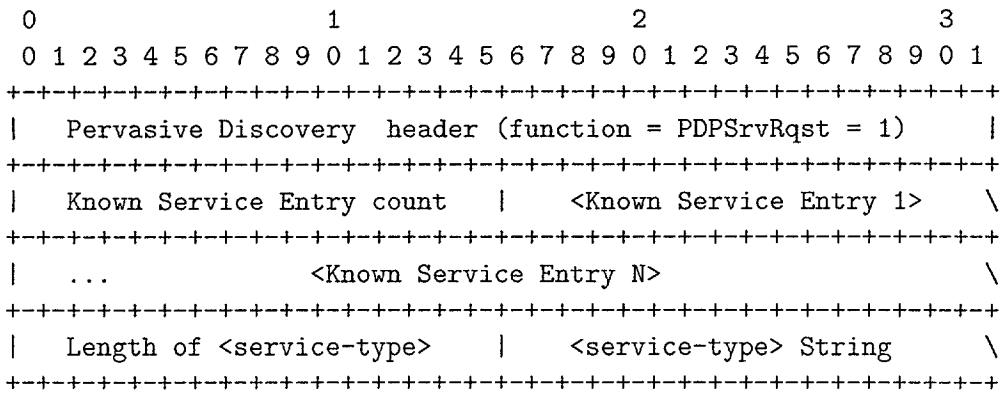


Figura B.13: Formato de PDPSrvRqst.

- Known Service Entry 1..N: descripción de servicios conocidos, según el formato descrito en la Figura B.1. Tamaño del campo variable.
- Length of <service-type>: longitud en bytes del campo <service-type>. Tamaño de 2 bytes.
- <service-type>: es un string que indica el tipo de servicios que se está solicitando. Tamaño del campo variable.

B.8.2. PDP Service Reply

El PDP_SA envía un mensaje PDP Service Reply como respuesta a un mensaje PDP Service Request escuchado en la red.

La respuesta a la búsqueda de un servicio debe contener al menos una Service Entry describiendo un servicio que corresponde con el tipo de servicio solicitado. En el caso de que no exista un servicio que verifique las restricciones de búsqueda no se generará ninguna respuesta.

En un PDPSrvRqst se proporcionarán todos los servicios que coincidan con el tipo de servicio solicitado. En el caso en el que el tipo de servicio sea service:all, se responderá incluyendo todos los servicios conocidos, tanto locales como remotos. Cuando el tamaño del mensaje generado sobrepasa el tamaño máximo de un paquete UDP, se activa el flag OVERFLOW de la cabecera y sólo se incluyen en el mensaje descripciones completas de servicios.

Cuando un PDP_UA procesa este tipo de mensajes debe almacenar las descripciones de los servicios incluidos en el mensaje en la caché de servicios, el

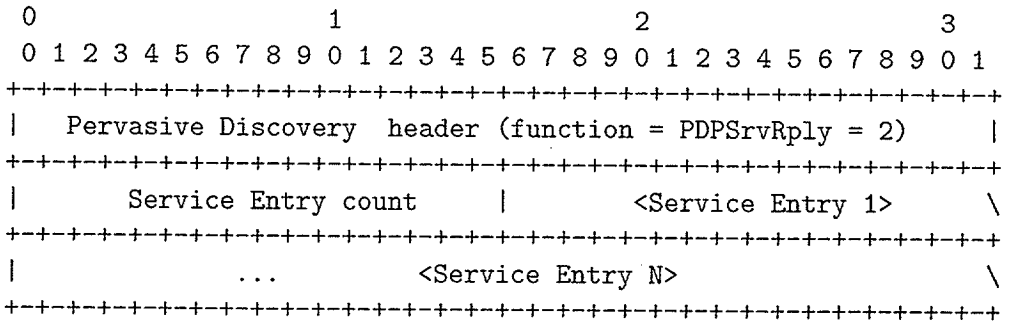


Figura B.14: Formato de PDPSrvRply.

Tiempo de expiración asociado a la entrada en la caché se calculará sumando al tiempo actual el mínimo entre el Lifetime en el campo Service Entry y el Tiempo de disponibilidad del dispositivo local, ver sección B.3.

A continuación describimos los campos contenidos en el mensaje, Figura B.14:

- Service Entry count: número de Service Entries contenidas en el mensaje. Tamaño del campo 2 bytes.

- Service Entry 1...N: descripción de un servicio, según el formato descrito en la Figura B.1. Tamaño del campo variable.

B.9. Características opcionales

Hemos incluido una característica opcional que pueden implementar los PDP_SA, aunque no es obligatorio y por lo tanto, los PDP_UA no tiene porque saber interpretarla, en cuyo caso deben descartar este tipo de mensajes.

Consiste en la posibilidad de enviar un mensaje PDP Service Deregister en el que se incluyen descripciones de servicios que se ofrecen de forma local, para que los PDP_UA que los tengan almacenados en su caché los borren. Este tipo de mensaje es enviado por un dispositivo que detecta que va a abandonar la red en la que se encuentra, o se va a apagar. Este mensaje es opcional porque no en todos los dispositivos y en todas las tecnologías de red es viable detectar este tipo de cambios. Este mensaje se transmite también por multicast/broadcast.

B.10. Mensajes PDP opcionales

B.10.1. PDP Service Deregister

Cuando un dispositivo detecta que va a abandonar la red en la que se encuentra, o va a dejar de estar disponible, su PDP_SA puede enviar un PDPDeReg indicando los servicios locales que posee, y que dejarán de estar disponibles. Al recibir este mensaje los PDP_UA deben eliminar de su caché los servicios almacenados que se correspondan con los servicios indicados en las Service Entries, independientemente del tiempo de expiración que posean.

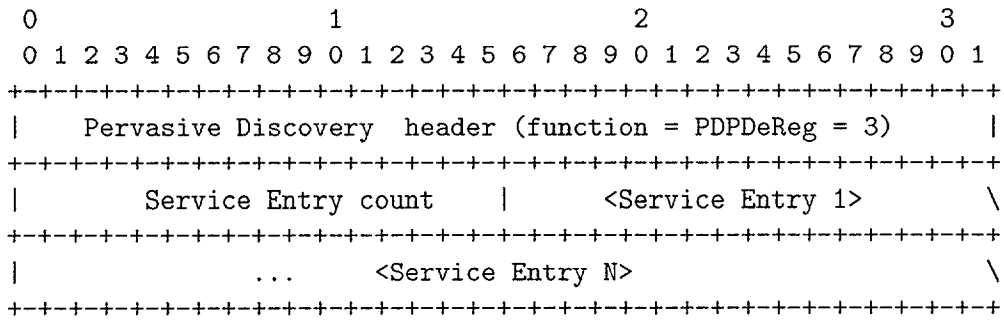


Figura B.15: Formato de PDPDeReg.

Cuando el tamaño del mensaje generado sobrepasa el tamaño máximo de un paquete UDP, se activa el flag OVERFLOW de la cabecera y sólo se incluyen en él descripciones completas de servicios.

A continuación describimos los campos contenidos en el mensaje, Figura B.15:

- Service Entry count: número de Service Entries contenidas en el mensaje. Tamaño del campo 2 bytes.
- Service Entry 1...N: descripción de un servicio, según el formato descrito en la Figura B.1. Tamaño del campo variable.

B.11. Temporizadores del protocolo

PDP emplea varios temporizadores tanto en los PDP_UAs como en los PDP_SAs.

B.11.1. Temporizadores en PDP_UA

Un PDP_UA emplea un temporizador para limitar el tiempo que máximo durante el que espera mensajes de respuesta después de la transmisión de un `PDPSrvRqst`. Este temporizador tiene un valor que hemos denominado:

- `CONFIG_WAIT_RPLY`: cuyo valor por defecto es de 15 segundos.

B.11.2. Temporizadores en PDP_SA

Un PDP_SA antes de enviar una `PDPSrvRply` espera durante un tiempo generado de forma aleatoria. Este tiempo debe ser inversamente proporcional al tiempo de disponibilidad del dispositivo y al número de servicios que vaya a incluir en la respuesta. De este forma, se priorizan a aquellos dispositivos que tienen un visión más acertada del entorno que les rodea.

El método propuesto por defecto para obtener este valor, es generar un número aleatorio según una distribución uniforme entre

$$(0, \text{CONFIG\_WAIT\_TRANSMIT} * \frac{\text{DEFAULT\_AVAILABILITY\_TIME}}{\text{DEFAULT\_AVAILABILITY\_TIME} + T_D * \text{Service\_Entries\_Number}})$$

Siendo,

- `CONFIG_WAIT_TRANSMIT`: intervalo máximo de tiempo que puede esperar un dispositivo antes de enviar su respuesta a un `PDPSrvRqst`. Este valor debe ser menor que `CONFIG_WAIT_RPLY`. Su valor por defecto es de 3 segundos.
- `DEFAULT_AVAILABILITY_TIME`: el tiempo mínimo a partir del que un dispositivo se considera fijo. Su valor por defecto son 3600 segundos.
- T_D: el tiempo de disponibilidad asociado al dispositivo.
- `Service_Entries_Number`: el número de `Service Entries` que se van a incluir en la respuesta.

Esta forma de generar el retardo de las respuestas, es la que se ha empleado en las simulaciones realizadas a lo largo de los capítulos 4 y 5. Se podrían emplear otros métodos y estudiar si se mejoran así las prestaciones del protocolo, pero recordando siempre que una modificación sobre este método nos llevaría a una nueva versión del protocolo.

Apéndice C

Distributed Directory Facilitator: A proposal for the FIPA Ad-hoc First CFT

C.1. Contact point

Name:	Celeste Campo
Postal address:	Universidad Carlos III de Madrid
	Avda. Universidad 30
	28911 Leganés (Madrid), Spain
Affiliation:	Depto. de Ingeniería Telemática
	Universidad Carlos III de Madrid (Spain)
Email address:	celeste@it.uc3m.es
Telephone number:	+34-91-624-5949
Fax number:	+34-91-624-8749
Date:	16-05-2002

C.2. Completeness of the submission

This submission to the FIPA TC Ad-hoc First Call For Technology addresses in part the second requirement stated in section 2 of the CFT: "definition of mechanisms and protocols for agent platform fragments to build, release, join and leave compounds". Specifically, our proposal centers on the fragmentation of the Directory Facilitator (DF).

