Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Soporte para la interacción de usuarios nómadas con áreas autónomas

Tesis que presenta

Víctor Alberto Gómez Pérez

para obtener el Grado de

Maestro en Ciencias

en Computación

Director de la Tesis

Dra. Sonia Guadalupe Mendoza Chapa

México, D.F. Diciembre 2009

Resumen

La computación ubicua integra soportes de red (desde pequeños sensores hasta dispositivos poderosos y dinámicos) en el entorno laboral e incluso en el entorno doméstico de las personas. Un área autónoma puede contener dispositivos que proporcionen información sobre el estado de cada artefacto (e.g., la falla de energía de un refrigerador) sin requerir la intervención del usuario. Los sistemas de descubrimiento de servicios son esenciales para lograr esta sofisticación porque permiten a dispositivos y servicios computacionales descubrir, configurar y comunicarse con otros. De otra manera, los usuarios de sistemas ubicuos podrían frustrarse fácilmente cuando traten de configurar dichos dispositivos y servicios para hacerlos interactuar. Precisamente, los sistemas de descubrimiento de servicios pretenden reducir esa frustración, facilitando el uso de dispositivos móviles en una red. Sin embargo, la mayoría de los sistemas de descubrimiento únicamente se han enfocado en dar soporte a la interacción entre servicios en términos de topología de red y localización. Además, dichos sistemas están diseñados principalmente para entornos domésticos (e.g., localización de servicios multimedia para permitir a un usuario recibir una video-llamada) y empresariales (e.g., determinación de los servicios disponibles para el tratamiento de una solicitud de impresión). Por lo tanto, las soluciones propuestas por estos sistemas difícilmente pueden ser reutilizadas en otros ámbitos. Sin embargo, los distintos enfoques de solución proporcionan puntos de referencia útiles para el diseño de futuros sistemas de descubrimiento. En esta tesis de maestría, se describe un nuevo sistema de descubrimiento de servicios llamado SEDINU (*SErvice DIscovery for Nomadic Users*), el cual ofrece un soporte computacional y comunicacional para facilitar la interacción entre un usuario nómada y los servicios ofrecidos por el área actual donde él se encuentra, así como la interacción entre usuarios nómadas inmersos en dicha área. Estos tipos de interacción dependen de un contexto específico, definido en términos de rol, localización y objetivos de los usuarios. El sistema propuesto permite a un usuario nómada: 1) seleccionar y ejecutar un servicio cuantas veces sea necesario; 2) interactuar con otros usuarios nómadas; 3) disponer de aplicaciones (formularios) para interactuar fácilmente con los servicios ofrecidos por el área autónoma actual, e.g., provisión de información requerida por esta área como identificación y perfil del usuario; 4) crear redes *ad hoc* (dentro del área actual) para permitir la transferencia de información entre usuarios nómadas y los empleados de la organización, e.g., transferencia de recibos electrónicos y aprobación de entrega de equipos.

Palabras clave: interacción entre usuarios nómadas, entornos ubicuos, descubrimiento de servicios, áreas autónomas, redes *ad hoc*.

Abstract

Ubiquitous computing integrates network supports (from small sensors to powerful and dynamic devices) into the people's working and domestic environments. An autonomous area can contain devices that provide information about the state of each artifact (e.g., power failure of a refrigerator) without needing user intervention. Service discovery systems are essential to achieve this sophistication as they allow computing devices and services to discover, configure and communicate with others. Otherwise, users of ubiquitous systems could easily become frustrated when trying to configure such devices and services to make them interact. Precisely, service discovery systems aim to reduce that frustration by facilitating the use of mobile devices on a network. However, most service discovery systems have only focused on proving support for service interaction in terms of network topology or localization. Moreover, these systems are mainly intended for domestic environments (e.g., localization of multimedia services to allow a user to receive a video call) and business environments (e.g., determination of the available services to treat a printing request). Therefore, the solutions proposed by these systems cannot be easily reused in other contexts. However, the different solution approaches provide useful reference points for the design of future discovery systems. In this master thesis, we describe a new service discovery system called SEDINU (*SErvice DIscovery for Nomadic Users*), which provides a computational and communicational support to facilitate the interaction between a nomadic user and the services provided by the current area where he is located, as well as the interaction between nomadic users located in such an area. These kinds of interactions depend on a specific context, defined in terms of users' role, location and goals. The proposed system allows a nomadic user: 1) to select and to execute a service as many times as he needs; 2) to interact with other nomadic users; 3) to have applications (formularies) to easily interact with the services offered by the current autonomous area, e.g., provision of information required by this area, such as user identification and profile; 4) to create ad-hoc networks (within the current area) to allow information transfer between nomadic users and the organization employees, e.g., transfer of e-vouchers and equipment delivery approval.

Keywords: interaction between nomadic users, ubiquitous environments, service discovery, autonomous areas, ad hoc networks.

Agradecimientos

"Para mí, la cosa más importante en este mundo no es dónde nos encontramos sino el rumbo al que nos dirigimos. Para llegar al puerto del paraíso a veces tenemos que navegar a favor del viento y a veces en contra, pero debemos navegar y no ir a la deriva y tampoco estar anclados"

Oliver Wendell Holmes Jr.

Esta tesis de maestría, ha requerido de mucho esfuerzo y dedicación por parte del autor y su directora de tesis. No hubiera sido posible su terminación sin la ayuda de todas y cada una de las personas que a continuación citaré y muchas de las cuales han sido un soporte importante en momentos difíciles.

A mi asesora de tesis de maestría, la Dra. Sonia G. Mendoza Chapa, quien ha venido guiando desde hace un año mi formación no solamente académica, sino como persona. Gracias Dra. Sonia, por sus sabios consejos.

A mis sinodales, Dr. Dominique Decouchant y Dr. José G. Rodríguez García por el tiempo invertido en la revisión de este documento y por sus valiosos comentarios para mejorar el mismo.

A las secretarias del Departamento de Computación, Sofía Reza Cruz y Felipa Rosas López, por su gran ayuda en trámites administrativos.

Quiero dar gracias a Dios, por estar conmigo siempre en cada paso que doy, por fortalecer mi corazón e iluminar mi mente con sabiduría y por haber puesto en mi camino a aquellas personas que han sido mi soporte, mi luz y mi fuerza.

A mis padres Víctor y Rocío y mis hermanas Arelis y Cristel porque el amor, ánimo, apoyo y alegría que me brindan me dan la fortaleza necesaria para seguir siempre adelante.

A mis abuelitos Abel y Amparo y a mis tíos Abel, Artemio, Guadalupe, Elvia y Alma por su amor y cariño que desde pequeño me han dado.

A mi amigo Roberto Enrique Alberto Lira, quien desde hace siete años me ha brindado su invaluable amistad, muchas gracias Roberto.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1 Contexto de investigación

En los últimos años, hemos sido testigos de importantes cambios derivados de la utilización de la tecnología de las telecomunicaciones (e.g., el creciente acceso a redes de área amplia mediante conexiones cableadas/inalámbricas o telefonía móvil) para satisfacer algunas necesidades de asistencia social (e.g., asistencia de salud domiciliaria a distancia) y de comunicación (e.g., transmisión de mensajes multimedia). Los beneficios de esta tecnología también se hacen evidentes en entornos laborales, donde los usuarios pueden acceder fácilmente a diversos servicios (e.g., consulta a bases de datos, navegación en la Web e impresión). En tales entornos, se puede asumir que la conectividad proporcionada por las redes corporativas es fiable, continua y ofrece un alto ancho de banda.

Sin embargo, debido al creciente uso de dispositivos móviles (e.g., *smartphones*, computadoras portátiles y PDAs) para interactuar con los servicios disponibles en el entorno, las personas tienden a volverse nómadas. A medida que se desplazan, las personas pueden encontrar cambios sustanciales referentes: 1) a la forma de interacción con los dispositivos móviles, 2) a los tipos de servicios disponibles, 3) al medio de comunicación y 4) al ancho de banda. Los dispositivos móviles no sólo se vuelven más pequeños, baratos y poderosos sino también ejecutan más aplicaciones y servicios de red. Estas características incrementan el uso del cómputo móvil y ubicuo.

1.1.1 Cómputo móvil y ubicuo

El avance tecnológico del cómputo móvil ha conducido a un cambio revolucionario en la sociedad. Estamos pasando de la era de la computadora personal (i.e., un dispositivo de cómputo por persona) a la era del cómputo ubicuo en la que un usuario emplea simultaneamente varios dispositivos de cómputo heterogéneos para acceder, en cualquier momento, a la información requerida [Bardram, 2007].

El cómputo móvil y el cómputo ubicuo representan pasos evolutivos importantes en la línea de investigación de los sistemas distribuidos, la cual se remonta a la década de 1970. En la Figura 1.1 se ilustra esta evolución: a medida que uno se mueve de izquierda a derecha en esta figura, se presentan nuevos problemas. Sin embargo, las soluciones de muchos problemas encontrados previamente se vuelven cada vez más complejas. Como in-

dican los símbolos de modulación, este aumento en complejidad es multiplicativo en vez de aditivo, e.g., resulta mucho más difícil diseñar e implementar un sistema de cómputo móvil que un sistema distribuido de robustez y madurez comparables, en tanto que un sistema de cómputo ubicuo es aún más difícil de desarrollar. Como se muestra en la Figura 1.1, el marco conceptual y algorítmico de los sistemas distribuidos proporciona una base sólida para el cómputo móvil y ubicuo.

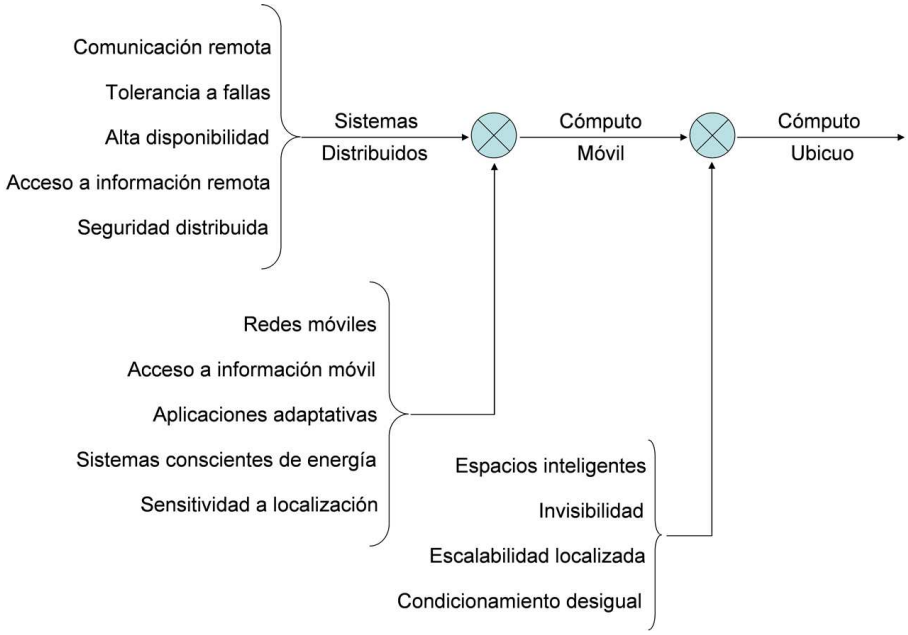

Figura 1.1: Evolución del cómputo móvil y ubicuo a partir del desempeño de TCP en sistemas distribuidos

El cómputo móvil nace en la década de 1990 con el advenimiento de las computadoras portátiles y de las redes LAN inalámbricas. Aunque muchos principios básicos del diseño de sistemas distribuidos continúan aplicándose, cuatro principales limitaciones de la movilidad obligan al desarrollo de técnicas especializadas. Estas cuatro limitaciones son: 1) variación impredecible en la calidad de servicio de la red, 2) poca solidez y confianza en los elementos móviles; 3) limitaciones impuestas a los recursos locales por el peso y las restricciones del tamaño de los archivos; y 4) preocupación por el consumo de energía de la batería. El cómputo móvil es un campo de investigación y desarrollo muy activo. Los resultados alcanzados hasta ahora pueden ser agrupados en los cinco temas siguientes: 1) creación de redes móviles, 2) acceso a información móvil, 3) soporte para aplicaciones adaptativas, 4) técnicas de ahorro de energía a nivel de sistema y 5) localización de dispositivos.

En el año 2000, las investigaciones en cómputo móvil comenzaron a abordar lo que ahora se conoce como cómputo ubicuo. El manifiesto que fundamenta el cómputo ubicuo es un artículo de 1991 titulado "The Computer for the 21st Century" escrito por Mark Weiser de Xerox PARC, quien señaló que "*las tecnologías más profundas son las que*

desaparecen... se tejen ellas mismas dentro de la vida cotidiana hasta que se vuelven indistinguibles" [Weiser, 2002]. Cuando el artículo se publicó en la década de 1990, proponía una visión demasiado avanzada para su tiempo. La tecnología de *hardware* necesaria para lograr esta visión, simplemente no existía. Ahora, casi 20 años después, la computación y las telecomunicaciones inalámbricas necesarias para su realización son cada vez más accesibles.

Como se muestra en la Figura 1.1, el cómputo ubicuo tiene en común varios temas de investigación con el cómputo móvil. Además, aborda temas clave como espacios inteligentes, invisibilidad, escalabilidad localizada y condicionamiento desigual.