As suggested in section 6 of the CFT, a device without DF functionality must use a remote DF to advertise its own services and to find out other services provided by other agents on other devices in its surroundings.

The question is: how does the device without DF functionality know the address of the remote DF? Two solutions are possible:

- The address of the remote DF is statically (manually) configured in the device.
- The address is dynamically "discovered" by the device, for example with broadcast messages announcing or requesting a DF.

However, we think pervasive environments cannot be relied upon to have any single device permanently present in order to act as remote DF, because they are dynamic in nature. The second solution described above is better because the device dynamically discovers the nearest remote DF, but it has many problems. First, if the environment changes frequently, the device will be continually discovering a new remote DF. Second, maybe none of the devices present at any moment may be suitable to act as the remote DF.

Here we propose all the devices to have a DF fragment that works in a distributed fashion and dynamically discovers services in its environment. This is what we call a *Distributed DF fragment* and its cost is equivalent at the one of the second solution described above when the environment is very changing.

We have considered existing approaches for dynamic service discovery, such as SLP, Jini, Salutation and UPnP's SSDP. None of them adapts well to the characteristics of ad-hoc networks. So, we have defined a new service discovery protocol, called Pervasive Discovery Protocol (PDP), to be used in the Distributed DF.

In part II of the proposal, we present the concept of Distributed DF, review existing service discovery protocols and describe the PDP algorithm.

C.3. Overview

Recent advances in microelectronic and wireless tecnologies have fostered the proliferation of small devices with limited communication and processing power. They are what are known as "pervasive systems".

Personal Digital Assistants (PDAs) and mobile phones are the more "visible" of these kinds of devices, but there are many others that surround us, unobserved.

For example, today most household appliances have embedded microprocessors. Each one of these small devices offers a specific service to the user, but thanks to their capacity for communication, in the near future they will be able to collaborate with each other to build up more complex services. In order to achieve this, devices in such "ad-hoc" networks should dynamically discover and share services between them when they are close enough. For example, sensors and air conditioning systems in intelligent buildings will talk with our PDAs to automatically adapt the environment to our needs or preferences. As Arthur C. Clarke has said – any sufficiently advanced technology is indistinguishable from magic. We are approaching the time when "open sesame" is a valid user command.

We think that agent technology will be of great help in pervasive systems development. Pervasive systems are inherently dynamic, with devices continually coming in and out. Agents are autonomous software entities that can interact with their environment, and therefore they adapt well to such continuous changes.

However, the use of Multi-Agent Systems (MAS) in pervasive environments poses important challenges, and it is necessary to adapt their design to meet these challenges. We are currently working on this.

In part II of this report, we will define the Distributed Directory Facilitator, that helps agents in search of services, offered by other agents or other systems in the ad-hoc network. Then, we will see that none of the service discovery protocols propose so far in the literature, adapts well to the case of pervasive systems. Finally, we will propose a new service discovery protocol, called PDP.

C.4. Distributed Directory Facilitator

To implement the Directory Facilitator in a pervasive environment, we propose to use a Distributed DF formed by the joint work of Distributed DF fragments in devices that coincide in a given moment in the same ad-hoc network, with no need of any remote "full" DF. The fragments that form the Distributed DF in an ad-hoc network change as devices join and leave the network.

The Distributed DF fragment helps agents working on the device in search of services offered by other devices in the ad-hoc network. It is a kind of "yellow pages" service adapted to the peculiarities of pervasive computing enviroments. One can think of it as a kind of shop assistant. New entrants to the shop will want assistance in order to better locate items of interest, and the agent, because of its relative expertise in the environment, will generally be able to provide the appropriate directions.

To carry out its work, a Distributed DF fragment needs to implement a service discovery protocol. As we will see in the next section, service discovery protocols proposed for use in the Internet do not work well in pervasive systems in ad-hoc networks. We have defined a new protocol, the Pervasive Discovery Protocol, PDP (see Section 4), specially designed to work in this environment. The internal operation of the Distributed DF fragment will be based on this protocol.

It is important to point out here that, since the Distributed DF fragment uses the PDP, it will be able to interoperate with any other system that implements PDP, be it an agent platform or not (e.g., sensors, air-conditioning system, lighting control, etc.).

C.5. Service Discovery

Dynamic service discovering is not a new problem. There are several solutions proposed for fixed networks, with different levels of acceptance. We well now briefley review some of them: SLP, Jini, Salutation and UPnP's SSDP.

The Service Location Protocol (SLP) [RFC 2165, 1997] is an Internet Engineering Task Force standard for enabling IP networks-based applications to automatically discover the location of a required service. The SLP defines three "agents": User Agents (UA), that perform service discovery on behalf of client software, Service Agents (SA), that advertise the location and attributes on behalf of services, and Directory Agents (DA), that store information about the services announced in the network. SLP has two different modes of operation: when a DA is present, it collects all service information advertised by SAs, and UAs unicast their requests to the DA, and when there is not a DA, UAs repeatedly multicast the request they would have unicast to a DA. SAs listen for these multicast requests and unicast responses to the UA.

Jini [Jini, 1999] is a technology developed by Sun Microsystems. Its goal is to enable truly distributed computing by representing hardware and software as Java objects that can form themselves into communities, allowing objects to access services on a network in a flexible way. The service discovery in Jini is based on a directory service, similar to the Directory Agent in SLP, named the Jini Lookup Service (JLS). JLS is necessary to the functioning of Jini, and clients should always discover services using it, and never can do so directly.

Salutation [Salutation Consortium, 1998] is an architecture for looking up, discovering, and accesing services and information. Its goal is to solve the problems of service discovery and utilization among a broad set of applications and equipment in an environment of widespread connectivity and mobility. The Salu-

tation architecture defines an entity called the Salutation Manager (SLM) that functions as a directory of applications, services and devices, generically called Networked Entities. The SLM allows networked entities to discover and to use the capabilities of the other networked entities.

Simple Service Discovery Protocol (SSDP) [Goland et al., 1999] was created as a ligthweight discovery protocol for Universal Plug-and-Play (UPnP) initiative, and defines a minimal protocol for multicast-based discovery. SSDP can work with or without its central directory sevice, called the Service Directory. When a service wants to join the network, first it sends an announcement message to notify its presence to the rest of the devices. This announcement may be sent by multicast, so all other devices will see it, and the Service Directory, if present, will record the announcement. Alternatively, the announcement may be sent by unicast directly to the Service Directory. When a client wants to discover a service, it may ask the Service Directory for it or it may send a multicast message asking for it.

The solutions described above can not be directly applied to the scenario we treat in this report, because they were designed for and are more suitable for (fixed) wired networks. We see two main problems in the solutions enumerated:

- First, many of them use a central server, that maintains the directory of services in the network. Pervasive environments cannot be relied upon to have any single device permanently present in order to act as central server, and furthermore, maybe none of the devices present at any moment may be suitable to act as the server.
- Second, the solutions that may work without a central server, are designed without considering the power constraints typical in wireless networks. They make an extensive use of multicast or broadcast transmissions almost costless in wired networks but are power hungry in wireless networks.

Accepting that alternatives to the centralized approach are required, we consider two alternative approaches to distributing service announcements:

- The "Push" solution, in which a device that offers a service sends unsolicited advertisements, and the other devices listen to these advertisements selecting those services they are interested on.
- The "Pull" solution, in which a device requests a service when it needs it, and devices that offer that service answers the request, perhaps with third devices taking note of the reply for future use.

In pervasive computing, it is very important to minimize the total number of transmissions, in order to reduce battery consume. It is also important to implement mechanisms to detect as soon as possible both the availability and unavailability of services produced when a device joins or leaves the network. These factors must be taken into account when selecting between a push solution or a pull solution.

The DEAPspace group of the IBM Research Zurich Lab. has proposed a solution to the problem of service discovering in pervasive systems without using a central server. The DEAPspace Algorithm [Nidd, 2001] is a pure push solution, in which all devices hold a list of all known services, the so called "world view". Each device periodically broadcasts its "world view" to its neighbours, which update their "world view" accordingly.

We think the DEAPspace algorithm has the following problem: the "world view" of a device spreads from neighbour to neighbour, perhaps arriving to a device where some of those services are in fact not available.

In this report we propose a new service discovery algorithm, the Pervasive Discovery Protocol (PDP), wich merges characteristics of both pull and push solutions. This way we think a better performance can be achived.

C.6. Pervasive Discovery Protocol

The Pervasive Discovery Protocol (PDP) is intended to solve the problem of enumerating the services available in a local cell in a low power short-range wireless network, composed of devices with limited transmision power, memory, processing power, etc. The classical service discovery protocols use a centralized server that listens for broadcast or multicast announcements of available services at a known port address, and lists the relevant services in response to enquiries. The protocol we propose does away with the need for the central server. The kind of enviroments described above cannot be relied upon to have any single device permanently present in order to act us central server, and further, none of the devices present at any moment may be suitable to act as the server.

One of the key objectives of the PDP is to minimize battery use in all devices. This means that the number of transmisions necessary to discover services should be reduced as much as possible. A device announces its services only when other devices request the service. Service announcements are broadcast to all the devices in the network, all of which will get to know about the new service simultaneously at that moment, without having to actively query for it.

In the remainder of this section, we present the application scenario for PDP and some considerations to be taken into account, and then we will formally describe the algorithm used to implement it.

C.6.1. Application scenario

We will suppose that there are D devices, each one with I network interfaces. Each device offers S services and expects to remain available for T seconds. This time T is previously configured in the device, depending on its mobility characteristics. The Distributed DF fragment has a cache associated with each interface which contains a list of the services that have been heard from on this interface. Each element e of this list has two fields: the service description, $e.description$, and a time that it is calculated that the service will be available for, $e.timeout$. The calculation is exactly the minimum of two values: the time that the local device has promised to remain available, T, and the time that the server that offers the service announced that it would be available for.

Services are not associated with any specific interface and the availability time T of the device is always included in the announcements of its services.