El presente trabajo de investigación se inscribe en el ámbito del cómputo ubicuo (ver Figura 1.2) cuyo objetivo es incrementar el uso de sistemas computacionales a través del entorno físico, haciéndolos disponibles pero invisibles al usuario. Por lo tanto, el cómputo ubicuo pretende desarrollar sistemas inteligentes que se adapten al usuario y que se utilicen de manera intuitiva.

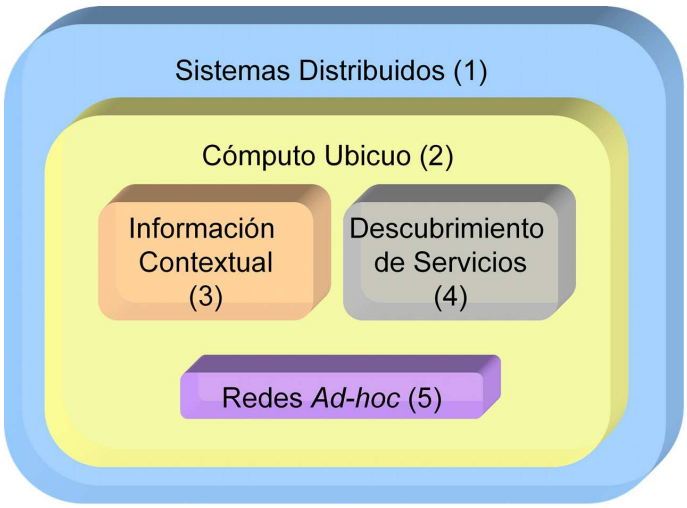

Figura 1.2: Contexto de investigación de la presente tesis de maestría

La tecnología requerida para diseñar sistemas ubicuos se presenta en tres partes: 1) dispositivos de bajo costo; 2) computadoras con poco poder de procesamiento pero con grandes despliegues y *software* para soportar sistemas ubicuos; y 3) una red que vincule todos estos componentes. Las tendencias actuales sugieren que los dos primeros requisitos se están cumpliendo [Weiser, 2002].

Aunque los procesadores y despliegues actuales son capaces de soportar el cómputo ubicuo, las tendencias del *software* y de las redes de computadoras plantean problemas mayores, e.g., las implementaciones actuales de los sistemas distribuidos hacen simplemente que los servidores de archivos en red, impresoras y otros dispositivos se vean como si estuvieran conectados directamente a la computadora del usuario.

Sin embargo, aún no se han explotado las capacidades de computadoras físicamente dispersas ni la información incorporada en ellas para conocer el lugar en dónde se localiza un dispositivo particular. La red que conectará el *hardware* y el *software* ubicuos plantea

otros desafíos (e.g., la velocidad de transmición de datos) aunque hoy en día las tasas de transmisión de datos de las redes inalámbricas están aumentando rápidamente.

Un estudio realizado por Cahners In-Stat Group[1] en el año 2003 pronosticó que el número de suscriptores a servicios de datos inalámbricos en el mundo crecería rápidamente de 170 millones en 2000 a más de 1.3 billones en 2004, en tanto que el número de mensajes inalámbricos enviados por mes aumentaría de 3 billones en diciembre de 1999 a 244 billones en diciembre de 2004 [Chlamtac et al., 2003]. Actualmente, la mayoría de las conexiones entre dispositivos inalámbricos se realiza a través de una infraestructura fija basada en proveedores de servicios o redes privadas.

Las redes inalámbricas constituyen la tecnología de interconexión más adecuada a la naturaleza móvil de los nuevos dispositivos de cómputo. Así, los usuarios de *smartphones* pueden consultar su correo electrónico en cualquier lugar; los viajeros pueden navegar en la Web dentro de los aeropuertos; los turistas pueden utilizar GPS para localizar museos, restaurantes o calles; los científicos pueden intercambiar documentos durante una conferencia; las personas en casa pueden transferir y sincronizar información entre su dispositivo móvil y su PC. Sin embargo, los usuarios nómadas requieren sistemas de descubrimiento eficientes para encontrar los servicios disponibles en una red.

1.1.2 Información contextual, descubrimiento de servicios y redes *ad hoc*

El cómputo ubicuo da soporte a la información contextual y al descubrimiento de servicios para realizar sus funciones e.g., detectar los cambios de ubicación de un usuario nómada y descubrir los servicios disponibles en el entorno en función de su ubicación (ver Figura 1.2). La información contextual [Dey, 2000] es aquella que caracteriza a una entidad la cual puede ser una persona, un lugar o un objeto que se considera relevante a la interacción entre un usuario y una aplicación. En la práctica, ciertos tipos de información contextual son más importantes que otros tales como la ubicación, la identidad, el tiempo y la actividad. Estos tipos de información contextual caracterizan la situación de una entidad particular, ya que no sólo responden a las preguntas ¿Quién? ¿Qué? ¿Cuándo? y ¿Dónde? sino también actúan como indicios de otras fuentes de información contextual, por ejemplo:

a) dada la identidad de una persona, se puede adquirir más información relacionada con ella, tal como número de teléfono, dirección postal, dirección de correo electrónico, fecha de nacimiento, lista de amigos y relaciones con otras personas;

b) a partir de la ubicación de una entidad, se puede determinar qué objetos o personas se encuentran a su alrededor y qué actividades se están produciendo en torno a esta entidad.

Por otra parte, el descubrimiento de servicios es el proceso mediante el cual una entidad es notificada de forma espontánea de la disponibilidad de servicios en una red, i.e., es un mecanismo para referenciar dinámicamente a un servicio. Estas referencias manejan un

[1]Importante firma consultora y de investigación de la industria inalámbrica. http://www.instat.com/

tipo de información que el cliente puede utilizar posteriormente para ponerse en contacto con el servicio [Edwards, 2006].

Una característica importante de los entornos ubicuos es la creación de redes *ad hoc* debido a la naturaleza móvil de las conexiones inalámbricas (ver Figura 1.2). Recientemente, han aparecido formas alternativas de prestar algún servicio. Estas formas se enfocan en conectar dispositivos móviles entre sí por medio de una red móvil *ad hoc* flexible y de gran alcance. De esta manera, los nodos móviles no sólo pueden comunicarse entre sí, sino también pueden acceder a servicios de Internet a través de un nodo *gateway*.

Las redes *ad hoc* móviles se forman dinámicamente por medio de un sistema autónomo de nodos móviles que están conectados a través de enlaces inalámbricos, sin utilizar la infraestructura de red existente o una administración centralizada [Chlamtac et al., 2003]. Como las redes inalámbricas siguen evolucionando, se espera que algunas capacidades de las redes *ad hoc* (e.g., la velocidad de transmisión de datos) se vuelvan más importantes. Las soluciones tecnológicas para apoyar de forma más significativa la investigación futura y los esfuerzos de desarrollo pueden esperarse de la industria y de la academia, por igual.

1.2 Planteamiento del problema

Una red hace disponibles a los servicios a través de su registro con el fin de ser localizados, mediante la búsqueda de un servicio de directorio o simplemente la notificación periódica de anuncios en la red. Durante este proceso de registro, los servicios aportan información general (e.g., su tipo y sus atributos), así como información que el cliente tendrá que utilizar para localizarlos (e.g., una dirección IP y un número de puerto).

Por su parte, los clientes establecen criterios que describen los servicios en los que están interesados. Estos criterios son utilizados por el sistema de descubrimiento para determinar los servicios adecuados que satisfagan la solicitud de un cliente. Este proceso de búsqueda implica la consulta de un directorio o anuncios de solicitud de servicios. Como el registro y la cancelación de servicios son operaciones dinámicas, la mayoría de los sistemas de descubrimiento notifican asincrónicamente a sus clientes de los servicios disponibles. Sin embargo, los clientes de estos servicios son principalmente programas, e.g., un servicio de impresión puede ser registrado en un servidor de búsqueda como un objeto *proxy* que actuará como control remoto. Por lo tanto, estos sistemas no soportan interacciones entre los usuarios y los servicios presentes en un entorno ni mucho menos interacciones entre usuarios.

En los últimos años, diversas organizaciones han diseñado y desarrollado sistemas de descubrimiento de servicios. En el ámbito académico se puede mencionar Intentional Naming System (INS) [Adjie-Winoto et al., 1999] del Instituto Tecnológico de Massachusetts y Ninja Service Discovery Service (SDS) [Herborn et al., 2005] de la Universidad de California en Berkeley. De igual manera, los principales proveedores de software ofrecen sistemas de descubrimiento de servicios "integrados" a sus actuales sistemas operativos, e.g., Jini Network Technology [Oaks and Wong, 2000] de Sun Microsystem, Universal Plug and Play (UPnP) [Jeronimo and Weast, 2003] de Microsoft y Bonjour [White, 2007] de Apple. Algunas otras organizaciones de diversa índole también han propuesto sistemas de descubrimiento, e.g., DEAPSpace [Nidd, 2001] de IBM Re-

search, Salutation [Jamalipour, 2003] del Consorcio Salutation, Service Location Protocol (SLP) [Zhao and Schulzrinne, 2005] de Internet Engineering Task Force y Bluetooth SDP [Chang et al., 2007] del Grupo de Interés Especial en Bluetooth.

Todos estos sistemas dan soporte a entornos ubicuos en términos de topología de red o ubicación [Zhu et al., 2005]. Cada uno aborda diferentes cuestiones, pero la mayoría están diseñados para ámbitos domésticos o empresariales. Aunque estos sistemas, no se orientan al cómputo ubicuo más allá de estos fines, los distintos enfoques de diseño proporcionan puntos de referencia útiles para el futuro diseño de sistemas de descubrimiento. En consecuencia, se observa que:

- la mayoría de estos sistemas únicamente se han enfocado en dar soporte a la interacción entre servicios computacionales con el fin de realizar una tarea específica;

- la interacción de un usuario con los servicios ofrecidos por el entorno, así como la interacción entre usuarios inmersos en dicho entorno, bajo un contexto específico, no han sido contempladas.

1.3 Objetivos del proyecto

Este proyecto de tesis de maestría se basa en el concepto de "área autónoma", la cual se define como una entidad lógica que proporciona servicios especializados y que maneja sus recursos de forma independiente, con el fin de ayudar a los usuarios nómadas en la realización de sus objetivos parciales y globales. Un área autónoma gestiona los recursos de su dominio, mientras observa las políticas de la organización (e.g., seguridad y facturación). La gestión de los recursos de un área autónoma incluye: información de los servicios disponibles, información contextual del área, roles de usuario, intercambio de información con otras áreas de la organización y plan de actividades.

Objetivo General

Diseñar e implementar el sistema SEDINU (*SErvice DIscovery for Nomadic Users*) que permita a los usuarios nómadas, inmersos en un área autónoma, interactuar con otros usuarios y con los servicios disponibles en dicha área. La interacción está regida por un plan de actividades que depende del contexto actual (ubicación, rol y objetivos del usuario).

Objetivos Particulares

- Desarrollar una red *ad hoc* que permita al usuario nómada interactuar con los servicios disponibles en el área autónoma (e.g., solicitud de información) y con otros usuarios nómadas (e.g., intercambio de información).

- Completar el desarrollo de dos aplicaciones (*Instant Messenger* y un sistema de transferencia de archivos, ambos utilizando el protocolo *Bluetooth*) que permitirán explotar la red *ad hoc*.

1.4 Organización del documento

Este documento está estructurado en cinco capítulos (ver Figura 1.3). Después de haber presentado el contexto de investigación donde se ubica la presente tesis de maestría y de haber planteado el problema que se pretende resolver, así como los objetivos que se persiguen, se expone el estado del arte en el capítulo 2. Particularmente, se explica el protocolo Bluetooth, el cual ha sido utilizado para la creación de redes *ad hoc* cuyos fundamentos se detallan en este mismo capítulo. También se analizan las características, ventajas y desventajas de los principales sistemas de descubrimiento de servicios reportados en la literatura científica.