For simplicity, we suppose that local services are stored in the cache associated with the loopback interface ($cache_0$).

C.6.2. Algorithm description

The PDP has two messages: **PDP request**, which is used to send service annoucements and **PDP reply**, which is used to answer a PDP request, announcing availables services.

Now, we will explain in detail how a Distributed DF fragment uses these primitives.

PDP request

When an application or the final user of the device needs a service, whether a specific service or any service offered by the enviroment, it requests the service from the Distributed DF fragment. The number of broadcast transmissions should be minimized, so:

- If a specific service S has been requested, the Distributed DF fragment

searches for that service in all its caches. If it is not found, it broadcasts a PDP request for that service (Table C.1).

```
for ( i = 0 to I ) {
  if (S ∈ cache_i) return i;
}
for (i = 1 to I ) {
  remote_service = PDP request(S);
  if (∃ remote_service) {
    add_service(remote_service, cache_i);
    return i;
   }
}
```

Table C.1: PDP request S

- If the request is for all available services in the network, the Distributed DF fragment updates its caches by sending a PDP request message through all the interfaces of the device (Table C.2).

```
for (i = 1 to I ) {
  remote_services_list= PDP request(ALL);
}
```

Table C.2: PDP request ALL

PDP reply

The Distributed DF fragments in all devices are continually listening on each interface for all type of messages (PDP requests and PDP replies).

When a PDP reply is received, announcing a service, the Distributed DF fragments update their caches accordingly.

When a PDP request for a specific service S (Table C.3) is received then the Distributed DF fragment:

- Checks whether the requested service, S, is one of its local services and therefore is stored in the loopback cache, or is not.

- If not, it generates a random time t, inversely proportional to the availability time of the device, T. So, the more time the device is able to offer the service, the higher the probability of the device answering first.
- During the interval t, the Distributed DF fragment listens to the network. If another reply to this PDP request arrives, it aborts its PDP reply, otherwise when the interval expires, it sends its PDP reply.

```
if ( ∃ e ∈ cache_0 / S = e.description) {
  t = generate_random_time(1/T);
  wait(t);
  if ( not listened PDP reply)
    PDP reply(e);
}
```

Table C.3: PDP reply S

When a PDP request for all services ALL (Table C.4) is received then the Distributed DF fragment:

- Generates a random time t, inversely proportional to the availability time of the device, T, and to the number of elements stored in the cache of that interface. So, the more time the device is able to offer the service and the biggest the cache, the higher the probability of answering first. We suppose the device with the highest availability time and the bigger cache is the one with the most accurate view of the world.
- During the interval t, the Distributed DF fragment listens to the network. If another reply to this PDP request arrives, it aborts its PDP reply, otherwise when the interval expires, it sends its PDP reply listing the services in the loopback cache plus the services in the cache of that interface.

```
t = generate_random_time( 1/(T*cache_size) );
wait(t);
if ( not listened PDP reply)
  PDP reply(cache_0, cache_i);
```

Table C.4: PDP reply ALL

C.7. Additional Authors

Andrés Marín (amarin@it.uc3m.es), Carlos García-Rubio (cgr@it.uc3m.es), and Peter T. Breuer (ptb@it.uc3m.es).

Apéndice D

Directory Facilitator and Service Discovery Agent

D.1. Contact point

Name:	Celeste Campo
Postal address:	Universidad Carlos III de Madrid
	Avda. Universidad 30
	28911 Leganés (Madrid), Spain
Affiliation:	Depto. de Ingeniería Telemática
	Universidad Carlos III de Madrid (Spain)
Email address:	celeste@it.uc3m.es
Telephone number:	+34-91-624-5949
Fax number:	+34-91-624-8749
Date:	02-08-2002

D.2. Scope of the document

This document is the contribution of the University Carlos III of Madrid to the white-paper "Agents in Ad Hoc Environments" to be discussed at FIPA26 in Helsinki.

In this document we broach the problem of implementing yellow pages services in agent platforms for ad-hoc networks. We present a solution integrated with FIPA specifications, centred in the adaption of the Directory Facilitator to work in ad-hoc environments.

First, we will start with an introduction to the characteristics and requirements of pervasive systems and ad-hoc networks. Secondly, and following the proposed index for the mentioned white-paper, we briefly review some existing discovery technologies and their limitations in ad-hoc networks; we also propose a new service discovery protocol, the Pervasive Discovery Protocol (PDP). Thirdly, we propose a solution for service discovery based on existing service discovery mechanisms. Finally, we propose another solution at an agent level that uses multicast ACL messages improved with the use of caches.

D.3. Introduction

Recent advances in microelectronic and wireless technologies have fostered the proliferation of small devices with limited communication and processing power. They are what are known as "pervasive systems".

Personal Digital Assistants (PDAs) and mobile phones are the more "visible" of these kinds of devices, but there are many others that surround us, unobserved. For example, today most household appliances have embedded microprocessors. Each one of these small devices offers a specific service to the user, but thanks to their capacity for communication, in the near future they will be able to collaborate with each other to build up more complex services. In order to achieve this, devices in such "ad-hoc" networks should dynamically discover and share services between them when they are close enough. For example, sensors and air conditioning systems in intelligent buildings will talk with our PDAs to automatically adapt the environment to our needs or preferences.

The most accepted definition of ad-hoc network is the following: it is a network that consists of mobile nodes that use wireless interfaces to send and receive data, with no central, permanent network infrastructure.

The dynamic topology of ad-hoc networks, the fluctuating quality and capacity of wireless links, and the limited capacity (memory, processing power, and battery life) of many of the devices connected to this kind of networks, pose important, new challenges that should be addressed when defining new proposals for this environment.

Next, we will enumerate some of the requirements of ad-hoc networks, different of those of legacy fixed or mobile networks:

- Distributed operation: In ad-hoc environments, nodes join and leave the network spontaneously, so network operation must be distributed between all the devices that form the network at any given time. This is one of

the requirements that more differ from traditional networks. In traditional networks, some elements perform special tasks without which the global operation of the network is not possible. For example, in GSM, network operation rely on the existence of base stations. Terminal equipments connect to these base stations to obtain connectivity and to access services.

- High cost of the communications: Communications in ad-hoc networks are expensive because they imply high battery consume in devices typically with limited battery power.
- Intransitive connectivity: In fixed networks, if A has connectivity with B, and B has connectivity with C, then A has connectivity with C. In ad-hoc networks this may not be true due to the short-range coverage of the underlying wireless protocols.
- Very changing environments: Nodes in ad-hoc networks continually change (join and leave the network).
- Broadcast domains: In ad-hoc networks, you can use broadcasts to reach all nodes in your range, that is, all the nodes you are able to communicate with. In fixed networks, broadcast domains are constrained to closer nodes (those within your local area network), and so broadcasts do not reach all nodes you are able to communicate with.

We think that agent technology will be of great help in pervasive systems development. Pervasive systems are inherently dynamic, with devices continually coming in and out. Agents are autonomous software entities that can interact with their environment, and therefore they adapt well to such continuous changes.

However, the use of Multi-Agent Systems (MAS) in pervasive environments poses important challenges, and it is necessary to adapt their design to meet these challenges. One of them is the implementation of yellow pages services in agent platforms for ad-hoc networks. Here we will present a solution integrated with FIPA specifications, centred in the adaption of the Directory Facilitator to work in ad-hoc environments.

D.4. Discovery technologies

D.4.1. Review of existing discovery technologies

Dynamic service discovering is not a new problem. There are several solutions proposed for fixed networks, with different levels of acceptance. We well now

briefly review some of them: SLP, Jini, Salutation and UPnP's SSDP.

Service Location Protocol (SLP)

The Service Location Protocol (SLP) [RFC 2165, 1997] is an Internet Engineering Task Force standard for enabling IP networks-based applications to automatically discover the location of a required service. The SLP defines three "agents": User Agents (UA), that perform service discovery on behalf of client software, Service Agents (SA), that advertise the location and attributes on behalf of services, and Directory Agents (DA), that store information about the services announced in the network. SLP has two different modes of operation: when a DA is present, it collects all service information advertised by SAs, and UAs unicast their requests to the DA, and when there is not a DA, UAs repeatedly multicast the request they would have unicast to a DA. SAs listen for these multicast requests and unicast responses to the UA.

Jini

Jini [Jini, 1999] is a technology developed by Sun Microsystems. Its goal is to enable truly distributed computing by representing hardware and software as Java objects that can form themselves into communities, allowing objects to access services on a network in a flexible way. The service discovery in Jini is based on a directory service, similar to the Directory Agent in SLP, named the Jini Lookup Service (JLS). JLS is necessary to the functioning of Jini, and clients should always discover services using it, and never can do so directly.

Salutation

Salutation [Salutation Consortium, 1998] is an architecture for looking up, discovering, and accessing services and information. Its goal is to solve the problems of service discovery and utilisation among a broad set of applications and equipment in an environment of widespread connectivity and mobility. The Salutation architecture defines an entity called the Salutation Manager (SLM) that functions as a directory of applications, services and devices, generically called Networked Entities. The SLM allows networked entities to discover and to use the capabilities of the other networked entities.

Simple Service Discovery Protocol (SSDP)

Simple Service Discovery Protocol (SSDP) [Goland et al., 1999] was created as a lightweight discovery protocol for Universal Plug-and-Play (UPnP) initiative, and defines a minimal protocol for multicast-based discovery. SSDP can work with or without its central directory service, called the Service Directory. When a service wants to join the network, first it sends an announcement message to notify its presence to the rest of the devices. This announcement may be sent by multicast, so all other devices will see it, and the Service Directory, if present, will record the announcement. Alternatively, the announcement may be sent by unicast directly to the Service Directory. When a client wants to discover a service, it may ask the Service Directory for it or it may send a multicast message asking for it.

D.4.2. Problems of existing discovery technologies in ad-hoc networks

The solutions described above can not be directly applied to the scenario we treat in this report, because they were designed for and are more suitable for (fixed) wired networks. We see two main problems in the solutions enumerated:

- First, many of them use a central server, that maintains the directory of services in the network. Pervasive environments cannot be relied upon to have any single device permanently present in order to act as central server, and furthermore, maybe none of the devices present at any moment may be suitable to act as the server.