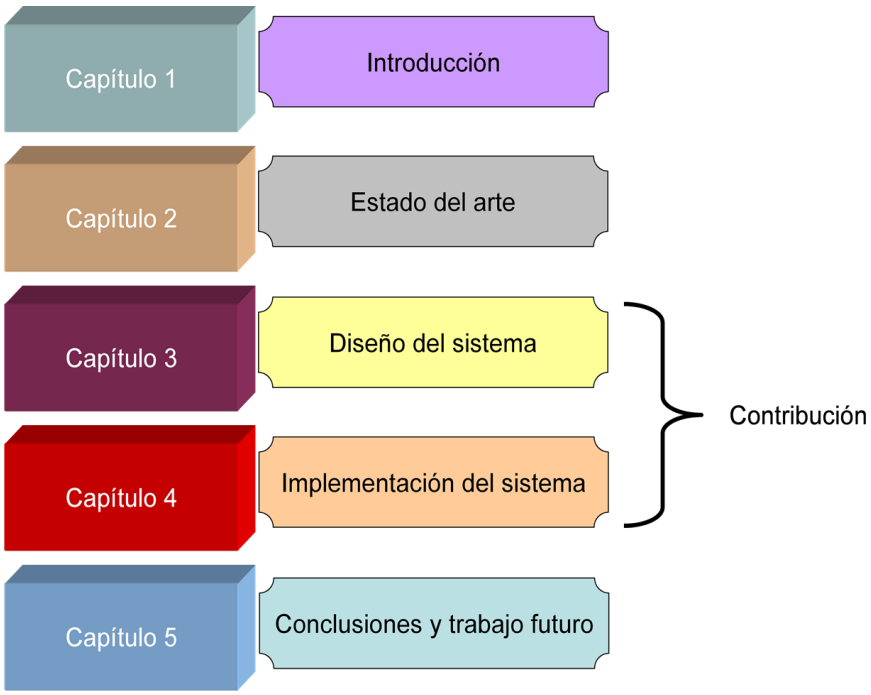

Figura 1.3: Organización del documento

En los capítulos 3 y 4 se describe la contribución del presente trabajo de investigación. Específicamente, en el capítulo 3 se detalla el análisis y el diseño del sistema SEDINU. En este capítulo, primeramente se describe el concepto de áreas autónomas, en el que se basa el diseño de los módulos del sistema propuesto. A continuación, se presenta un escenario de uso que pone en evidencia los beneficios que SEDINU ofrecería a sus usuarios, si se implantara en una organización. Posteriormente, se detalla las arquitecturas funcional y de distribución de este sistema, así como los módulos constituyentes, los cuales se describen brevemente a continuación:

a) *Location Detector*, encargado de detectar las coordenadas actuales de un usuario nómada y de enviarlas al sistema de administración de flujos de trabajo para que sean procesadas;

b) *Tiny-SEDINU Application*, que ayuda a los usuarios nómadas a interactuar con los servicios disponibles en el área autónoma actual o con otros usuarios nómadas;

c) *Service Manager*, responsable de establecer una red *ad hoc* para acceder a los servicios disponibles;

d) *Ad-hoc Creator*, encargado de inicializar de forma dinámica una red *ad hoc*: 1) entre el módulo *Tiny-SEDINU Application*, que se ejecuta en los dispositivos móviles de los usuarios y el módulo *Service Manager* o 2) entre dos módulos *Tiny-SEDINU Application*, que se ejecutan en los dispositivos móviles de los usuarios, quienes pueden iniciar una interacción/colaboración.

En el capítulo 4, se describe los detalles técnicos de la implementación del módulo *Ad hoc Creator* del sistema SEDINU. En primer lugar, se justifica la selección de la plataforma de desarrollo empleada para la implementación de este módulo. A continuación, se describe las bases de la programación en Java ME (*Micro Edition*), utilizada para la creación de dos aplicaciones Bluetooth (Aplicación Eco Cliente-Servidor e *Instant Messenger*) que constituyen la base de la implementación del módulo *Ad hoc Creator*, el cual se describe al final de este capítulo.

Finalmente, en el capítulo 5, se presentan las conclusiones de la presente tesis de maestría, así como también algunas ideas de trabajo a futuro.

Capítulo 2

Estado del Arte

En este capítulo se describen los trabajos relacionados con el tema de investigación que se estudia en esta tesis de maestría. Primeramente, se describe la tecnología Bluetooth y el medio de comunicación que maneja (sección 2.1). Posteriormente, se hace un estudio detallado de las redes *ad hoc* (sección 2.2) las cuales han sido utilizadas para ofrecer un soporte de interacción, basado en Bluetooth, entre los dispositivos de los usuarios nómadas, así como también entre dichos dispositivos y un *host* de servicios. Finalmente, se hace un estudio de las principales características de los sistemas de descubrimiento de servicios más relevantes en la literatura científica, tales como *Service Location Protocol*, *Universal Plug and Play*, *Bonjour* y *Jini*, entre otros (sección 2.3).

2.1 Bluetooth

Los orígenes de las comunicaciones Bluetooth se remontan a 1994, cuando Ericsson empezó a investigar diferentes alternativas para comunicar teléfonos móviles con computadoras, e.g., para avanzar y retroceder las diapositivas de una presentación. A principios de 1998, Ericsson se unió a Intel, IBM, Nokia y Toshiba para formar *Bluetooth Special Interest Group* (SIG) cuyo objetivo era desarrollar una tecnología que se convirtiera en un estándar abierto. Posteriormente, otras compañías se añadieron y en julio de 1999 SIG publicó la versión 1.0 de la especificación de Bluetooth.

Al igual que las tecnologías Wi-Fi y de los infrarrojos, Bluetooth es una tecnología de comunicación inalámbrica. A diferencia de la tecnología Wi-Fi, Bluetooth está diseñada para dispositivos que se caracterizan por ofrecer bajo consumo de energía y conexiones de corta distancia (10 y 100 metros). A diferencia de la tecnología de los infrarrojos, Bluetooth es omnidireccional y ofrece un mayor ancho de banda (hasta 11 Mbit/segundo). Por lo tanto, Bluetooth es una tecnología adecuada para la conexión de dispositivos de bajas prestaciones (e.g., teléfonos móviles, cámaras fotográficas, auriculares e impresoras).

Uno de los principales ámbitos de utilización de Bluetooth es sin duda el de los teléfonos móviles. Cada vez es más común encontrar dispositivos móviles con soporte para Java y Bluetooth, por lo tanto resulta evidente la necesidad de programar estos dispositivos mediante Java. Para satisfacer esta necesidad, JCP (*Java Community Process*) desarrolló una especificación JSR (*Java Specification Request*) llamada JSR-82, que será explicada a continuación.

2.1.1 JSR-82

La especificación JSR-82 define un API de alto nivel para crear algunos protocolos y perfiles que se definen en la especificación Bluetooth. El API de JSR-82, que depende de la configuración CLDC (*Connected Limited Device Configuration*) de J2ME (*Java 2 Micro Edition*), facilita la programación de dispositivos Bluetooth. Esta especificación fue producida por un grupo de expertos, los cuales se enumeran a continuación en orden alfabético:

- Extended Systems
- IBM
- Mitsubishi Electric
- Motorola (specification lead)
- Newbury Networks
- Nokia
- Parthus Technologies
- Research in Motion
- Rococo Software
- Sharp Laboratories of America
- Sony Ericsson Mobile Communications
- Smart Fusion
- Smart Network Devices
- Sun Microsystems
- Symbian
- Telecordia
- Vaultus
- Zucotto

El API de JSR-82 está dividido en dos partes: el paquete *Javax.Bluetooth* y el paquete *Javax.Obex*, los cuales son totalmente independientes. El primero de ellos define clases e interfaces básicas para: 1) el descubrimiento de dispositivos y servicios, 2) la conexión y 3) la comunicación de bajo nivel mediante flujos de datos o transmisión de arreglos de octetos. Por el contrario, el paquete *Javax.Obex* permite el manejo del protocolo de alto nivel OBEX (*OBject EXchange*) el cual es similar al protocolo HTTP (*HyperText*

Transfer Protocol) ya que se utiliza principalmente para el intercambio de archivos. El protocolo OBEX, es un estándar desarrollado por la organización IrDA (*Infrared Data Association*), pero también es empleado en otras tecnologías inalámbricas distintas de Bluetooth.

2.1.2 Javax.Bluetooth

En una comunicación Bluetooth existe un dispositivo que ofrece un servicio (aplicación servidor) y otros dispositivos que acceden a él (aplicaciones cliente). Dependiendo de qué parte de la comunicación se debe programar, se realiza una serie de acciones diferentes. Un cliente Bluetooth lleva a cabo los siguientes pasos para establecer una conexión:

- **búsqueda de dispositivos:** la aplicación cliente realiza una búsqueda de los dispositivos Bluetooth a su alcance que estén en modo de conexión;
- **búsqueda de servicios:** la aplicación cliente realiza una búsqueda de servicios por cada dispositivo encontrado;
- **establecimiento de la conexión:** una vez que se ha localizado algún dispositivo que ofrece el servicio deseado, se realiza una conexión con dicho dispositivo;
- **comunicación:** ya establecida la conexión, se puede leer y escribir en ella.

Por otro lado, un servidor Bluetooth sigue los siguientes pasos para establecer una conexión:

- **apertura de la conexión:** se crea una conexión como servidor;
- **inicialización de servicios:** se especifican los atributos del servicio;
- **establecimiento de la conexión:** se establecen las conexiones con las aplicaciones cliente.

Algunas clases del paquete *Javax.Bluetooth* son las siguientes:

- **Clase *LocalDevice***: un objeto de la clase *LocalDevice* representa al dispositivo local. Entre la información que se puede obtener a partir de este objeto se encuentra la dirección Bluetooth del dispositivo y el alias o "*friendly-name*" (también llamado "*Bluetooth device name*" o "*user-friendly name*"). Por medio de este objeto también se puede obtener y establecer el modo de conectividad, i.e., si el dispositivo está o no visible a otros.
- **Clase *DeviceClass***: un objeto de la clase *DeviceClass* retorna un entero por cada servicio disponible en un dispositivo. Este entero contiene tanto el tipo servicio (e.g., impresión, videollamada o transferencia de archivos) como el tipo de dispositivo (e.g., PC, teléfono celular o auriculares). Para obtener el tipo de servicio se requiere emplear el método *getServiceClasses()*, mientras que para obtener el tipo de dispositivo, se necesita utilizar el método *getMajorDeviceClass()*.

- **Clase *UUID***: un objeto de la clase UUID (*Universally Unique IDentifier*) representa un identificador único y universal, formado de enteros de 128 bits, que identifica un protocolo o un servicio. Puesto que un dispositivo puede ofrecer varios servicios, los identificadores UUID permiten designar a cada uno.

- **Clase *DiscoveryAgent***: la búsqueda de dispositivos y servicios Bluetooth se realiza mediante la clase *DiscoveryAgent*. Un objeto de la clase *DiscoveryAgent* se obtiene mediante el método *getDiscoveryAgent()* de un objeto de la clase *LocalDevice*, como a continuación se muestra:

  $$DiscoveryAgent\ discoveryAgent = LocalDevice.getLocalDevice().getDiscoveryAgent();$$

  A partir de la clase *DiscoveryAgent* se tiene la posibilidad de obtener: 1) un arreglo de dispositivos que el usuario ha especificado como "ya conocidos" (PREKNOWN), e.g., una lista de dispositivos "favoritos" y 2) un arreglo de dispositivos descubiertos en búsquedas anteriores (CACHED). La lista de dispositivos "ya conocidos" y "favoritos" se obtiene mediante el método *retrieveDevices()* al pasarle como parámetro *DiscoveryAgent.PREKNOWN* y *DiscoveryAgent.CACHED*, respectivamente.

El paquete *Javax.Bluetooth* permite utilizar dos mecanismos de conexión: 1) SPP (*Serial Port Profile*) y 2) L2CAP (*Logical Link Control and Adaptation Protocol*). Mediante SPP se obtienen los objetos *InputStream* y *OutputStream*, en tanto que a través de L2CAP se envían y reciben arreglos de octetos.

Para abrir cualquier tipo de conexión, se hace uso de la clase *Javax.microedition.io.Connector*. Concretamente, se utiliza el método estático y sobrecargado *open()*, cuya versión más sencilla requiere un parámetro de tipo *String* que contiene una URL con los datos necesarios para realizar la conexión. La URL será diferente dependiendo de si es una aplicación cliente o servidor de tipo L2CAP o SPP.

2.1.3 Javax.Obex

Este paquete es totalmente independiente del paquete *Javax.Bluetooth*. OBEX es un protocolo similar a HTTP, ya que se utiliza pincipalmente para la transferencia de archivos de un servidor a un cliente bajo la petición de este último. De esta manera, el cliente envía comandos (CONNECT, PUT, GET, DELETE, SETPATH, DISCONNECT) al servidor junto con algunas cabeceras de mensaje (ver Tabla 2.1), y en ocasiones, un cuerpo de mensaje (PUT). Las cabeceras de mensaje están encapsuladas en un objeto *HeaderSet*, mientras que el cuerpo del mensaje se lee y escribe mediante los objetos *InputStream* y *OutputStream*, respectivamente.

El servidor recibe los comandos del cliente y responde con un código de respuesta indicando el éxito o fracaso de la petición. Además envía una serie de cabeceras de mensaje con información adicional y un cuerpo de mensaje en caso de tratarse de una respuesta al comando GET.

Algunas clases del paquete *Javax.Obex* son las siguientes:

- **Clase *HeaderSet***: la clase *HeaderSet* representa las cabeceras de un mensaje enviado tanto por un cliente como por un servidor. Las cabeceras de mensaje se guardan como pares clave-valor, donde la clave es un número entero. Estos identificadores numéricos son utilizados en los métodos *setHeader()* y *getHeader()* para establecer y obtener respectivamente una cabecera de mensaje. También se puede obtener un arreglo de todos los identificadores de cabeceras, que guarda un objeto de la clase *HeaderSet* mediante el método *getHeaderList()*.

- **Clase *Operation***: un objeto de la clase *Operation* encapsula las cabeceras y el cuerpo de un mensaje. Las cabeceras de mensaje se guardan en un objeto de la clase *HeaderSet* que se obtiene mediante el método *getRecivedHeaders()*. Para el envío de datos, se utiliza un objeto de la clase *OutputStream* o *DataOutputStream* que se obtiene por medio del método *openOutputStream()* o *openDataOutputStream()*, respectivamente. En el caso de la recepción de datos, se utiliza un objeto de la clase *InputStream* o *DataInputStream* que se adquiere mediante el método *openInputStream()* o *openDataInputStream()*, respectivamente.

Comando	Función
CONNECT	Inicia una sesión
PUT	Envía datos de un cliente a un servidor
GET	Envía datos de un servidor a un cliente
DELETE	Elimina un recurso de un servidor
SETPATH	Crea directorios y navega por ellos
DISCONNECT	Cierra una sesión

Figura 2.1: Tabla de comandos OBEX

2.1.4 Características de Bluetooth

Bluetooth es un protocolo de comunicación diseñado especialmente para dispositivos de poco consumo (en términos de transmisión de datos y de energía), baja cobertura y basados en transceptores económicos.