- Second, the solutions that may work without a central server, are designed without considering the power constraints typical in wireless networks. They make an extensive use of multicast or broadcast transmissions almost costless in wired networks but are power hungry in wireless networks.

Accepting that alternatives to the centralised approach are required, we consider two alternative approaches to distributing service announcements:

- The "Push" solution, in which a device that offers a service sends unsolicited advertisements, and the other devices listen to these advertisements selecting those services they are interested on.

- The "Pull" solution, in which a device requests a service when it needs it, and devices that offer that service answers the request, perhaps with third devices taking note of the reply for future use.

In pervasive computing, it is very important to minimise the total number of transmissions, in order to reduce battery consume. It is also important to implement mechanisms to detect as soon as possible both the availability and unavailability of services produced when a device joins or leaves the network. These factors must be taken into account when selecting between a push solution or a pull solution.

The DEAPspace group of the IBM Research Zurich Lab. has proposed a solution to the problem of service discovering in pervasive systems without using a central server. The DEAPspace Algorithm [Nidd, 2001] is a pure push solution, in which all devices hold a list of all known services, the so called "world view". Each device periodically broadcasts its "world view" to its neighbours, which update their "world view" accordingly.

We think the DEAPspace algorithm has the following problem: the "world view" of a device spreads from neighbour to neighbour, perhaps arriving to a device where some of those services are in fact not available.

In this report we propose a new service discovery algorithm, the Pervasive Discovery Protocol (PDP), which merges characteristics of both pull and push solutions. This way we think a better performance can be achieved.

D.4.3. Pervasive Discovery Protocol

The Pervasive Discovery Protocol (PDP) is intended to solve the problem of enumerating the services available in a local cell in a low power short-range wireless network, composed of devices with limited transmission power, memory, processing power, etc. The classical service discovery protocols use a centralised server that listens for broadcast or multicast announcements of available services at a known port address, and lists the relevant services in response to enquiries. The protocol we propose does away with the need for the central server. The kind of environments described above cannot be relied upon to have any single device permanently present in order to act us central server, and further, none of the devices present at any moment may be suitable to act as the server.

One of the key objectives of the PDP is to minimise battery use in all devices. This means that the number of transmissions necessary to discover services should be reduced as much as possible. A device announces its services only when

other devices request the service. Service announcements are broadcast to all the devices in the network, all of which will get to know about the new service simultaneously at that moment, without having to actively query for it.

In the remainder of this section, we present the application scenario for PDP and some considerations to be taken into account, and then we will formally describe the algorithm used to implement it.

Application scenario

We will suppose that there are D devices, each one with I network interfaces. Each device offers S services and expects to remain available for T seconds. This time T is previously configured in the device, depending on its mobility characteristics. The Distributed DF fragment has a cache associated with each interface which contains a list of the services that have been heard from on this interface. Each element e of this list has two fields: the service description, $e.description$, and a time that it is calculated that the service will be available for, $e.timeout$. The calculation is exactly the minimum of two values: the time that the local device has promised to remain available, T, and the time that the server that offers the service announced that it would be available for.

Services are not associated with any specific interface and the availability time T of the device is always included in the announcements of its services.

For simplicity, we suppose that local services are stored in the cache associated with the loopback interface ($cache_0$).

Algorithm description

The PDP has two messages: **PDP request**, which is used to send service announcements and **PDP reply**, which is used to answer a PDP request, announcing available services.

Now, we will explain in detail how a Distributed DF fragment uses these primitives.

PDP request When an application or the final user of the device needs a service, whether a specific service or any service offered by the environment, it requests the service from the Distributed DF fragment. The number of broadcast transmissions should be minimised, so:

- If a specific service S has been requested, the Distributed DF fragment

searches for that service in all its caches. If it is not found, it broadcasts a PDP request for that service (Table D.1).

```
for ( i = 0 to I ) {
  if (S ∈ cache_i) return i;
}
for (i = 1 to I ) {
  remote_service = PDP request(S);
  if (∃ remote_service) {
    add_service(remote_service, cache_i);
    return i;
   }
}
```

Table D.1: PDP request S

- If the request is for all available services in the network, the Distributed DF fragment updates its caches by sending a PDP request message through all the interfaces of the device (Table D.2).

```
for (i = 1 to I ) {
  remote_services_list= PDP request(ALL);
}
```

Table D.2: PDP request ALL

PDP reply The Distributed DF fragments in all devices are continually listening on each interface for all type of messages (PDP requests and PDP replies).

When a PDP reply is received, announcing a service, the Distributed DF fragments update their caches accordingly.

When a PDP request for a specific service S (Table D.3) is received then the Distributed DF fragment:

- Checks whether the requested service, S, is one of its local services and therefore is stored in the loopback cache, or is not.

- If not, it generates a random time t, inversely proportional to the availability time of the device, T. So, the more time the device is able to offer the service, the higher the probability of the device answering first.
- During the interval t, the Distributed DF fragment listens to the network. If another reply to this PDP request arrives, it aborts its PDP reply, otherwise when the interval expires, it sends its PDP reply.

```
if ( ∃ e ∈ cache_0 / S = e.description) {
  t = generate_random_time(1/T);
  wait(t);
  if ( not listened PDP reply)
    PDP reply(e);
}
```

Table D.3: PDP reply S

When a PDP request for all services ALL (Table D.4) is received then the Distributed DF fragment:

- Generates a random time t, inversely proportional to the availability time of the device, T, and to the number of elements stored in the cache of that interface. So, the more time the device is able to offer the service and the biggest the cache, the higher the probability of answering first. We suppose the device with the highest availability time and the bigger cache is the one with the most accurate view of the world.
- During the interval t, the Distributed DF fragment listens to the network. If another reply to this PDP request arrives, it aborts its PDP reply, otherwise when the interval expires, it sends its PDP reply listing the services in the loopback cache plus the services in the cache of that interface.

D.5. Solution for service discovery based on the underlying service discovery mechanisms

Here we present a solution integrated with FIPA specifications, centred in the adaption of the Directory Facilitator to work with any service discovery protocol

```
t = generate_random_time( 1/(T*cache_size) );
wait(t);
if ( not listened PDP reply)
  PDP reply(cache_0, cache_i);
```

Table D.4: PDP reply *ALL*

used by the underlying ad-hoc network technology (BT SDP, Jini, UPnP SSDP, PDP or whatever). This solution is based in the introduction of a new agent, the Service Discovery Agent.

D.5.1. Directory Facilitator in Ad-hoc networks

We have assumed the following prerequisites to adapt the DF to ad-hoc environments:

- A DF in an ad-hoc network must provide the same functionality than a DF in a traditional network, i.e., it must be a yellow pages service in which agents may register their services (to offer them to other agents), and search for services offered by other agents.
- A DF in an ad-hoc network must keep the same functions specified for the DF in FIPA00023. So, agents do not need to modify the way they interact with the DF. Therefore, it will have the following functions: `register`, `deregister`, `modify` and `search`.
- A DF must provide flexible service search mechanisms that include both local services (in the same platform) and remote services. In ad-hoc networks, remote services will be very changing because devices that offer them join and leave the network continually.

We centre now in the last point. We will argue the need to add a new functional block to the FIPA architecture to adapt it to ad-hoc networks.

In the traditional DF definition, the search of remote services is accomplished by using the concept of DF federation. DFs, besides registering services offered by local agents (native or not), they may also register other DFs (the so called federated DFs). This allows them to extend the search of services to the ones

registered in these other DFs. The number of hops between federated DFs is limited by means of a parameter that restricts the depth of a search.

We will see now the disadvantages of DF federation in remote service discovery in ad-hoc networks:

- The search of an specific remote service always implies one communication with that remote system, because we just know that there is a DF in that platform, but not the services registered in it. This means at least one transmission is needed, with no success guarantee.
- If search conditions allow a multihop search, i.e., a search in a remote DF may spread to other remote DFs federated in it, then it is possible that services found in those remote DFs will not be reachable from the first system, and therefore they are not valid search results.

D.5.2. Service Discovery Agent

To overcome the problems stated above, we propose the introduction of a new agent, the **Service Discovery Agent (SDA)**, in the FIPA architecture for ad-hoc networks. This new agent will be mandatory, and it will have the following functionality:

- To register and to update, in the platform's DF, the list of remote services offered in the ad-hoc network. Therefore, the DF will have entries corresponding to remote services, not to remote (federated) DFs, and so the number of transmissions will be minimised [1].
- To propagate the search of remote services when solicited (because search conditions allow so).
- To announce services registered in the DF using the associated SDP, when allowed so.

The SDA will use a Service Discovery Protocol (SDP) to discover remote services. In this sense, in order to guarantee an efficient solution for ad-hoc networks, a proper SDP should be selected that fits well into the restrictions exposed in the former section. In the second part of the document, we analyse some SDPs and propose a new SDP for ad-hoc networks.

[1] Transmissions will be minimised as long as the SDP (Service Discovery Protocol) the agent uses is adapted to work in ad-hoc environments and makes a minimum number of transmissions to discover remote services

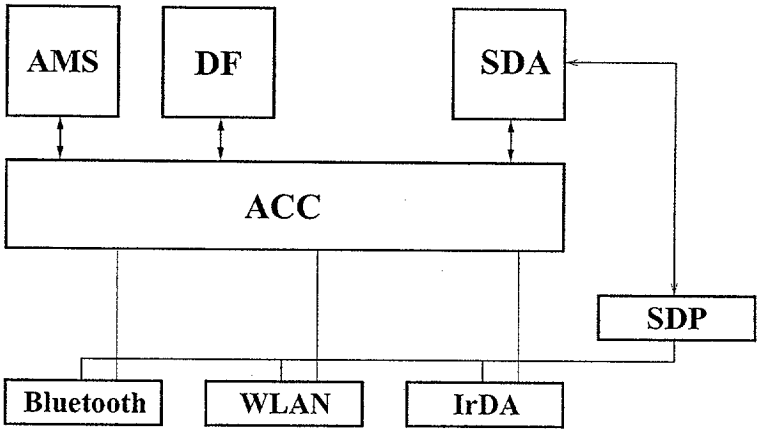


Figure D.1: Proposed Architecture for the FIPA Ad-hoc

D.5.3. Registration of the Service Discovery Agent with the Directory Facilitator

The SDA agent provides a service that is mandatory to be registered in the DF of the platform where it resides. This way, there will be always one entry corresponding to the SDA in DFs in ad-hoc networks, with the following reserved AID:

```
(agent-identifier
  :name sda@hap
  :addresses (sequence hap_transport_address))
```

This agent will register in the DF by setting the `:type` parameter of the `service-description` to the `sda` value.