Clase	Potencia máxima permitida (mW)	Potencia máxima permitida (dBm)	Rango aproximado (metros)
1	100	20	~100
2	2.5	4	~10
3	1	0	~1

Figura 2.2: Tabla de potencias por clases de Bluetooth

Los dispositivos que implementan este protocolo pueden comunicarse entre sí cuando se encuentran dentro de su respectivo alcance. Las comunicaciones se realizan por radiofrecuencia, por lo tanto los dispositivos no tienen que estar alineados e incluso pueden encontrarse en habitaciones separadas si la potencia de transmisión lo permite.

Bluetooth opera en la banda libre de radio ISM (*Industrial, Scientific and Medical*) a 2.4 Ghz. Su máxima velocidad de transmisión de datos es de 1 Mbps. Su rango de alcance depende de la potencia empleada en la transmisión.

Los dispositivos Bluetooth se clasifican en "Clase 1", "Clase 2" y "Clase 3" respecto a su potencia de trasmisión, siendo totalmente compatibles los dispositivos de una clase con los de las otras. La tabla 2.2 muestra la potencia máxima en mW y dBm, asi como el rango de alcance de un dispositivo con respecto a la clase a la que pertenece.

Debido a que la banda ISM está abierta a cualquiera, el sistema de radio Bluetooth deberá estar preparado para evitar las múltiples interferencias que pudieran producirse. Estas pueden ser evitadas mediante un sistema que busque una parte no utilizada del espectro o un sistema de salto de frecuencia.

La técnica de salto de frecuencia es aplicada a una alta velocidad y a una corta longitud de los paquetes (1600 saltos/segundo). La banda de frecuencia es dividida en varios canales de salto, donde los transceptores cambian de un canal de salto a otro de manera pseudo-aleatoria durante la conexión.

Los paquetes de datos están protegidos por el esquema ARQ (*Automatic Repeat reQuest*), el cual retransmite automáticamente los paquetes perdidos.

Bluetooth utiliza un sistema FH/TDD (salto de frecuencia/división de tiempo *duplex*), que divide el canal en intervalos de 625 μs, llamados *slots*.

Dos o más dispositivos Bluetooth pueden compartir el mismo canal dentro de una *piconet* (pequeña red que es establecida automáticamente por los dispositivos Bluetooth para comunicarse entre sí). Un dispositivo Bluetooth maestro controla el tráfico de datos que se genera entre los demás dispositivos (esclavos), mediante el envío y la recepción de señales hacia y desde él mismo.

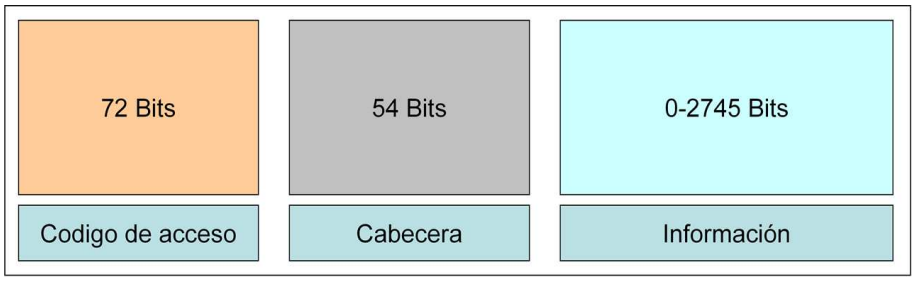

Figura 2.3: Datagrama de Bluetooth

La información que puede ser intercambiada entre dos dispositivos Bluetooth se transmite mediante un conjunto de *slots* que forman un paquete de datos (ver Figura 2.3). Cada paquete comienza con un código de acceso de 72 bits, seguido de un paquete de datos de cabecera de 54 bits, el cual contiene la siguiente información de control: a) tres bits de acceso de dirección, b) tipo de paquete, c) bits de control de flujo, d) bits para la

retransmisión automática de paquetes y e) bits para el control de errores de campos de cabecera. Finalmente, el paquete de información tiene una longitud de 0 a 2745 bits.

2.1.5 Protocolos Bluetooth

La Figura 2.4 muestra un diagrama de bloques de la pila de protocolos de Bluetooth. Varios protocolos han sido definidos en la especificación de Bluetooth, pero la Figura 2.4 muestra solo los más comunes. Las casillas sombreadas representan los protocolos dirigidos por *Java APIs for Bluetooth Wireless Technology* (JABWT). La pila de protocolos se compone de protocolos específicos de la tecnología inalámbrica Bluetooth, tales como *Service Discovery Protocol* (SDP) y *OBject EXchange* (OBEX).

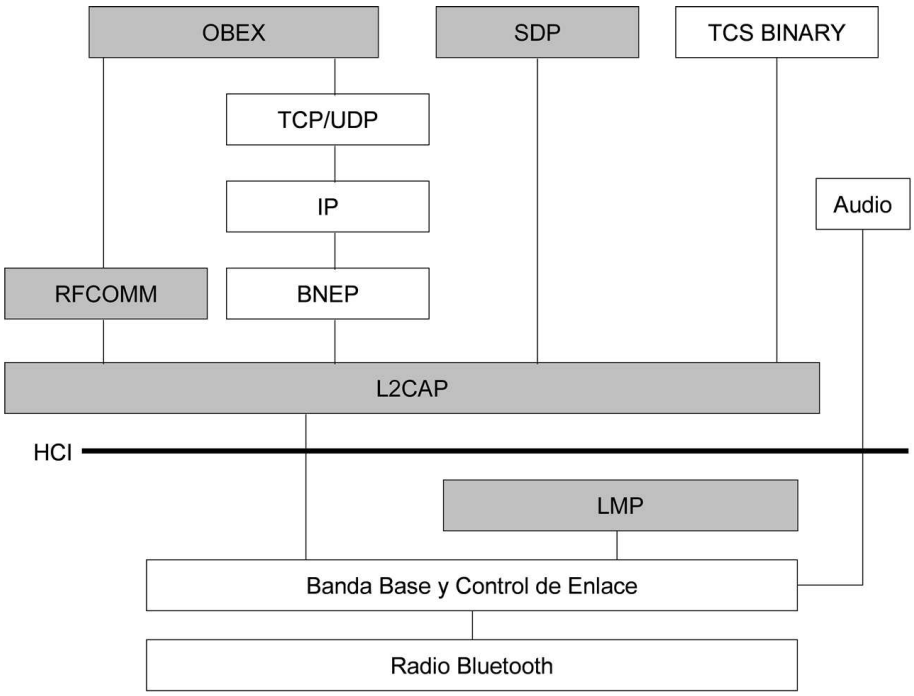

Figura 2.4: Pila de protocolos Bluetooth

A continuación se describe el diagrama de bloques de la pila de protocolos de Bluetooth (ver Figura 2.4):

- La capa de **Radio Bluetooth** está ubicada en la parte más baja de la especificación Bluetooth. Define los requisitos del dispositivo transceptor Bluetooth y opera en los 2.4 GHz de la banda ISM (*Industrial, Scientific and Medical*).

- La capa de **Banda Base** y **Control de Enlace** permite crear el enlace físico por radiofrecuencia entre dispositivos Bluetooth. La Banda Base maneja el canal de procesamiento y tiempo, en tanto que el Control de Enlace se encarga del control de acceso al canal. Existen dos tipos de enlaces físicos: 1) orientado a conexión

síncrona (SCO[1] por sus siglas en inglés) y 2) orientado a conexión asíncrona (ACL[2] por sus siglas en inglés). Un enlace ACL transporta paquetes de datos, mientras que un enlace SCO admite tráfico de audio en tiempo real.

- El componente **Audio** no es realmente una capa de la pila de protocolos Bluetooth, sino que actúa como ejemplo de una transmisión de datos por medio de Bluetooth. Los datos de audio suelen llegar directamente hacia y desde la Banda Base a través de una capa de enlace de SCO. Si se utiliza un canal de datos de audio (e.g., en aplicaciones de VoIP), los datos de audio se transmitirán a través de un enlace ACL.

- El protocolo ***Link Manager Protocol*** (**LMP**) es responsable de la configuración del enlace entre dispositivos Bluetooth y del manejo de los tamaños de paquete de la Banda Base. LMP maneja aspectos de seguridad, como autenticación y cifrado, mediante la generación, el intercambio y la comprobación de vínculos entre claves de cifrado.

- La interfaz ***Host Controller Interface*** (**HCI**) proporciona una interfaz de comando para el Radio Bluetooth, la Banda Base y el Control de Enlace. Se trata de una interfaz estándar para acceder a las capacidades de la Banda Base de Bluetooth, al estado del *hardware* y a los registros de control.

- El protocolo ***Logical Link Control and Adaptation Protocol*** (**L2CAP**) proporciona servicios de datos (orientados o no a conexión) a los protocolos de las capas superiores. También multiplexa las conexiones lógicas creadas por las capas superiores.

- El protocolo ***Service Discovery Protocol*** (**SDP**) encuentra los servicios que ofrecen otros dispositivos.

- El protocolo **RFCOMM** (***Radio Frequency Communication***) emula el puerto serie sobre el protocolo L2CAP, ofreciendo una comunicación segura y con control de errores.

- Los dispositivos Bluetooth tienen la capacidad de formar redes e intercambiar información, por lo que requieren un formato de paquete común que encapsule los protocolos de la capa de red. El protocolo opcional ***Bluetooth Network Encapsulation Protocol*** (**BNEP**) [Chan et al., 2003] realiza esta tarea de encapsulamiento con el fin de permitir que estos protocolos puedan trabajar directamente sobre el protocolo L2CAP.

- La especificación ***Telephony Control Protocol Specification, Binary*** (**TCS Binary**) define la señalización del control de llamadas para el establecimiento de llamadas de voz y datos entre dispositivos Bluetooth. Esta especificación está construida sobre el protocolo L2CAP.

[1] Synchronous Connection Oriented
[2] Asynchronous Connectionless

- Algunos protocolos aprobados, tales como **OBEX** e **IP**, están definidos con base en algunos de los protocolos señalados anteriormente (e.g., OBEX está implantado sobre RFCOMM e IP está construido sobre BNEP).

- El grupo Bluetooth SIG (*Special Interest Group*) también define nuevos protocolos en función de alguno de los protocolos que han sido descritos anteriormente. Estos nuevos protocolos principalmente se construyen sobre el protocolo L2CAP. ***Audio/Video Control Transport Protocol*** y ***Audio/Video Distribution Transport Protocol*** [Thompson et al., 2008] son ejemplos de nuevos protocolos.

2.1.6 Perfiles Bluetooth

Además de protocolos, el grupo Bluetooth SIG define perfiles Bluetooth. Un perfil Bluetooth especifica formas estandarizadas de utilizar protocolos para ciertos modelos. Dicho de otra forma, un perfil Bluetooth define cómo pueden ser utilizadas las diferentes partes del estándar Bluetooth para realizar una tarea específica.

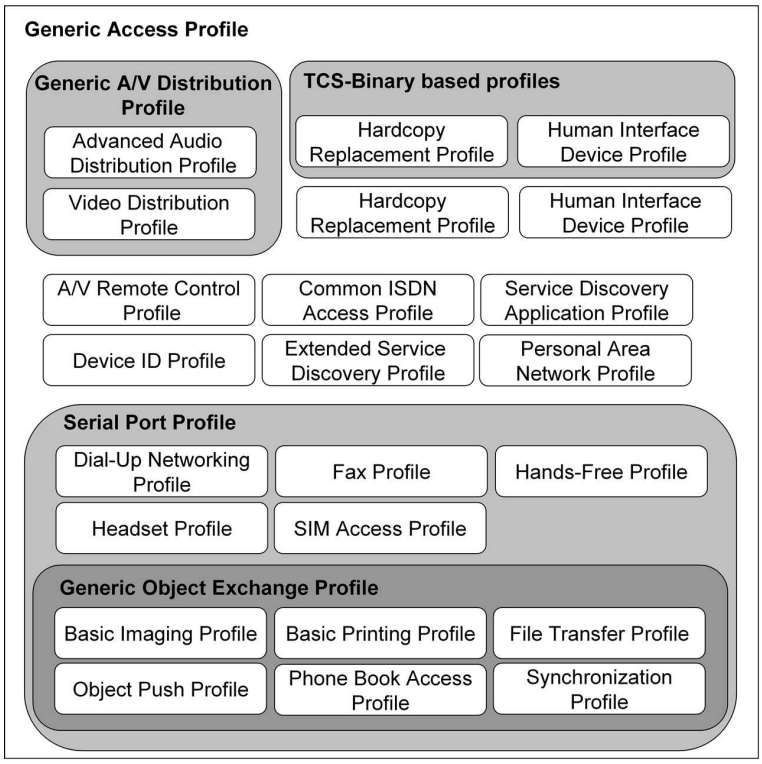

Figura 2.5: Perfiles de Bluetooth

Un dispositivo Bluetooth puede soportar uno o más perfiles. Los cuatro perfiles básicos son: Generic Access Profile (GAP), Serial Port Profile (SPP), Service Discovery Application Profile (SDAP) y Generic Object Exchange Profile (GOEP). A continuación se detalla cada uno de ellos:

- **Generic Access Profile (GAP)**: es la base de todos los perfiles. GAP define los procedimientos básicos para establecer conexiones entre dos dispositivos, incluyendo el descubrimiento de dispositivos Bluetooth, sus configuraciones y procedimientos de seguridad.
- **Serial Port Profile (SPP)**: define los requerimientos necesarios para que los dispositivos Bluetooth configuren las conexiones emuladas del puerto serie, utilizando RFCOMM.
- **Service Discovery Application Profile (SDAP)**: describe las operaciones necesarias para el descubrimiento de servicios. Particularmente, este perfil define los protocolos y procedimientos requeridos por las aplicaciones para localizar servicios en otros dispositivos Bluetooth.
- **Generic Object Exchange Profile (GOEP)**: define todos los elementos necesarios para el soporte de OBEX (e.g., transferencia de archivos, sincronización y carga de objetos).