When the DF processes a `search` and search propagation in the ad-hoc network is allowed (the restriction `max-depth = 1` may be reused for this) the DF will delegate the search to the SDA.

Each time the SDA (through the SDP) knows about a new remote service in the ad-hoc network, it registers the service in the DF by a `register` request. Since the network is changing, this registers will have an associated `timeout`. The timeout will be managed by the SDA, so when the time expires, the SDA will do a `deregister` request to the DF. The SDA just has to store the corresponding `agent-identifier` and its associated `timeout`.

The SDA in its turn may provide the SDP with the local services registered in the DF. To do so, the SDA will make a `search` request in the DF.

D.6. Solution for service discovery at an agent level

Service discovery could also be done at an agent level without using the service discovery protocol possibly provided by the underlying network technology. Here we present our proposal in this sense.

Our solution here is also based in the use of a Service Discovery Agent (SDA), but now it will not work in cooperation with a service discovery protocol, but by sending and receiving special multicast ACL messages.

To minimise the number of transmissions we propose the way the SDA will work to be the same of our PDP protocol. That is, when the SDA wants to search a service in the ad-hoc network, it sends a multicast search message to all the SDAs in the network, requesting that service. The answer will be also multicasted, so the other SDAs will also learn from the answer and maintain a cache. As in PDP, just one SDA, the faster one (see the timer mechanism of PDP), will answer the request.

If FIPA TC Ad-Hoc finally adopts a solution like this, it will be necessary to deal with the definition of a multicast format for ACL messages. For example, to contact all the SDA agents present in an ad-hoc network, a multicast message with something like `sda@*` in the `receiver` field could be used.

D.7. Conclusions

In this paper we have proposed two possible solutions for FIPA platform and service discovery in ad-hoc networks. They both are based in the use of an agent, the Service Discovery Agent. In the first one, any underlying Service Discovery Protocol could be used. We reviewed existing discovery protocols and proposed a new one, the PDP protocol. In the second solution the SDA itself would discover the services through the use of multicast ACL messages in a way similar to the PDP protocol. These multicast messages need to be defined.

One of our premises was not to require modification of existing FIPA specifications, so we have maintained the DF as a mandatory component. If in the near future FIPA allows the DF to be optional, then the SDA of our proposal could assume the functions of the DF, and so no DF would be necessary and the implementation could have an smaller memory footprint.

Glosario de términos

2

2G Segunda Generación.

2.5G Generación intermedia entre 2G y 3G.

3

3G Tercera Generación.

3GPP 3rd Generation Partnership Project.

A

AFH Adaptative Frequency Hopping.

AIR Advanced Infrared.

AMS Agent Management System.

API Application Programming Interface.

AWT Abstract Window Toolkit.

C

CCK Complementary Code Keying.

CDC Connected Device Configuration.

CLDC Connected Limited Device Configuration.

CPU Central Process Unit.

CSMA/CA Carrier-Sense Multiple-Access with Collision-Avoidance.

CSMA/CD Carrier-Sense Multiple-Access with Collision-Detection.

D

DAS Device Access Specification.

DECT Digital Enhanced Cordless Telephone.

DF Directory Facilitator.

DFPQ Distributed Fair Priority Queuing.

DM Discovery Module.

DSSS Direct-Sequence Spread Spectrum.

E

EEPROM Electrically Erasable Programmable Read-Only Memory.

EDGE Enhanced Data rates for Global Evolution.

ETSI European Telecommunications Standards Institute.

ETSI–BRAN ETSI - Broadband Radio Access Networks.

F

FDQAM Frequency Diversity Quadrature Amplitude Modulation.

FHSS Frequency-Hopping Spread Spectrum.

FP Foundation Profile.

G

GERAN GSM/EDGE Radio Access Network.

GPRS General Packet Radio Service.

GSM Global System for Mobile communications.

GSN GPRS Support Node.

GGSN Gateway GPRS Support Node

H

HAP Home Agent Platform.

HomePNA Home Phone Line Networking Alliance.

HTTP Hypertext Transfer Protocol.

HTTPU Hypertext Transfer Protocol over UDP.

HTTPMU Hypertext Transfer Protocol over Multicast.

I

IAS Information Access Service.

IETF Internet Engineering Task Force.

IMT-2000 International Mobile Telecommunications – 2000.

IrCOMM Serial and Parallel Port Emulation over Infrared.

IrDA Infrared Data Association.

IrLAN Infrared Local Area Network.

IrLAP Infrared Link Access Protocol.

IrLMP Infrared Link Management Protocol.

IrMC IrDA Mobile Communication.

IrOBEX Infrared Object Exchange.

IrPHY Infrared physical layer.

IrTRAN-P Infrared Transfer Picture.

ISM Industrial Scientific Medical.

ITU International Telecommunication Union.

J

J2ME Java 2 Platform Microedition.

JCP Java Community Process.

JLS Jini Lookup Service.

L

LAN Local Area Network.

M

MAC Medium Access Control.

MIDP Mobile Information Device Profile.

MTS Message Transport System.

O

OBEX Object Exchange.

OFDM Orthogonal Frequency-Division Multiplexing.

OSGi Open Services Gateway Initiative.

P

P2P Peer-to-Peer.

PBP Personal Basis Profile.

PDA Personal Digital Assistant.

PDAP Personal Digital Assistant Profile.

PDP Pervasive Discovery Protocol.

PP Personal Profile.

PSTN Public Switched Telephone Nerwork.

Q

QAM Quadrature Amplitude Modulation.

R

RAM Read Access Memory.

RFC Request For Comments.

RMI Remote Method Invocation.

S

SDA Service Discovery Agent.

SDP Service Discovery Protocol.

SGSN Serving GPRS Support Node.

SIP Session Initiation Protocol.

SLP Service Location Protocol.

SMS Short Messaging Service.

SRVLOC IETF Service Location Protocol Working Group.

SSDP Simple Service Discovery Protocol.

SWAP Shared Wireless Access Protocol

T

TinyTP Tiny Transport Protocol.

TDMA Time Division Multiple Access.

TDD Time Division Duplex.

U

UMTS Universal Mobile Telecommunications System.

UPnP Universal Plug-and-Play.

URI Universal Resource Identifier.

URL Universal Resource Locator.

UTRAN UMTS Terrestrial Radio Access Network.

V

VHE Virtual Home Enviroment.

W

WPAN Wireless Personal Area Networks.

X

XML eXtensible Markup Language.

Z

ZeroConf IETF Zero Configuration Working Group.

Bibliografía

[3GPP TS 21.101, 2000] 3GPP TS 21.101 (2000). Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); 3rd Generation mobile system Release 1999 Specifications.

[3GPP TS 21.102, 2001] 3GPP TS 21.102 (2001). Universal Mobile Telecommunications System (UMTS); 3rd Generation mobile system Release 4 Specifications.

[3GPP TS 21.103, 2002] 3GPP TS 21.103 (2002). Universal Mobile Telecommunications System (UMTS); 3rd Generation mobile system Release 5 Specifications.

[3GPP TS 21.103, 2003] 3GPP TS 21.103 (2003). Universal Mobile Telecommunications System (UMTS); 3rd Generation mobile system Release 6 Specifications.

[3GPP TS 43.051, 2002] 3GPP TS 43.051 (2002). Digital cellular telecommunications system (Phase 2+); GSM/EDGE Radio Access Network (GERAN) overall description; Stage 2.

[Adjie-Winoto et al., 1999] Adjie-Winoto, W., Schwartz, E., Balakrishnan, and Lilley, J. (1999). The design and implementation of an intentional naming system. In *17th ACM Symposium on Operating Systems Principles (SOSP'99)*, pages 186–201.

[Almenárez y Campo, 2003] Almenárez, F. and Campo, C. (2003). SPDP: A Secure Service Discovery Protocol for Ad-hoc networks. In *EUNICE 2003 9th Open European Summer School and IFIP Workshop on Next Generation Networks*, pages 213–218, Hungary, Budapest - Balatonfüred.

[Andersson, 2001] Andersson, C. (2001). *GPRS and 3G wireless applications.* John Wiley & Sons.

[Ankolenkar et al., 2002] Ankolenkar, A., Burstein, M., Hobbs, J. R., Lassila, O., Martin, D. L., McDermott, D., McIlraith, S. A., Narayanan, S., Paolucci, M., Payne, T. R., and Sycara, K. (2002). DAML-S: Web Service Description for the Semantic Web. In *The First International Semantic Web Conference (ISWC)*, Sardinia (Italy).

[Baldi y Picco, 1998] Baldi, M. and Picco, G. (1998). Evaluating the tradeoffs of mobile code design paradigms in network management applications. In Kemmerer, R., editor, *Proceedings of the 20th International Conference on Software Engineering*, IEEE CS Press, pages 146–155.

[Banavar et al., 2000] Banavar, G., Beck, J., Gluzberg, E., Munson, J., Sussman, J., and Zukowshi, D. (2000). Challenges: An Application Model for Pervasive Computing. In *Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000).*

[Bellifemine et al., 1999] Bellifemine, F., Poggi, A., and Rimassa, G. a. (1999). JADE: A FIPA-compliant agent framework. In *Proceedings of PAAM'99*, pages 97–108, London.

[Berger y Watzke, 2002] Berger, M. and Watzke, M. (2002). AdHoc Proposal - Reviewed Draft for FIPA 25. Response to 1st AdHoc Call for Technology. http://www.fipa.org/docs/input/f-in-00064.