La Figura 2.5 muestra las relaciones entre los diversos perfiles de Bluetooth, los cuales están organizados de manera jerárquica. Por ejemplo, el perfil *File Transfer Profile* está construido en la parte superior del perfil GOEP, que depende del perfil SPP, el cual está construido sobre el perfil GAP. Los perfiles básicos (GAP, SDAP, SPP, y GOEP) también son conocidos como perfiles de transporte, sobre los cuales se puede construir otros perfiles, denominados perfiles de aplicación.

2.2 Redes *ad hoc* móviles

Una red *ad hoc* [Chlamtac et al., 2003] es una colección de nodos inalámbricos (i.e., dispositivos de cómputo), los cuales cooperan para formar una red que opera sin la ayuda de infraestructura de comunicaciones física (ver Figura 2.6). Los nodos inalámbricos deben ser capaces de descubrir y compartir servicios de forma dinámica [Campo et al., 2006a].

Históricamente, las redes *ad hoc* móviles han sido utilizadas principalmente en aplicaciones relacionadas con redes tácticas para mejorar la comunicación y la supervivencia en los campos de batalla. La naturaleza dinámica de las operaciones militares provocaba que los militares no confiaran en el acceso a una infraestructura de comunicación fija en los campos de batalla. Sin embargo, la comunicación inalámbrica pura también tiene la limitación de que las señales de radio están sujetas a interferencia, además de que la radiofrecuencia, que es mayor que 100 MHz, rara vez se propaga más allá de la línea de visión [Freebersyser and Leiner, 2001].

Una red *ad hoc* móvil crea un marco conveniente para abordar estas cuestiones, ya que proporciona una red inalámbrica *multi-hop* sin infraestructura pre-establecida y conectividad más allá de la línea de visión. Los nodos de una red *ad hoc* son libres de moverse de forma aleatoria y se organizan arbitrariamente [Zhu and Chlamtac, 2006]. Por lo tanto, la topología de la red inalámbrica puede cambiar rápidamente y de forma impredecible.

En general, las rutas entre los nodos de una red *ad hoc* pueden incluir múltiples saltos. Por lo tanto, es conveniente llamar "redes inalámbricas *ad hoc multi-hop*" a este tipo

de redes. Cada nodo es capaz de comunicarse directamente con cualquier otro nodo que se encuentre dentro de su rango de transmisión. Para comunicarse con los nodos que están ubicados más allá de este rango, un nodo debe utilizar los nodos intermedios para transmitir los mensajes salto por salto.

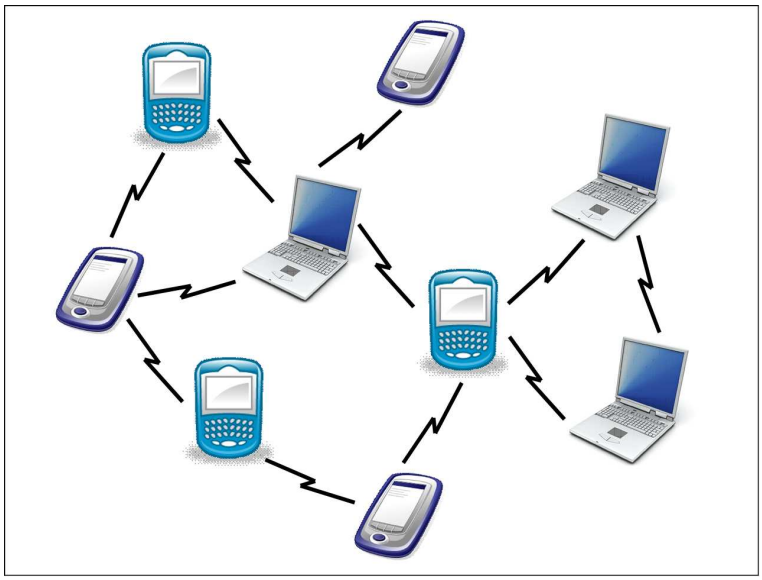

Figura 2.6: Red *ad hoc*

2.2.1 Cuestiones de las redes *ad hoc*

La flexibilidad y la comodidad de las redes *ad hoc* pagan un precio [Kesselman et al., 2005], ya que heredan los problemas tradicionales de la comunicacion inalámbrica:

- el canal está desprotegido de señales externas;
- los medios inalámbricos son mucho menos fiables que el medio de comunicación cableado;
- el canal tiene características de tiempo variable y de propagación asimétrica;
- es susceptible de fenómenos de terminal oculta o expuesta.

A estos problemas y complejidades, se agregan otras características y facilidades de diseño que son específicas al crear una red *ad hoc* [Ji and Corson, 2003]:

- **Autonomía y sin infraestructura fija**: las redes *ad hoc* no dependen de ninguna infraestructura o administración centralizada; cada nodo opera en modo *peer to peer*, actúa como un *router* autónomo y genera datos independientes;
- **Enrutamiento *multi-hop***: ningún *router* está habilitado por omisión; cada nodo actúa como un *router* que envía los paquetes de los demás para que puedan compartir información entre ellos;

- **Topologías de red cambiantes de forma dinámica**: los nodos pueden moverse arbitrariamente; la topología de la red, que suele ser *multi-hop*, puede cambiar con frecuencia y de forma impredecible, resultando en cambios de ruta, particiones frecuentes de red y posibles pérdidas de paquetes;

- **Variación de la capacidad del enlace y de los nodos**: cada nodo puede estar equipado con una o más interfaces de radio que tienen diversas capacidades de transmisión/recepción y operan a través de diferentes bandas de frecuencia;

- **Energía de operación limitada**: dado que las baterías de cada nodo móvil tienen energía limitada, el poder de procesamiento también es limitado, lo que a su vez restringe las aplicaciones y los servicios que pueden ser soportados por cada nodo; el problema de la energía de operación limitada se vuelve mayor en redes móviles *ad hoc* ya que, como cada nodo actúa simultáneamente como sistema final y *router*, se requiere energía adicional para reenviar paquetes a otros nodos.

Actualmente, muchos dispositivos móviles cuentan con la tecnología Bluetooth, e.g., teléfonos celulares, PDAs y computadoras portátiles. En una red Bluetooth, existen dos tipos de nodos: un esclavo y un maestro. Cada nodo tiene la capacidad de jugar uno o ambos roles al mismo tiempo. Una red Bluetooth en realidad se compone de pequeñas subredes o *piconets*. A su vez, una *piconet* está conformada de dos o más nodos conectados que comparten el mismo canal. Cada *piconet* tiene un maestro y hasta 7 esclavos, pero nunca se lleva a cabo una transmisión directa entre esclavos.

2.2.2 Piconets

Cuando un dispositivo Bluetooth se encuentra dentro del radio de cobertura de otro, estos pueden establecer una conexión entre ellos. Cada dispositivo tiene una dirección única de 48 bits, basada en el estándar IEEE 802.11 para WLAN. Solo se requiere un par de dispositivos con las mismas características de *hardware* para establecer una conexión. De esta manera, dos o más dispositivos Bluetooth que comparten un mismo canal forman una *piconet* (ver Figura 2.7).

Para regular el tráfico en el canal, uno de los dispositivos se convierte en maestro. Por definición, el dispositivo que establece la *piconet* asume este papel, mientras que todos los demás juegan el rol de esclavo.

Los nodos Bluetooth pasan por varios estados cuando se establece una conexión. Los dos estados principales de un nodo son STANDBY - cuando el dispositivo no es parte de una *piconet* y CONNECTION - cuando el dispositivo si es parte de una *piconet*. Para formar una *piconet*, el nodo maestro transmite un paquete de identificación sobre 32 de los 79 canales. Los dispositivos en el estado STANDBY escuchan periódicamente este paquete para poder enviar su dirección al maestro. De manera recíproca, cada dispositivo se pone en escucha del maestro para poderlo registrar. Cuando el maestro ha identificado a todos los dispositivos que se encuentran a su alcance, comienza a formar la *piconet*. El maestro registra el DAC (*Device Access Code*) de cada dispositivo, utilizando una secuencia de salto de frecuencia que está basada en la dirección de los esclavos. Cuando un esclavo escucha su DAC envía un paquete de confirmación. Posteriormente, el maestro

envía al esclavo el DAC maestro. El esclavo cambia entonces al estado CONNECTION. El maestro envía el DAC maestro a todos los esclavos en la *piconet* y finalmente cambia al estado CONNECTION.

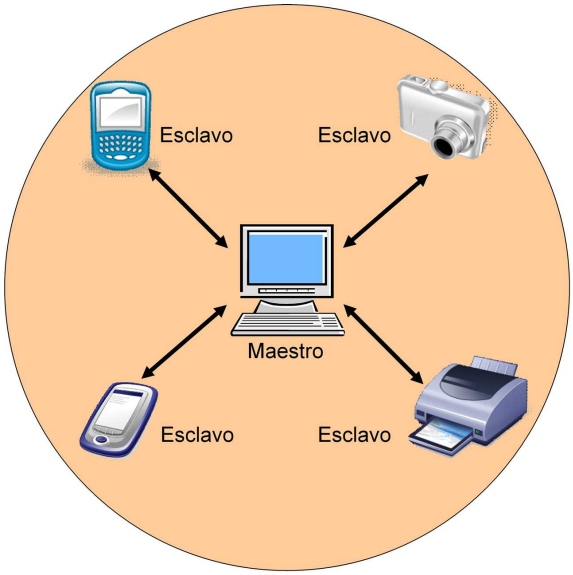

Figura 2.7: Ejemplo de una *Piconet*

2.3 Sistemas de descubrimiento de servicios

Debido al dinamismo de los entornos de cómputo ubicuo y a la creciente cantidad y diversidad de dispositivos móviles, es necesario proporcionar un mecanismo de descubrimiento y anuncio de servicios. Algunas soluciones que ofrecen dicho mecanismo están ligadas a un protocolo de red (e.g., IP), mientras que otras forman parte de la arquitectura de un *framework* de desarrollo de servicios en entornos dinámicos. Por lo tanto, la utilización de estas soluciones está muy ligada a la forma en que se desarrollan y se proporcionan estos servicios [Campo et al., 2006b].

En esta sección, se realiza un análisis de los más importantes sistemas de descubrimiento de servicios para poner en evidencia sus principales características.

2.3.1 Service Location Protocol

El objetivo del sistema *Service Location Protocol* (SLP) propuesto por *Internet Engineering Task Force* (IETF) [Zhao and Schulzrinne, 2005] es definir un protocolo de descubrimiento de servicios en redes IP, descentralizado y extensible. Para la descripción de servicios, este sistema utiliza URLs, las cuales permiten a las aplicaciones conocer qué servicios existen en la red y acceder a ellos.

La infraestructura de SLP consiste en tres tipos de agentes: agentes de usuario, agentes de servicio y agentes de directorio. Los agentes de usuario están encargados de realizar el

descubrimiento de servicios para satisfacer las necesidades que demandan las aplicaciones de los usuarios finales. Los agentes de servicio son responsables de anunciar las características y la localización de los servicios. Los agentes de directorio están encargados de almacenar información sobre los servicios que se anuncian en la red.

El sistema SLP tiene dos modos de funcionamiento: basado y no basado en agentes de directorio. Cuando SLP está basado en agentes de directorio, los agentes de servicio registran los servicios que brindan ante los agentes de directorio, en tanto que los agentes de usuario buscan los servicios que necesitan. Cuando SLP no está basado en agentes de directorio, los agentes de usuario envían por medio de mensajes *multicast* peticiones de servicios, a las cuales responden los agentes que ofrecen dichos servicios mediante mensajes *unicast*.

SLP define dos mecanismos para descubrir agentes de directorio: modo pasivo y modo activo. De acuerdo con el mecanismo de **modo pasivo**, los agentes de servicio y de usuario escuchan los mensajes *multicast* en los que los agentes de directorio anuncian su existencia de forma periódica. Según el mecanismo de **modo activo**, los agentes de servicio y de usuario envían un mensaje *multicast* o utilizan DHCP para descubrir los agentes de directorio existentes. Si existe alguno, los agentes de servicio y de usuario emplean comunicación *unicast* con dicho agente de directorio para registrar y buscar servicios, respectivamente.

El sistema SLP proporciona un mecanismo de búsqueda de servicios, pero en ningún momento señala cómo los clientes hacen uso de ellos. Asimismo, introduce algunos mecanismos de seguridad, sobre todo para garantizar que no exista información falsa referente a la localización de los servicios.

2.3.2 Ninja: Secure Service Discovery Service

El sistema *Secure Service Discovery Service* (SSDS) [Herborn et al., 2005] es parte de un proyecto de investigación de la Universidad de California en Berkeley. SSDS es similar a otros sistemas de descubrimiento, pero ofrece una serie de mejoras en términos de fiabilidad, escalabilidad y seguridad. Aunque SSDS se basa en Java, utiliza archivos XML en lugar de objetos Java para la descripción y la ubicación de un servicio.