[Berger et al., 2003] Berger, M., Watzke, M., and Helin, H. (2003). Towards a FIPA Approach for Mobile Ad hoc Environments. In *8th International Conference on Intelligence in next generations Netwroks. ICIN 2003*, Bordaux (France).

[Bettstetter et al., 1999] Bettstetter, C., Vogel, H.-J., and Eberpacher, J. (1999). GSM Phase 2+ General Packet Radio Service GPRS: Architecture, Protocols, and Air Interface. *IEEE Communications Surveys*, 2(3).

[Bisdikian, 2001] Bisdikian, C. (2001). An Overview of the Bluetooth Wireless Technology. *IEEE Communications Magazine*, pages 86–94.

[Bluetooth v1.1, 2001] Bluetooth v1.1 (2001). Specification of the Bluetooth System v1.1. http://www.bluetooth.com/dev/specifications.asp.

[Borenstein, 1994] Borenstein, N. S. (1994). EMail With A Mind of Its Own: The Safe-Tcl Language for Enabled Mail. In *IFIP International Conference.*

[Bray y Sturman, 2001] Bray, J. and Sturman, C. F. (2001). *Bluetooth : connect without cables.* Prentice Hall Professional Technical Reference.

[Cable Home 1.0, 2002] Cable Home 1.0 (2002). CableHome 1.0 Specification.

[Callaway et al., 2002] Callaway, E., Gorday, P., Hester, L., Gutierrez, J. A., Naeve, M., Heile, B., and Bahl, V. (2002). Home Networking with IEEE 802.15.4: A Developing Standard for Low-Rate Wireless Personal Area Networks. *IEEE Communications Magazine*, pages 70–77.

[Campo, 2002a] Campo, C. (2002a). Agentes móviles en computación ubicua. In Aedo Cuevas, I., Díaz Pérez, P., and Fernández Llamas, C., editors, *Actas del Tercer Congreso Interacción Persona-Ordenador Interacción 2002*, pages 215–219, Leganés, Spain.

[Campo, 2002b] Campo, C. (2002b). Directory Facilitator and Service Discovery Agent. Technical report, FIPA Ad-hoc Technical Committee.

[Campo, 2002c] Campo, C. (2002c). Service Discovery in Pervasive Multi-Agent Systems. In Finin, T. and Maamar, Z., editors, *AAMAS Workshop on Ubiquitous Agents on embedded, wearable, and mobile agents*, Bologna, Italy.

[Campo et al., 2002a] Campo, C., Carcía, C., Almenares, F., Marín, A., and Delgado, C. (2002a). TAgentsP y PDP: propuestas para una plataforma de agentes en computación ubicua. In *Segundo Congreso Iberoamericano de Telemática CITA 2001*, Mérida, Venezuela.

[Campo et al., 2001a] Campo, C., García, C., Marín, A., and Delgado, C. (2001a). Plataformas de Agentes en Terminales de Telefonía Móvil. In *XI Jornadas de I+D en Telecomunicaciones*, Madrid, Spain.

[Campo et al., 2001b] Campo, C., García, C., Marín, A., and Delgado, C. (2001b). Tecnología de agentes en los sistemas de telefonía móvil. In *III Jornadas de Ingeniería Telmática. JITEL 2001*, Barcelona, Spain. ISBN 84-7653-783-2.

[Campo et al., 2003] Campo, C., García-Rioja, R. M., and Díez-Andino, G. (2003). Mecanismos de serialización J2ME. In *I Congreso JavaHispano*, Leganés. Madrid.

[Campo et al., 2001c] Campo, C., Marín, A., García, A., Díaz, I., Breuer, P., Delgado, C., and García, C. (2001c). JCCM: Flexible Certificates for Smartcards with Java Card. In Attali, I. and Jensen, T., editors, *Proceedings of the International Conference on Research in Smart Cards*, volume 2140 of *Lecture Notes in Computer Science*, pages 34–42, Cannes, France.

[Campo et al., 2002b] Campo, C., Marín, A., García, C., and Breuer, P. (2002b). Distributed Directory Facilitator: A proposal for the FIPA Ad-hoc First CFT. Technical report, FIPA Ad-hoc Technical Committee.

[Casillas, 2003] Casillas, J. M. (2003). Implementación de los protocolos de descubrimiento de servicios PDP y SSDP bajo Network Simulator y desarrollo de un sistema de instrumentación y medida. Master's thesis, Ingeniería en Informática. Escuela Politécnica Superior. Universidad Carlos III de Madrid.

[CDC, 2002] CDC (2002). *Connected Device Configuration (JSR-36).* SUN Microsystems.

[Chakraborty, 2001] Chakraborty, D. (2001). Service Composition in Ad-hoc Environments. Technical Report TR-CS-01-20, Department of Computer Science and Electrical Engineering. University of Maryland, Baltimore Country. Baltimore.

[Chen, 2000] Chen, Z. (2000). *Java Card technolgy for smart cards : architecture and programmer's guide.* Addison-Wesley.

[Cheshire, 2003a] Cheshire, S. (2003a). DNS-Based Service Discovery. Internet-Draft (work in progress).

[Cheshire, 2003b] Cheshire, S. (2003b). Performing DNS queries via IP Multicast. Internet-Draft (work in progress).

[CLDC, 2000] CLDC (2000). *Connected, Limited Device Configuration (JSR-30).* SUN Microsystems.

[CLDC 1.1, 2003] CLDC 1.1 (2003). *Connected, Limited Device Configuration 1.1 (JSR-139).* SUN Microsystems.

[Cohen y Aggarwal, 98] Cohen, J. and Aggarwal, S. (98). General Event Notification Architecture Base. Internet-Draft (work in progress).

[Cugola et al., 1997] Cugola, G., Ghezzi, C., Pietro Picco, G., and Vigna, G. (1997). *Analyzing Mobile Code Languages. In Mobile Object Systems: Towards the Programmable Internet*, volume 1222 of *Lecture Notes in Computer Science.* Springer-Verlag.

[Czerwinski et al., 1999] Czerwinski, S. E., Zhao, B. Y., Hodes, T. D., Joseph, A. D., and Katz, R. H. (1999). An architecture for a secure service discovery service. In *Proc. Mobicom'99.*

[Dayem, 1997] Dayem, R. A. (1997). *Mobile data and wireless LAN technologies.* Prentice Hall.

[Dey, 2000] Dey, A. K. (2000). *Providing Architectural Support for Building Context-Aware Applications.* PhD thesis, College of Computing. Georgia Institute of Technology.

[Díez-Andino, 2002] Díez-Andino, G. (2002). Diseño e implementación de un servidor HTTP y mecanismos de serialización en J2ME. Master's thesis, Ingeniería en Informática. Escuela Politécnica Superior. Universidad Carlos III de Madrid.

[Díez-Andino et al., 2003a] Díez-Andino, G., García-Rioja, R. M., and Campo, C. (2003a). Design of a fipa-compliant agent platform for limited devices. In *5th International Workshop on Mobile Agents for Telecommunications Applications (MATA 2003)*, number 2881 in Lecture Notes in Computer Science (LNCS), pages 180–189, Marrakech, Morocco.

[Díez-Andino et al., 2003b] Díez-Andino, G., García-Rioja, R. M., and Campo, C. (2003b). Diseño de una plataforma de agentes compatible con FIPA para dispositivos limitados. In *IV Jornadas de Ingeniería Telemática (JITEL 2003)*, pages 221–228, Gran Canaria. España.

[Díez-Andino et al., 2003c] Díez-Andino, G., García-Rioja, R. M., and Campo, C. (2003c). Servidor HTTP para J2ME. In *I Congreso JavaHispano*, Leganés. Madrid.

[Diego Bartolomé et al., 2002] Diego Bartolomé, M. V. d., Gallego Pérez, D., López Mora, J. A., and Gómez Vicente, A. (2002). UMTS: hacia una red todo IP. *Comunicaciones de Telefónica I+D*, pages 85–106.

[Dobrev et al., 2002] Dobrev, P., Famolari, D., Kurzke, C., and Miller, B. A. (2002). Device and Service Discovery in Home Networks with OSGi. *IEEE Communications Magazine*, pages 86–92.

[Doufexi et al., 2002] Doufexi, A., Armour, S., Butler, M., Nix, A., Bull, D., and McGeehan, J. (2002). A Comparison of the HIPERLAN/2 and IEEE 802.11a wireless LAN Standards. *IEEE Communications Magazine*, pages 172–179.

[Droms, 1997] Droms, R. (1997). Dynamic Host Configuration Protocol.

[Dutta-Roy, 1999] Dutta-Roy, A. (1999). Networks for Homes. *IEEE Spectrum*, pages 26–33.

[Eastlake, 1999] Eastlake, D. (1999). RFC 2535: Domain Name System Security Extensions.

[Echelon, 1999] Echelon (1999). Introduction to the LonWorks System. Technical report, Echelon Corporation.

[Edens, 2001] Edens, G. T. (2001). Home Networking and the CableHome Project at CableLabs. *IEEE Communications Magazine*, pages 112–121.

[Esibov et al., 2003] Esibov, L., Adoba, B., and Thaler, D. (2003). Linklocal Multicast Name Resolution (LLMNR). Internet-Draft (work in progress).

[Esler et al., 1999] Esler, M., Hightower, J., Anderson, T., and Borriello, G. (1999). Next Century Challenges: Data-Centric Networking for Invisible Computing The Portolano Project at the University of Washington. In *Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 1999).*

[FAGOR, 2003] FAGOR (2003). Hogar Digital Fagor. http://www.fagor.com/es/domotic_n/.

[Finkenzeller, 2003] Finkenzeller, K. (2003). *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification.* John Wiley & Sons, 2nd edition.

[FIPA00001, 2002] FIPA00001 (2002). FIPA Abstract Architecture Specification. Status Standard.

[FIPA00014, 2002] FIPA00014 (2002). FIPA Nomadic Application Support Specification. Status Experimental.

[FIPA00023, 2002] FIPA00023 (2002). FIPA Agent Management Specification. Status Standard.

[FIPA00026, 2002] FIPA00026 (2002). FIPA Request Interaction Protocol Specification. Status Standard.

[FIPA00091, 2001] FIPA00091 (2001). FIPA Device Ontology Specification. Status Preliminary.

[FIPA00092, 2001] FIPA00092 (2001). FIPA Message Buffering Service Specification. Status Preliminary.

[FIPA00093, 2001] FIPA00093 (2001). FIPA Messaging Interoperability Service Specification. Status Preliminary.

[FP, 2002] FP (2002). *Foundation Profile (JSR-46)*. SUN Microsystems.

[Frank y Hollaway, 2000] Frank, E. H. and Hollaway, J. (2000). Connecting the Home with a Phone Line Network Chip Set. *IEEE MICRO*, pages 1–12.

[Frengle, 2001] Frengle, N. (2001). *I-Mode: A Primer*. John Wiley & Sons.

[Frodigh et al., 2000] Frodigh, M., Johansson, P., and Larsson, P. (2000). Wireless ad hoc networking-The art of networking without a network. *Ericsson Review*, pages 248–293.

[Gardner et al., 2000] Gardner, S., Markwalter, B., and Yonge, L. (2000). HomePlug Standard Brings Networking to the Home. Technical report, CommsDesign.com.

[Garlan et al., 2002] Garlan, D., Siewiorek, D. P., Smailagic, A., and Steenkiste, P. (2002). Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Personal Communications*, pages 22–31.

[Glass, 1999] Glass, G. (1999). *Mobility: Processes, Computers, and Agents*, chapter ObjectSpace Voyager Core Package Technical Overview, pages 612–627. Addison-Wesley.

[Glitho y Magedanz, 2002] Glitho, R. H. and Magedanz, T. (2002). Applicability of Mobile Agents to Telecommunications. *IEEE Network*.

[Glitho y Pierre, 2002] Glitho, R. H. and Pierre, S. (2002). Mobile Agents and Their Use for Information Retrieval: A Brief Overview and an Elaborate Case Study. *IEEE Network*.

[Goland, 1999] Goland, Y. Y. (1999). Multicast and Unicast UDP HTTP Requests. Internet-Draft (work in progress). draf-goland-http-udp-00.txt.

[Goland et al., 1999] Goland, Y. Y., Cai, T., Leach, P., and Gu, Y. (1999). Simple Service Discovery Protocol/1.0. Internet-Draft (work in progress). draft-cai-ssdp-v1-03.txt.

[Gong, 2001] Gong, L. (2001). JXTA: A Network Programming Environment. *IEEE Internet Computing*, pages 88–95.

[Goossens et al., 1994] Goossens, M., Mittelbach, F., and Samarin, A. (1994). *The Latex Companion*. Addison-Wesley, 2nd edition.

[Grasshopper, 1998] Grasshopper (1998). Grasshopper Technical Overview.

[Grasshopper MicroEdition, 2001] Grasshopper MicroEdition (2001). http://www.grasshopper.de/.

[Grimm et al., 2002] Grimm, R., Davis, J., Lemar, E., Macbeth, A., Swanson, S., Anderson, T., Bershad, B., Borriello, G., Gribble, S., and Wetherall, D. (2002). Programming for Pervasive Computing Enviroments.

[GSM 02.60, 1997] GSM 02.60 (1997). Digital cellular telecommunications system (Phase 2+); General Packet Radio Service (GPRS); Service description; Stage 1.

[Gulbrandsen et al., 2000] Gulbrandsen, A., Vixie, P., and Esibov, L. (2000). RFC 2782: A DNS RR for specifying the location of services (DNS SRV).

[Guttman, 2001] Guttman, E. (2001). RFC 3059: Attribute List Extension for the Service Location Protocol.

[Guttman y Kempf, 1999] Guttman, E. and Kempf, J. (1999). Automatic Discovery of Thin Servers: SLP, Jini and the SLP-Jini Bridge. In Press, I., editor, *25th Ann. Conf. IEEE Industrial Electronics Soc. (IECON 99)*, Piscataway, N. J.

[Guttman et al., 1999a] Guttman, E., Perkins, C., and Kempf, J. (1999a). RFC 2609: Service Templates and Service: Schemes.

[Guttman et al., 1999b] Guttman, E., Perkins, C., Veizades, J., and Day, M. (1999b). RFC 2608: Service Location Protocol, Version 2.

[Hansmann et al., 2001] Hansmann, U., Merk, L., Nicklous, M. S., and Stober, T. (2001). *Pervasive Computing Handbook.* Springer-Verlag.

[Hansmann et al., 2002] Hansmann, U., Mettala, R., Purakayastha, A., and Thompson, P. (2002). *SyncML: Synchronizing and Managing Your Mobile Data.* Prentice Hall.

[Helal, 2002] Helal, S. (2002). Standards for Service Discovery and Delivery. *IEEE Pervasive Computing*, pages 95–100.

[Hermann et al., 2001] Hermann, R., Husemann, D., Moser, M., Nidd, M., Rohner, C., and Schade, A. (2001). DEAPspace – Transient ad hoc networking of pervasive devices. *Computer Networks*, pages 411–428.

[Hightower et al., 2002] Hightower, J., Brumitt, B., and Borriello, G. (2002). The Location Stack: A Layered Model for Location in Ubiquitous Computing. In *4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002)*, Callicoon, NY. IEEE Computer Society Press.

[Hightower et al., 2001] Hightower, J., Vakili, C., Borriello, G., and Want, R. (2001). Design and Calibration of the SpotON Ad-Hoc Location Sensing System.

[HIPERLAN/2, 1999] HIPERLAN/2 (1999). ETSI Broadband radio access networks (BRAN); HIPERLAN type 2 technical specification; Physical layer.

[IBM Research Zurich, 1999] IBM Research Zurich (1999). Advanced Infrared. http://www.zurich.ibm.com/cs/wireless/ircommunication.html.

[IEEE 802.15.1, 2002] IEEE 802.15.1 (2002). IEEE Standard for information technology - Telecommunication and information exchange between systems - LAN/MAN - Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Wireless Personal Area Networks (WPANs).

[IEEE Std 1394-1995, 1996] IEEE Std 1394-1995 (1996). IEEE Standard for a High Performance Serial Bus.

[IEEE Std. 802.11, 1999] IEEE Std. 802.11 (1999). IEEE 802.11 Local and Metropolitan Area Networks: Wireless LAN Medium Acess Control (MAC) and Physical (PHY) Specifications.

[IEEE Std. 802.11a, 1999] IEEE Std. 802.11a (1999). Supplement to IEEE standard for information technology telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements. Part 11: wireless LAN Medium Access Control (MAC) and Physical layer (PHY) specifications: high-speed physical layer in the 5 Ghz band.

[IEEE Std. 802.11b, 1999] IEEE Std. 802.11b (1999). Supplement to IEEE standard for information technology telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements. part 11: wireless LAN Medium Access Control (MAC) and Physical layer (PHY) specifications. Amendment 2: higher-speed physical layer (PHY) extension in the 2.4 GHz band.

[IEEE Std. 802.11g, 2003] IEEE Std. 802.11g (2003). IEEE Std 802.11h-2003 (Amendment to IEEE Std 802.11, 1999 edn. (Reaff 2003)) as amended by IEEE Stds 802.11a-1999, 802.11b-1999, 802.11b-1999/cor 1-2001, and 802.11d-2001).

[IEEE Std. 802.11h, 2003] IEEE Std. 802.11h (2003). IEEE Std 802.11h-2003 (Amendment to IEEE Std 802.11, 1999 edn. (Reaff 2003)).

[IEEE Std 802.15.3, 2003] IEEE Std 802.15.3 (2003). IEEE standard for information technology - telecommunications and information exchange between

systems - local and metropolitan area networks - specific requirements part 15.3: wireless medium access control (MAC) and physical layer (PHY) specifications for high rate wireless personal area networks (WPANs).

[IEEE Std 802.15.4, 2003] IEEE Std 802.15.4 (2003). IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks specific requirements part 15.4: wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs).

[Java 2 Micro Edition, 1999] Java 2 Micro Edition (1999). Java 2 Platform Micro Edition. http://java.sun.com/j2me.

[Java APIs Bluetooth, 2002] Java APIs Bluetooth (2002). *Java APIs for Bluetooth (JSR-82).* SUN Microsystems.

[Jini, 1999] Jini (1999). Jini Architectural Overview. White Paper.

[Johansen et al., 1999] Johansen, D., Renesse, R. v., and Schneider, F. B. (1999). *Mobility: Processes, Computers, and Agents*, chapter Operating System Support for Mobile Agents, pages 557–566. Addison-Wesley.

[Jones et al., 2001] Jones, C. E., Sivalingam, K. M., Agrawel, P., and Chen, J. C. (2001). A Survey of Energy Efficient Network Protocols for Wireless Networks. *Wireless Networks*, 7(4):343–358.

[Kaaranen et al., 2001] Kaaranen, H., Ahtiainen, A., Laitinen, L., Naghian, S., and Niemi, V. (2001). *UMTS networks : architecture, mobility, and services.* John Wiley & Sons.

[Kapp, 2002] Kapp, S. (2002). 802.11: Leaving the Wire Behind. *IEEE Internet Computing*, pages 82–85.

[Karaoğz, 2001] Karaoğz, J. (2001). High-Rate Wireless Personal Area Networks. *IEEE Communications Magazine*, pages 96–102.

[Kempf y Goldschmidt, 2001] Kempf, J. and Goldschmidt, J. (2001). RFC 3082: Notification and Subscription for SLP.

[Kent y Atkinson, 1998] Kent, S. and Atkinson, S. (1998). RFC 2401: Security Architecture for the Internet Protocol.

[Khun-Jush et al., 2000] Khun-Jush, J., Malmgren, G., Schramm, P., and Torsner, J. (2000). Hiperlan type 2 for broadband wireless communication. *Ericsson Review*, pages 108–119.

[Khun-Jush et al., 2002] Khun-Jush, J., Schramm, P., Malmgren, G., and Torsner, J. (2002). HiperLAN2: Broadband Wireless Communications at 5 GHz. *IEEE Communications Magazine*, 40(6):130–136.