El modelo de SSDS consta de clientes, servicios y servidores seguros de descubrimiento de servicios (SDS). Un servicio SDS disponible es anunciado por un servidor SDS mediante mensajes *multicast* periódicos (autenticados), los cuales contienen la URL del servicio SDS disponible. Tanto el servidor SDS como los clientes pueden almacenar en caché la información del servicio; además el estado del sistema puede ser construido íntegramente a partir de mensajes *multicast*. Este esquema proporciona escalabilidad y facilita la recuperación de errores con una mínima intervención manual. Estas caracteristicas hacen que el sistema SSDS sea muy robusto.

SSDS está implementado mediante métodos remotos de Java RMI. El servidor SDS consta de un directorio de búsqueda de servicios, un administrador de capacidades y una autoridad de certificación. La ubicación de servicios se realiza por medio de la comparación de etiquetas XML.

Los servidores SDS se organizan en una jerarquía, a la que otros servidores pueden añadirse dinámicamente. La jerarquía de servidores detecta si un servidor presenta al-

guna falla para reiniciarlo o reemplazarlo.

2.3.3 Jini

Jini [Oaks and Wong, 2000] es una arquitectura distribuida orientada a servicios que fue propuesta por Sun Microsystems. El objetivo de Jini es convertir una red en una herramienta flexible y de fácil administración en la que los clientes puedan encontrar servicios de un modo eficiente. Jini se apoya en Java, por lo que es necesario que todos los dispositivos participantes dispongan de una Máquina Virtual Java o que exista algún otro dispositivo que si la tenga y actúe en su nombre.

Jini define principalmente tres componentes: servicios, clientes y servicios de directorio, también llamados *Jini Lookup Services* (JLS). Un servicio está representado por un objeto Java que proporciona algún tipo de cálculo o control en un dispositivo. Un cliente es aquel que hace uso de un servicio, por lo tanto un servicio puede ser a su vez cliente de otro.

JLS debe existir dentro de la arquitectura de Jini, por lo tanto los clientes y los servicios siempre se descubren a través de él, nunca de forma directa. Para poder registrar o buscar un servicio primero es necesario localizar algún JLS, mediante alguno de los tres protocolos siguientes: *Unicast Discovery Protocol*, *Multicast Request Protocol* y *Multicast Announcement Protocol*. El protocolo *Unicast Discovery Protocol* se emplea cuando ya se conoce un JLS. Por su parte, el protocolo *Multicast Request Protocol* se utiliza para buscar un JLS, en tanto que el protocolo *Multicast Announcement Protocol* es empleado por los JLS para anunciar su disponibilidad. Jini implementa un mecanismo de alquiler que obliga a que los servicios actualicen su registro periódicamente para mantener su entrada en un JLS. De esta manera, los servicios pueden ser localizados por los clientes.

Un usuario que busca un servicio en una red primero envía una consulta (*multicast*) para encontrar un JLS. Si este existe, el objeto remoto correspondiente se descarga en el dispositivo del usuario, quien entonces utiliza el objeto remoto para encontrar el servicio deseado. El descubrimiento de servicios se lleva a cabo por coincidencia (*matching*) de interfaz de comunicación o atributos Java. Si un JLS contiene un servicio válido que implementa la interfaz especificada por el usuario, entonces se descargará un *proxy* de ese servicio en el dispositivo del usuario. A partir de ese momento, el *proxy* se utiliza para llamar a las diferentes funciones ofrecidas por el servicio.

2.3.4 Universal Plug and Play

La arquitectura *Universal Plug and Play* (UPnP) [Jeronimo and Weast, 2003], impulsada por Microsoft, extiende el modelo original de periféricos *Plug and Play* de Microsoft a un entorno dinámico, donde pueden interactuar múltiples dispositivos de red ofrecidos por diversos proveedores. UPnP trabaja básicamente mediante conjuntos de protocolos de red (TCP/IP) de las capas más bajas del modelo de Internet e implementa estándares a este nivel. Al brindar un conjunto de protocolos de red definidos, UPnP permite a los dispositivos definir sus propias APIs (*Application Programming Interfaces*) para implementar estos protocolos en cualquier lenguaje o plataforma que dichos dispositivos soporten.

UPnP utiliza el protocolo *Simple Service Discovery Protocol* (SSDP) para descubrir

servicios en redes basadas en IP. SSDP puede operar en la red con o sin un servicio de directorio o búsqueda. SSDP trabaja sobre los protocolos abiertos existentes mediante envío único (*unicast*) y envío múltiple (*multicast*) sobre el protocolo HTTP (*HyperText Transfer Protocol*).

Cuando un servicio quiere anunciarse en la red, primero envía un mensaje de aviso (o anuncio) para notificar su presencia. En el caso de anuncio *multicast*, el servicio envía el aviso a una dirección reservada. Si está presente un servicio de directorio o búsqueda, este puede registrar tales avisos. El mensaje de aviso contiene dos URLs (*Universal Resource Locator*), una que identifica el servicio publicado y otra que brinda una descripción de dicho servicio.

Cuando un cliente quiere descubrir un servicio, puede contactarlo directamente a través de la URL que se provee en el anuncio del servicio o bien puede enviar una solicitud de búsqueda *multicast*. Si un cliente descubre un servicio a través de esta solicitud de búsqueda, el pedido del cliente puede ser atendido directamente por el servicio o por un servicio de directorio o búsqueda.

2.3.5 Bonjour

La tecnología Bonjour, impulsada por Apple, se basa en el trabajo iniciado en 1999 por el grupo IETF Zeroconf. El objetivo de esta tecnología es permitir la creación de redes *ad hoc* sin infraestructura fija [White, 2007]. Bonjour utiliza interfaces estándares DNS (*Domain Name System*), servidores y formatos de paquetes para buscar servicios en la red. Bonjour brinda soporte al descubrimiento de servicios mediante registros de recursos DNS ya existentes. Además, Bonjour realiza consultas que permiten al usuario obtener una lista de las instancias de un tipo específico de servicios.

Bonjour es un sistema basado en una estructura jerárquica de servicios y en interfaces DNS para la resolución de nombres en redes completamente distribuidas, como Multicast DNS o LLMNR (*Link Local Multicast Name Resolution*). Dado el tipo de servicio y el dominio que el cliente busca, Bonjour le permite descubrir una lista de instancias de nombres del servicio deseado que utilizan únicamente consultas de tipo DNS.

2.3.6 DEAPSpace

El sistema DEAPSpace [Nidd, 2001] fue desarrollado por IBM Research para operar de forma eficiente en redes inalámbricas *ad hoc* de un solo salto. Por medio del sistema DEAPSpace, un dispositivo puede: 1) detectar la presencia de dispositivos próximos, 2) compartir información de los servicios disponibles y 3) detectar la indisponibilidad de otros dispositivos. El objetivo principal de este sistema de descubrimiento de servicios es dar respuesta a los cambios frecuentes que se producen en el entorno, teniendo en cuenta las limitaciones de potencia de los dispositivos.

El sistema DEAPSpace se basa en un método *push* puro [Kim et al., 2001], en el que todos los dispositivos mantienen una vista global que transmiten a todos sus vecinos cada cierto periodo de tiempo. La vista general se actualiza cuando DEAPSpace recibe la vista global de cada uno de los dispositivos. La principal contribución de DEAPSpace es la definición del formato de descripción de servicios y del mecanismo de codificación, el

cual minimiza la cantidad de datos a transmitir durante el proceso de descubrimiento de servicios.

2.3.7 Salutation

Salutation [Jamalipour, 2003] es un protocolo de descubrimiento de servicios y de administración de sesiones que fue desarrollado por *Salutation Consortium*. Salutation es un estándar abierto e independiente de sistemas operativos, protocolos de comunicación y plataformas de *hardware*. Salutation resuelve los problemas de descubrimiento y acceso de servicios ofrecidos por un amplio conjunto de dispositivos, que interactúan en un entorno con gran cobertura de conectividad y movilidad.

La arquitectura de Salutation define tres entidades:

- *Functional Units* que desde el punto de vista de un cliente definen un servicio. Para algunos de los servicios más habituales, e.g., impresoras, faxes o almacenamiento de documentos, el consorcio Salutation define 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 por la red, i.e., funciona como un intermediario entre los servicios de la red y los clientes.
- *Transport Managers* (TMs) que aisla al SLM del protocolo de transporte que se emplea 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.

La entidad SLM puede ser descubierta por los servicios mediante:

1. una tabla estática que guarda la dirección de transporte del SLM remoto;
2. una consulta de descubrimiento, utilizando el protocolo SLP;
3. la dirección de transporte del SLM remoto de un servidor de directorio central, el cual está indefinido por la arquitectura de Salutation; sin embargo, la especificación actual sugiere el uso del protocolo SLP;
4. una especificación directa de la dirección de transporte del SLM remoto.

El proceso de descubrimiento de servicios puede realizarse mediante múltiples SLMs. Un SLM puede descubrir otros SLMs remotos para determinar los servicios que están registrados allí. El descubrimiento de servicios se lleva a cabo comparando el tipo de servicio requerido, según lo especifica el SLM local, con el tipo de servicio disponible en un SLM remoto. Se utilizan llamadas a procedimientos remotos para transmitir: 1) el tipo de servicio requerido desde el SLM local hacia el SLM remoto y 2) la respuesta desde el SLM remoto hacia el SLM local. SLM determina las características de todos los servicios registrados en un SLM remoto, al manipular la especificación del tipo de servicio requerido. También puede determinar las características de un servicio específico que ha sido registrado en un SLM remoto o la presencia de un servicio específico en un SLM remoto, al comparar un conjunto específico de características.

2.3.8 Análisis comparativo de los sistemas estudiados

En la tabla 2.8 se presenta una descripción de los sistemas de descubrimiento de servicio analizados anteriormente. Se incluyen las siguientes características: a) el nombre del sistema de descubrimiento de servicios; b) si depende o no de un lenguaje; c) si está basado o no en un directorio; d) el tipo de almacenamiento de la información sobre los servicios disponibles; e) la técnica de descubrimiento de servicios utilizada por el sistema; f) la topología empleada para manejar la información de los servicios y de las consultas de los clientes; y g) el tipo de red.

Sistema	Dependiente de lenguaje	Basado en directorio	Almacenamiento de la información de los servicios	Técnica de descubrimiento de servicios	Topología	Tipo de red
SLP	×	✓	*Dictionary Agent* (DA)	Repositorio centralizado de servicios	- Centralizado - P2P	Subred
Ninja SSDS	×	✓	*Lookup Service*	Descripciones XML	Cliente-Servidor	Estructura jerárquica de servidores
Jini	✓	✓	*Lookup Service*	Repositorio centralizado de servicios	Centralizado	Subred
UPnP	×	×	Punto de control	Inundación de peticiones	P2P	Subred
Bonjour	×	✓	Directorio jerárquico	Repositorio centralizado de servicios	- Centralizado - P2P	Intranet
DEAPSpace	×	×	Vista global	Inundación de peticiones	P2P	*Mobile ad hoc 1-Hop*
Salutation	×	✓	*Salutation Manager*	Repositorio centralizado de servicios	- Cliente-Servidor - P2P	Cualquier red depende del transporte

Figura 2.8: Tabla comparativa de los principales sistemas de descubrimiento de servicios

Todos los sistemas descritos anteriormente se basan en la comparación de atributos típicos (e.g., el tipo de servicio especificado por una URL) o de atributos de la interfaz de comunicación (e.g., la dirección IP y el número de puerto del *host* de servicio) para comprobar la existencia y la disponibilidad de los servicios requeridos por un usuario.

A excepción de Jini, todos los demás sistemas analizados son independientes del lenguaje. Dado que Jini es totalmente dependiente de Java, es necesario que exista una Máquina Virtual de Java en los dispositivos implicados.

Los sistemas de descubrimiento están o no basados en directorio para almacenar la información de los servicios disponibles en la red. Particularmente SLP, Ninja SSDS, Jini, Bonjour y Salutation están basados en un directorio, el cual recibe distintos nombres dependiendo del sistema, e.g., en SLP se le denomina *Dictionary Agent* (DA). Para acceder a un servicio a través de un directorio, el cliente primero contacta el directorio central para obtener la descripción del servicio y posteriormente interactuar con el proveedor de servicios. Por el contrario, los sistemas UPnP y DEAPSpace no contienen un directorio

para almacenar información de los servicios disponibles en la red. Los proveedores no distribuyen la descripción de sus servicios al resto de los nodos de la red sino que la mantienen en el dispositivo mediante el empleo de un caché local.

La tecnica de descubrimiento de servicios utilizada por los sistemas basados en directorio emplea un repositorio centralizado de servicios, a excepción de Ninja SSDS que hace uso de descripciones XML. Por el contrario, los sistemas no basados en directorio envían peticiones (i.e., inundación de peticiones) a todos los nodos de la red, con el fin de encontrar la descripción del servicio solicitado.

La topología se refiere a cómo los sistemas manejan la información de los servicios y las consultas de los clientes. La topología puede ser de tipo P2P, cliente-servidor o centralizada. UPnP y Jini emplean respectivamente topologías de tipo P2P y centralizada dentro de una subred. Por el contrario, Ninja SSDS maneja una topología de tipo cliente-servidor con base en una estructura jerárquica de servidores. Dependiendo del transporte, Salutation puede trabajar mediante una topología tipo cliente-servidor o P2P dentro de cualquier tipo de red.