[Kotz et al., 1999] Kotz, D., Gray, R., Nog, S., Rus, D., Chawla, S., and Cybenko, G. (1999). *Mobility: Processes, Computers, and Agents*, chapter AGENT TCL: Targeting the Needs of Mobile Computers, pages 513–523. Addison-Wesley.

[KVM, 2000] KVM (2000). *Java 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices.* SUN Microsystems.

[Lamport, 1994] Lamport, L. (1994). *Latex: a document preparation system.* Addison-Wesley, 2nd edition.

[Lange y Oshima, 1998] Lange, D. B. and Oshima, M. (1998). *Programming and Developing Java Mobile Agents with Aglets.* Addison-Wesley.

[Law y Kelton, 2000] Law, A. M. and Kelton, W. D. (2000). *Simulation Modeling and Analysis.* Mc Graw Hill, 3rd edition. Excellent book on simulation, including necessary mathematics on statistics. Make sure you get the 3rd edition, earlier editions use FORTRAN as programming language. Quite expensive, though.

[Lawrence, 2002] Lawrence, J. (2002). Response to the FIPA Call for Technology agents in ad-hoc environments. Response to 1st AdHoc Call for Technology. http://www.fipa.org/docs/input/f-in-00062.

[Lea et al., 2000] Lea, R., Gibbs, S., Dara-Abrams, A., and Eytchison, E. (2000). Networking home entertainment devices with HAVi. *IEEE Computer*, pages 35–43.

[LEAP Project, 2000] LEAP Project (2000). Deliverable D31: Specifications of the LEAP Architecture.

[Letsch, 2000] Letsch, T. (2000). Redesign and implementation of a mobile agent system compliant with the maffinder part of masif stanadard. Master's thesis, Technische Universitat Munchen. Institut fur Informatik.

[Margherita2000.com, 2000] Margherita2000.com (2000).

[Marples y Kriens, 2001] Marples, D. and Kriens, P. (2001). The Open Services Gateway Initiative: An Introduction Overview. *IEEE Communications Magazine*, pages 110–114.

[Matthes et al., 1994] Matthes, F., Müssig, S., and Schmidt, J. (1994). Persistent Polymorphic Programming in Tycoon: An Introduction. Technical report, FIDE Project Coordinator, Dept. of Computing Sciences, University of Glasgow, Glasgow.

[Micro FIPA-OS, 2001] Micro FIPA-OS (2001). http://fipa-os.sourceforge.net.

[MIDP, 2000] MIDP (2000). *Mobile Information Device Profile (JSR-37).* SUN Microsystems.

[MIDP 2.0, 2002] MIDP 2.0 (2002). *Mobile Information Device Profile (JSR-118).* SUN Microsystems.

[Mihailescu y Kendall, 2002] Mihailescu, P. and Kendall, E. A. (2002). MAE: A Mobile Agent Platform for Building Wireless M-Commerce Applications. In *8th ECOOP Workshop on Mobile Object Systems: Agent Applications and New Frontiers*, Spain.

[Miller y Pascoe, 1999] Miller, B. and Pascoe, R. (1999). Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer. http://www.salutation.org/whitepaper/BtoothMapping.PDF.

[Miller y Pascoe, 2000] Miller, B. A. and Pascoe, R. A. (2000). Salutation service discovery in pervasive computing environments. Technical report, IBM White Paper.

[Mobile Media API, 2003] Mobile Media API (2003). *Wireless Messaging API (JSR-135).* SUN Microsystems.

[Mockapetris, 1987a] Mockapetris, P. (1987a). Domain Names - Concepts and Facilities.

[Mockapetris, 1987b] Mockapetris, P. (1987b). Domain Names - Implementation and Specification.

[MODSIM, 1997] MODSIM (1997). *Reference Manual MODSIM III.* CACI Products Company.

[Moore, 1965] Moore, G. E. (1965). Cramming More Components into Integrated Circuits. *Electronics*, 38(8).

[Muller et al., 2001] Muller, F., Sorelius, J., and Turina, D. (2001). Further evolution of the gsm/edge radio access network. *Ericsson Review*, pages 116–123.

[Negus et al., 2000] Negus, K. J., Stephens, A. P., and Lansforb, J. (2000). HomeRF: Wireless Networking for the Connected Home. *IEEE Personal Communications*, pages 20–27.

[Nidd, 2001] Nidd, M. (2001). Service Discovery in DEAPspace. *IEEE Personal Communications*.

[Nwana, 1996] Nwana, H. S. (1996). Software Agents: an overview. *Knowledge Engineering Review*, 1(3):205–244.

[Ortiz, 2002] Ortiz, E. C. (2002). A Survey of J2ME Today. Technical report, Wireless Java.

[Papastravrou et al., 1999] Papastravrou, S., Samaras, G., and Pitoura, E. (1999). Mobile agents for WWW distributed database access. In *Proceedings of the International Conference on Data Engineering (ICDE99)*.

[Partridge et al., 2000] Partridge, K., Arnstein, L., Borriello, G., and Whitted, T. (2000). Fast Intrabody Signaling (Demostration). In *3rd Workshop on Mobile Computer Systems and Applications (WMCSA 2000)*, Monterey, California.

[Pascoe, 1999] Pascoe, B. (1999). Salutation-Lite. Find-And-Bind Technologies For Mobile Devices. Technical report, Salutation Consortium.

[Pawlikowski, 1990] Pawlikowski, K. (1990). Steady–state simulation of queueing processes: A survey of problems and solutions. *ACM Computing Surveys*, 22(2).

[PBP, 2002] PBP (2002). *Personal Basis Profile (JSR-129)*. SUN Microsystems.

[PDAP, 2003] PDAP (2003). *Personal Digital Assistant Profile (JSR-75)*. SUN Microsystems.

[Peine y Stolpmann, 1999] Peine, H. and Stolpmann, T. (1999). *Mobility: Processes, Computers, and Agents*, chapter The Architecture of the Ara Platform for Mobile Agents, pages 583–595. Addison-Wesley.

[Perea, 2003] Perea, J. C. (2003). Implementación de los protocolos de descubrimiento slp y pdp en j2me. Master's thesis.

[Perkins y Guttman, 1999] Perkins, C. and Guttman, E. (1999). RFC 2610: DHCP Options for Service Location Protocol.

[Plummer, 1982] Plummer, D. C. (1982). An Ethernet Address Resolution Protocol.

[PP, 2002] PP (2002). *Personal Profile (JSR-62).* SUN Microsystems.

[Qi et al., 2001] Qi, H., Iyengar, S., and Chakrabarty, K. (2001). Multi-resolution data integration using mobile agents in distributed sensor networks. *IEEE Trans. Syst. Man. Cybern. C*, 31:383–391.

[Ratsimor et al., 2002] Ratsimor, O., Chakraborty, D., Tolia, S., Khushraj, D., Gupta, G., Kunjithapatham, A., Joshi, A., and Finin, T. (2002). Allia: Police-based Alliance Formation for Agents in Ad hoc Environments. Response to 1st AdHoc Call for Technology. http://www.fipa.org/docs/input/f-in-00061.

[Reddy et al., 99] Reddy, S., Lowry, D., Reddy, S., Henderson, R., Davis, J., and Babich, A. (99). DAV Searching & Locating. Internet-Draft (work in progress).

[RFC 2165, 1997] RFC 2165 (1997). RFC 2165: Service Location Protocol.

[Richard III, 2000] Richard III, G. G. (2000). Service Advertisement and Discovery: Enabling Universal Device Cooperation. *IEEE Internet Computing*, pages 18–27.

[Rose, 2001] Rose, B. (2001). Home Networks: A Standards Perpective. *IEEE Communications Magazine*, pages 78–85.

[Salutation Consortium, 1998] Salutation Consortium (1998).

[Satyanarayanan, 1996] Satyanarayanan, M. (1996). Mobile Information Access. *IEEE Personal Communications*, pages 26–33.

[Satyanarayanan, 2001] Satyanarayanan, M. (2001). Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 8(4):10–17.

[Schilit et al., 1993] Schilit, B., Adams, N., Gold, R., Tso, M., and Want, R. (1993). The PARCTAB Mobile Computing System. In *Fourth Workshop on Workstation Operating Systems*, pages 34–39.

[SDP, 2001] SDP (2001). *Bluetooth Specification v1.1, Part E: Service Discovery Protocol (SDP).* http://www.bluetooth.com/dev/specifications.asp.

[Singhal, 2001] Singhal, S. (2001). *WAP The Wireless Application Protocol : Writing Applications for the Mobile Internet.* Addison-Wesley.

[Suvak, 2000] Suvak, D. (2000). IrDA and Bluetooth: A Complementary Comparison. Technical report, Extended Systems, Inc.

[Tschudin, 1994] Tschudin, C. F. (1994). An Introduction to the MO Messenger Language. Technical report, University of Geneva, Switzerland.

[Vixie et al., 2000] Vixie, P., Thomson, S., Rekhter, Y., and Bound, J. (2000). Dynamic Updates in the Domain Name System (DNS UPDATE).

[Wacks, 2002] Wacks, K. (2002). Home Systems Standards: Achievements and Challenges. *IEEE Communications Magazine*, pages 152–159.

[Weiser, 1991] Weiser, M. (1991). The Computer for the 21st Century. *Scientific American.*

[WG-AdHoc, 2002] WG-AdHoc (2002). Agents in Ad Hoc Environments. A Whitepaper.

[White, 1995] White, J. E. (1995). Telescript Technology: Mobile Agents. Technical report, General Magic.

[Williams, 2000] Williams, S. (2000). IrDA: Past, Present and Future. *IEEE Personal Communications*, pages 11–19.

[Willians, 2002] Willians, A. (2002). Requirements for Automatic Configuration of IP Hosts. Internet-Draft (work in progress).

[Wireless Messaging API, 2003] Wireless Messaging API (2003). *Wireless Messaging API (JSR-120).* SUN Microsystems.

[X10, 1997] X10 (1997). http://www.x10.com.

[Zhao et al., 2001] Zhao, W., Schulzrinne, H., Guttman, E., Bisdikian, C., and Jerome, W. (2001). RFC 3421: Select and Sort Extensions for the Service Location Protocol (SLP).