Todos estos sistemas de descubrimiento de servicios solo proporcionan apoyo en términos de la localización de servicios (e.g., Jini, UPnP y Bonjour) o de la topología de red (e.g., DEAPSpace y Salutation). Cada uno resuelve un problema de tipo diferente, e.g., el control de acceso al servicio, el reconocimiento del dispositivo, la conexión a la red o la identificación del servicio. Sin embargo, la mayoría de ellos se centra solamente en entornos domésticos (e.g., la localización de servicios multimedia para permitir a un usuario recibir una videollamada) o en entornos empresariales (e.g., la determinación de los servicios disponibles para tratar una petición de impresión). Por lo tanto, estos sistemas de descubrimiento no pueden ser utilizados en otros contextos como por ejemplo, en entornos colaborativos dinámicos.

Todos los sistemas analizados exploran distintos aspectos del descubrimiento de servicios en sistemas distribuidos. Empero, no ofrecen una solución adaptable a los problemas derivados de los entornos colaborativos dinámicos, e.g., la detección automática de los servicios prestados por las diferentes áreas de una organización o la creación dinámica de redes *ad hoc* para dar soporte tanto a la interacción usuario-servicio como a la interacción usuario-usuario.

Capítulo 3

Análisis y diseño del sistema SEDINU

El objetivo del presente capítulo es describir el análisis y el diseño del sistema SEDINU (*SErvice DIscovery for Nomadic Users*). Este sistema tiene el objetivo de proporcionar un soporte a los usuarios nómadas para interactuar con los servicios disponibles y con otros usuarios nómadas dentro de áreas autónomas. Primero, se describe en forma detallada el concepto de áreas autónomas (sección 3.1), asi como un escenario que pone en evidencia los potenciales beneficios de la implantación del sistema SEDINU dentro de una organización (sección 3.2). Después se presenta las arquitecturas funcional (sección 3.3) y de distribución (sección 3.4) de este sistema y finalmente se describe los módulos que lo conforman (sección 3.5).

3.1 Áreas autónomas

Hoy en día, las grandes organizaciones se dividen en sub-organizaciones (e.g., servicios administrativos, departamentos y secciones) con el fin de ser más eficientes en proporcionar a los usuarios servicios especializados. Las sub-organizaciones se refieren a áreas autónomas que apoyan a la organización global en la administración de sus recursos (e.g., servicios, información contextual, flujos de trabajo y roles), los cuales son manejados y controlados en una forma distribuida.

Así, cada área autónoma se convierte en una unidad de administración que maneja sus propios recursos, respetando las políticas de la organización (e.g., seguridad y facturación). Las diferentes áreas intercambian información para coordinarse entre ellas, con el fin de ayudar a los usuarios nómadas a alcanzar sus objetivos parciales y globales dentro de la organización. Desde el punto de vista físico, un área autónoma puede estar localizada en varios edificios, en un edificio o en una parte de un edificio.

Las áreas autónomas constituyen una jerarquía de sub-organizaciones donde los niveles más altos proporcionan roles y actividades generales, mientras que los niveles más bajos ofrecen roles y actividades específicas. La Figura 3.1 ilustra el principio de atribución de un rol a un usuario nómada dentro de una jerarquía simple, donde el área A0 es la organización global y el área A1 es una sub-organización. Cada área define un conjunto

de roles, e.g., el área A0 define los roles R01 y R02, mientras que el área A1 define el rol R11. Así, cuando el usuario nómada entra al área A0 por primera vez, se le atribuye el rol R01, e.g., en la recepción del CINVESTAV, el rol `visitante de Computación` es asignado a un computólogo, cuyo objetivo es dar una conferencia en el `Departamento de Computación`. Posteriormente, cuando el usuario nómada entra al área A1, su rol cambia de R01 a R11. De esta manera, él puede llevar a cabo algunas actividades dentro del área A1. Por último, cuando el usuario nómada termina la Tarea T11, pasa del área A1 al área A0 donde se le atribuye el rol R02, i.e., un usuario puede obtener un rol diferente al que previamente obtuvo cuando entró al área A0 por primera vez.

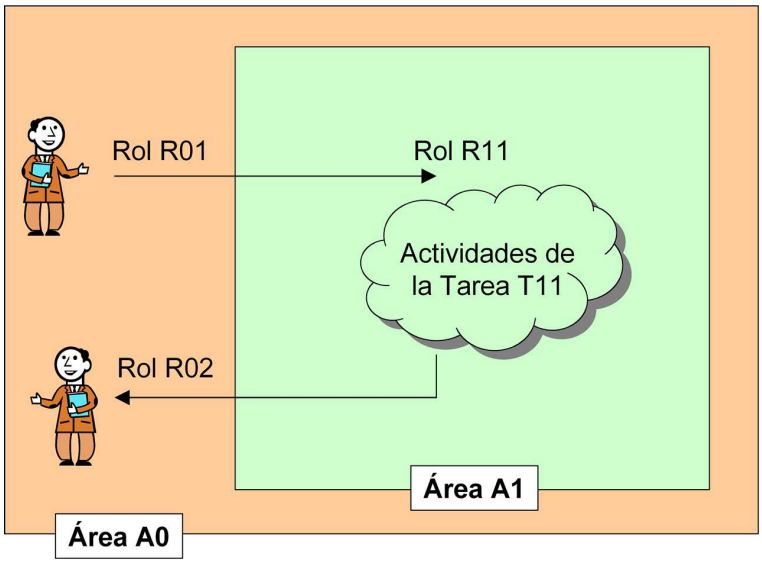

Figura 3.1: Principio de atribución de un rol a un usuario en función de su ubicación

Jerarquía de áreas

Las grandes organizaciones suelen dividirse en sub-organizaciones que se estructuran siguiendo una relación jerárquica bien definida. Nuestra propuesta consiste en definir una jerarquía de áreas que más o menos siga la estructura de una organización: cada sub-organización puede estar asociada a una única área autónoma o a varias áreas interconectadas jerárquicamente, e.g., el `Departamento de Física` puede actuar como un área principal que incluye la sub-área `Laboratorio de Investigaciones Nucleares`. Esta última área define usos específicos y políticas administrativas (e.g., procedimientos de entrada y salida de humanos y productos).

Por otra parte, las grandes organizaciones requieren un sofisticado sistema de flujo de trabajo [Han et al., 2006] que administre sus recursos y tareas. Sin embargo, la definición y gestión de un sistema de flujo de trabajo global constituye un complejo y casi irrealizable objetivo, debido a la especificidad de las políticas administrativas de cada sub-organización. Entre otras consideraciones, resulta interesante tomar los beneficios de una estructura basada en áreas para definir y administrar, de manera eficiente, el sistema de flujo de trabajo de una organización. De esta manera, una gran organización contaría con

un flujo de trabajo que está lógicamente distribuido entre sus sub-áreas. Las áreas que engloban a otras definen flujos de trabajo generales, en tanto que las áreas englobadas especifican flujos de trabajo que implementan políticas administrativas y tareas particulares.

Por lo tanto, se propone una organización jerárquica en la que cada área autónoma está encargada de definir y administrar su propio componente del flujo de trabajo global. De esta manera, cada sub-flujo de trabajo asociado a un área define sus puntos de conexión, i.e., de entrada y salida. A guisa de ilustración, la Figura 3.2 muestra los flujos de trabajo de las áreas A1, A3 y A.3.1 (ver parte central) y la organización jerárquica de dichas áreas (ver parte izquierda). Estas tres áreas están vinculadas por una relación "englobando-englobado". De acuerdo a esta relación, el sub-flujo de trabajo de un área constituye una parte independiente del flujo de trabajo global que está conectado al área que engloba. De esta manera, los puntos de conexión C1 y C2 del área A3 están asociados respectivamente a los puntos de inserción I1 e I2 del área A1.

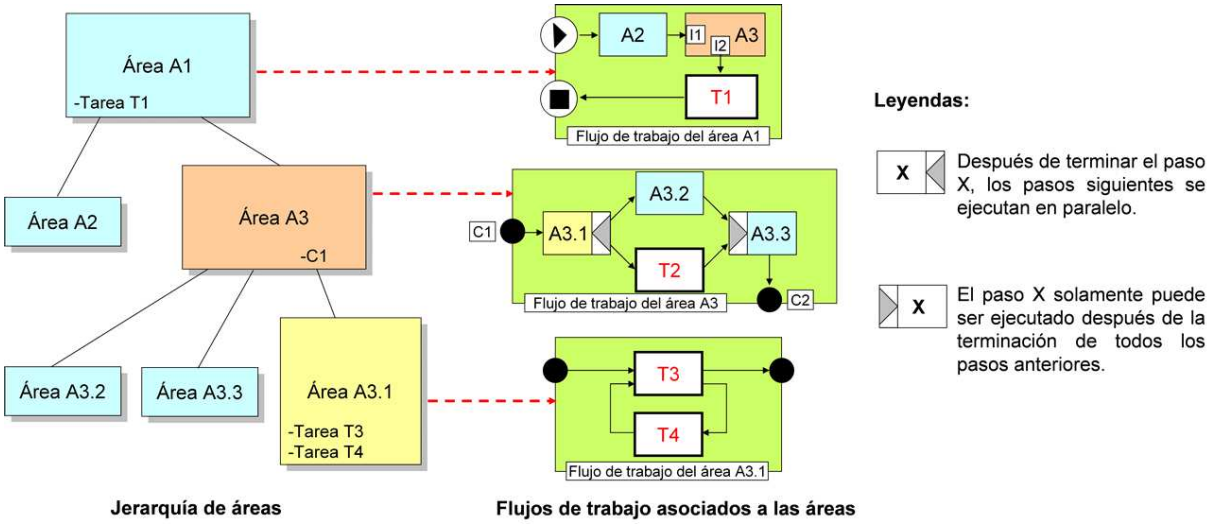

Figura 3.2: Principio de organización jerárquica de áreas autónomas

Cada sub-flujo de trabajo permite definir y gestionar de manera eficiente el flujo de trabajo global. De esta manera, diferentes sub-flujos de trabajo son manejados de una manera independiente en las correspondientes áreas autónomas. Más precisamente, en la Figura 3.2 se presenta una jerarquía de áreas autónomas, donde el área A1 incluye dos sub-áreas, A2 y A3; esta última a su vez engloba tres sub-áreas, A3.1, A3.2 y A3.3. Sin embargo, cada área maneja su propio entorno, i.e., información contextual, servicios, roles y un componente del flujo de trabajo global.

El flujo de trabajo del área A1 define un flujo de trabajo secuencial que sucesivamente ejecuta los flujos de trabajo de las áreas A2 y A3 y termina con la ejecución de la tarea T1 dentro del área A1. Por su parte, el sub-flujo de trabajo del área A3 consiste en la ejecución del sub-flujo de trabajo del área A3.1, seguido por la ejecución paralela de: a) el sub-flujo de trabajo del área A3.2 y b) la tarea T2 dentro del área A3. El flujo de trabajo

secuencial del área A3 termina definiendo un punto de unión del que depende la ejecución del sub-flujo de trabajo del área A3.3.

Este ejemplo destaca el principio de la gestión distribuida e independiente del flujo de trabajo de una organización. Este principio se basa en la técnica conocida como interconexión anidada de cajas negras. La simplicidad y la eficacia obtenidas mediante la estructuración y la gestión del flujo de trabajo global de una organización demuestra una vez más el gran interés de organizar y administrar una organización que sigue una jerarquía de áreas englobadas.

3.2 Escenario de uso de SEDINU

El sistema SEDINU (*SErvice DIscovery for Nomadic Users*) ha sido diseñado con el fin de ser implantado dentro de la institución CINVESTAV-IPN. El sistema SEDINU se ayuda del sistema RBAC-Soft [Mendoza et al., 2009] para complementar sus funciones. RBAC-Soft es un sistema de control de acceso basado en roles, el cual es responsable de manejar las listas de roles, recursos y permisos definidos por cada área autónoma.

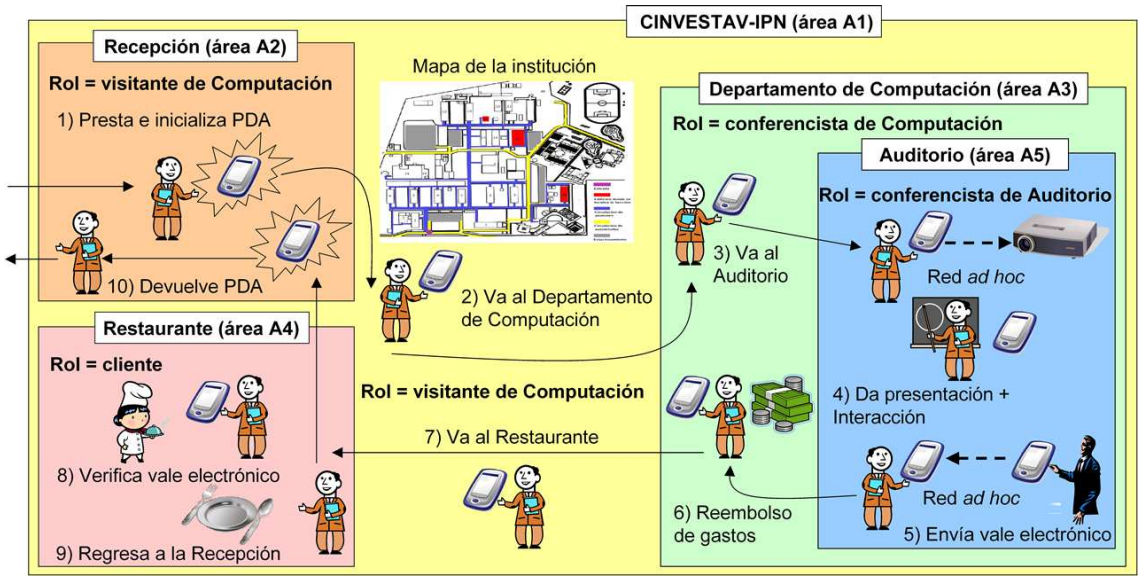

Figura 3.3: Escenario de uso del sistema SEDINU

Suponga que la organización ha sido dividida en cinco áreas jerarquizadas (véase la Figura 3.3) las cuales están estructuradas de la siguiente manera: la primera área (A1) corresponde al `CINVESTAV-IPN` (la raíz de la jerarquía); esta área incluye tres sub-áreas: la `Recepción` (área A2), el `Departamento de Computación` (área A3) y el `Restaurante` (área A4); a su vez, el `Departamento de Computación` abarca el `Auditorio` (área A5).

Dentro del `CINVESTAV-IPN`, es posible establecer redes *ad hoc* Bluetooth para apoyar las comunicaciones de proximidad, e.g., transferencia de los mensajes de un usuario de una PDA a un sistema de conferencia o a otro usuario de una PDA. Esta institución está

totalmente cubierta por una red inalámbrica (WiFi) que proporciona apoyo a la comunicación entre las diferentes áreas. Además, los usuarios nómadas utilizan principalmente dispositivos móviles WiFi (e.g., PDAs, teléfonos celulares o laptops) para interactuar con los servicios ofrecidos por las diferentes áreas y/o colaborar con otros usuarios.

Con el fin de mostrar las funcionalidades del sistema SEDINU, suponga que un investigador llega al `CINVESTAV-IPN` (área A1), con el objetivo de impartir una conferencia en el `Departamento de Computación` (área A3). En primer lugar, el investigador se registra en la `Recepción` (área A2), donde recibe una PDA: RBAC-Soft le asigna el rol de `visitante de Computación` y carga un flujo de trabajo general en su PDA para guiarlo y ayudarlo en sus actividades dentro del `CINVESTAV-IPN` (ver Figura 3.3 ref. #1). Por medio de un mapa de la organización y la tecnología GPS, el sistema SEDINU le indica uno de los posibles caminos para llegar al `Departamento de Computación` (ver Figura 3.3 ref. #2).

Cuando el visitante arriba al `Departamento de Computación` (área A3), el sistema SEDINU determina sus coordenadas, utilizando una de las dos siguientes técnicas (la que esté disponible):

- la primera se refiere a un sistema de reconocimiento de caras [Garcia et al., 2008], que permite identificar dinámicamente a los usuarios y ubicarlos en lugares específicos (e.g., espacios externos, salas de reunión, pasillos) donde han sido instaladas cámaras de video; la finalidad de esta técnica es seguir a cada usuario nómada en vez de localizar su dispositivo móvil, del cual puede estar separado por algún momento;

- la segunda técnica consiste en localizar el dispositivo móvil del usuario nómada utilizando el método de triangulación de señales WiFi [Herborn et al., 2005].

Estas dos diferentes técnicas de localización son complementarias por lo que se pueden combinar de manera fructuosa.

El sistema SEDINU transmite las coordenadas del visitante al sistema RBAC-Soft. Como el edificio comprende varias áreas, RBAC-Soft relaciona las coordenadas del visitante con el `Departamento de Computación` (área A3) y, en consecuencia, le atribuye el rol de `conferencista de Computación`. De forma complementaria, el sistema RBAC-Soft carga un flujo de trabajo específico en la PDA del conferencista para guiarlo en sus actividades dentro del área A3. Por su parte, el sistema SEDINU utiliza la tecnología WiFi y le proporciona un mapa para ayudarlo a localizar el `Auditorio` (área A5) donde dará su conferencia (ver Figura 3.3 ref. #3).

Tan pronto como el conferencista llega al `Auditorio`, el sistema SEDINU determina sus nuevas coordenadas y las envía al sistema RBAC-Soft, el cual las asocia con el `Auditorio` (área A5) y luego le asigna el rol de `conferencista de Auditorio`. El sistema SEDINU ofrece al conferencista el servicio de `proyección de diapositivas` que le permite realizar su presentación (ver Figura 3.3 ref. #4). De esta manera, se crea dinámicamente una red *ad hoc* Bluetooth entre la PDA del conferencista y dicho servicio con el fin de: 1) transferir sus diapositivas y 2) controlar su presentación mediante la PDA. Durante la conferencia, todos los asistentes (quienes juegan el rol de `oyente`) pueden utilizar el servicio de `proyección de diapositivas` y consultar o anotar la presentación. Durante

la sesión de preguntas, todos los usuarios (conferencista, *chairman* y oyentes) pueden establecer una sesión de preguntas-respuestas apoyada por el servicio de `proyección de diapositivas`.

Cuando termina la presentación, el conferencista es invitado a tomar un almuerzo en el `Restaurante` del `CINVESTAV-IPN`. Por lo tanto, el *chairman* de la conferencia transfiere un `vale electrónico` desde su PDA al PDA del conferencista a través de una red *ad hoc* Bluetooth (ver Figura 3.3 ref. #5). A continuación, el sistema SEDINU proporciona al conferencista la información pertinente para dirigirlo al `Restaurante` (área A4), donde actúa con el rol de `cliente` (ver Figura 3.3 ref. #6 y #7). Allí, el cajero verifica y carga el `vale electrónico` del cliente (ver Figura 3.3 ref. #8). Una hora más tarde, el cliente sale del `Restaurante` (área A4) y regresa a la `Recepción` (área A2) con el rol de `visitante saliente`, guiado en todo momento por el sistema SEDINU (ver Figura 3.3 ref. #9). Por último, el visitante devuelve la PDA al encargado de la `Recepción` (ver Figura 3.3 ref. #10) y sale del `CINVESTAV-IPN` (área A1).

3.3 Arquitectura funcional de SEDINU

El sistema SEDINU ha sido diseñado con la finalidad de permitir a un usuario nómada, localizado en un área autónoma: a) colaborar con otros usuarios, e.g., producción cooperativa de información e b) interactuar con los servicios disponibles en dicha área, e.g., solicitud de información. Las interacciones del usuario nómada están dirigidas por un flujo de trabajo sensible al contexto (definido en términos de la ubicación, del rol y de los objetivos del usuario). Cada área administrada por el sistema SEDINU ha sido concebida como una entidad que se autoadministra y que se apoya en el sistema RBAC-Soft para comunicarse y coordinarse con otras áreas.

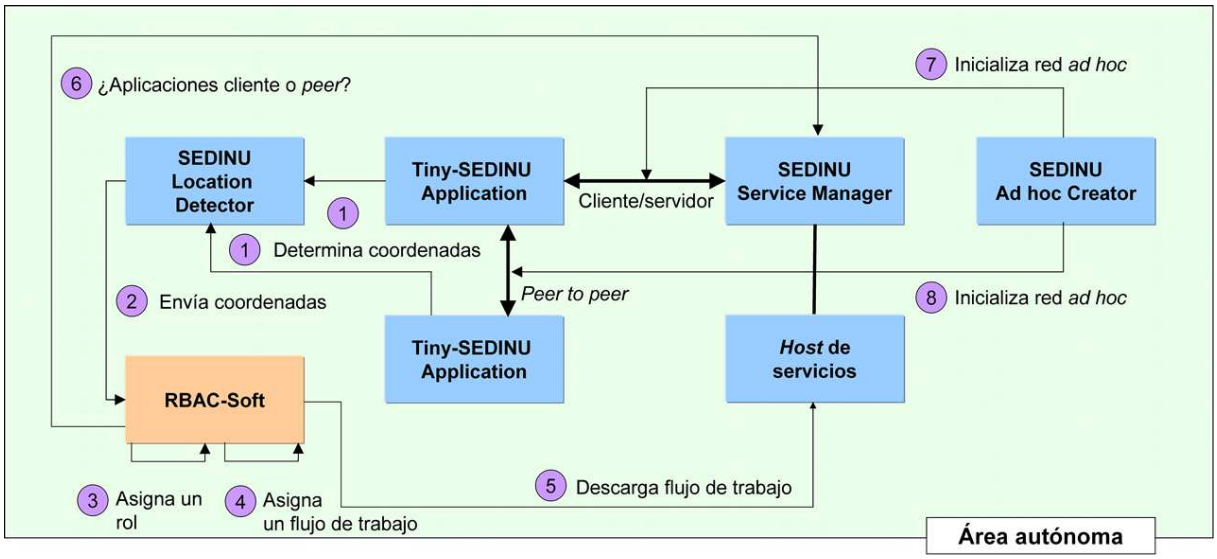

Figura 3.4: Arquitectura funcional del sistema SEDINU

La Figura 3.4 muestra la arquitectura funcional del sistema SEDINU. El módulo `Location Detector` (ver Figura 3.4 ref. #1) determina las coordenadas actuales del usuario nómada dentro de un edificio por medio de las dos técnicas mencionadas en la sección 3.2: 1) un sistema que utiliza triangulación de señales WiFi llamado WiTrack y 2) un sistema de reconocimiento de caras (cf. sección 3.4.1). A continuación, el módulo `Location Detector` transmite estas coordenadas al sistema RBAC-Soft, a fin de definir el área autónoma correspondiente (ver Figura 3.4 ref. #2). Dependiendo de la ubicación actual del usuario nómada, RBAC-Soft determina: a) el rol atribuido al usuario nómada y b) los servicios disponibles a este rol (ver Figura 3.4 ref. #3).

Con base en esta información contextual, el sistema RBAC-Soft crea un flujo de trabajo (o busca uno ya definido) para guiar al usuario nómada a lograr sus objetivos (ver Figura 3.4 ref. #4). Cada una de las tareas del flujo de trabajo está asociada a un conjunto de servicios que permite a los usuarios nómadas llevar a cabo dichas tareas. De esta manera, RBAC-Soft carga el flujo de trabajo correspondiente en el dispositivo móvil del usuario nómada (ver Figura 3.4 ref. #5) y solicita al módulo `Service Manager` que cargue las aplicaciones cliente o *peer* para que el usuario nómada pueda acceder a los servicios requeridos (ver Figura 3.4 ref. #6).

Tan pronto como el usuario nómada tiene acceso a los servicios, este puede seleccionar uno de ellos desde su dispositivo móvil. El conjunto de servicios disponibles en un momento dado está controlado por el correspondiente flujo de trabajo, e.g., el flujo de trabajo puede activar: a) un servicio, en el caso de una única tarea serializada o b) varios servicios, en el caso de tareas cuyo orden de ejecución es irrelevante.

De esta manera, el módulo `Ad hoc Creator` (ver la Figura 3.4 ref. #7) establece dinámicamente una red *ad hoc*: a) entre el módulo `Tiny-SEDINU Application` (que se ejecuta en el dispositivo móvil del usuario) y el módulo `Service Manager` o b) entre dos módulos `Tiny-SEDINU Application` que se ejecutan en los dispositivos móviles de los usuarios nómadas que van a interactuar (ver la Figura 3.4 ref. #8). Este último tipo de red *ad hoc* facilita tanto el intercambio directo de información (e.g., facturas o vales) como la colaboración entre usuarios por medio de una comunicación *peer to peer*, en vez de pasar por un servidor central.

3.4 Arquitectura de distribución de SEDINU

Una arquitectura de distribución define la repartición de las entidades y de los componentes de un sistema entre los sitios implicados en un proceso de colaboración. Las arquitecturas de distribución difieren principalmente en tres aspectos: 1) la representación de cada entidad compartida en los sitios participantes; 2) el número de ejemplares de cada entidad que están presentes en el sistema y 3) la posible movilidad de cada entidad entre dichos sitios.

La arquitectura de distribución del sistema SEDINU comprende tres tipos de sitios que albergan diferentes componentes. Un tipo de sitio A aloja al módulo `Tiny-SEDINU Application` (ver Figura 3.5 ref. A), el cual puede jugar el rol de cliente o *peer* ya que permite a un usuario nómada interactuar respectivamente con los servicios del área autónoma o con otros usuarios nómadas. Puede existir varios sitios de tipo A, al menos

uno por cada usuario nómada.

Por otra parte, existen dos tipos de sitios que juegan el rol de servidor:

- el tipo de sitio B almacena el módulo `Location Detector` (sistema de reconocimiento de caras) y la información que requiere y produce (ver Figura 3.5 ref. B). Puede existir varios sitios de este tipo, uno por cada cámara ubicada en la organización;
- el tipo de sitio C aloja: 1) al sistema RBAC-Soft, el cual recibe las coordenadas de un usuario nómada y las relaciona con un área autónoma y, además, es responsable de asignar roles y flujos de trabajo a dicho usuario, 2) al módulo `Service Manager`, que almacena información sobre los servicios y flujos de trabajo de un área autónoma y 3) al módulo `Ad hoc Creator`, el cual está encargado de la inicialización de una red *ad hoc* entre dos módulos `Tiny-SEDINU Application` o entre un módulo `Tiny-SEDINU Application` y un módulo `Service Manager` (ver Figura 3.5 ref. C).

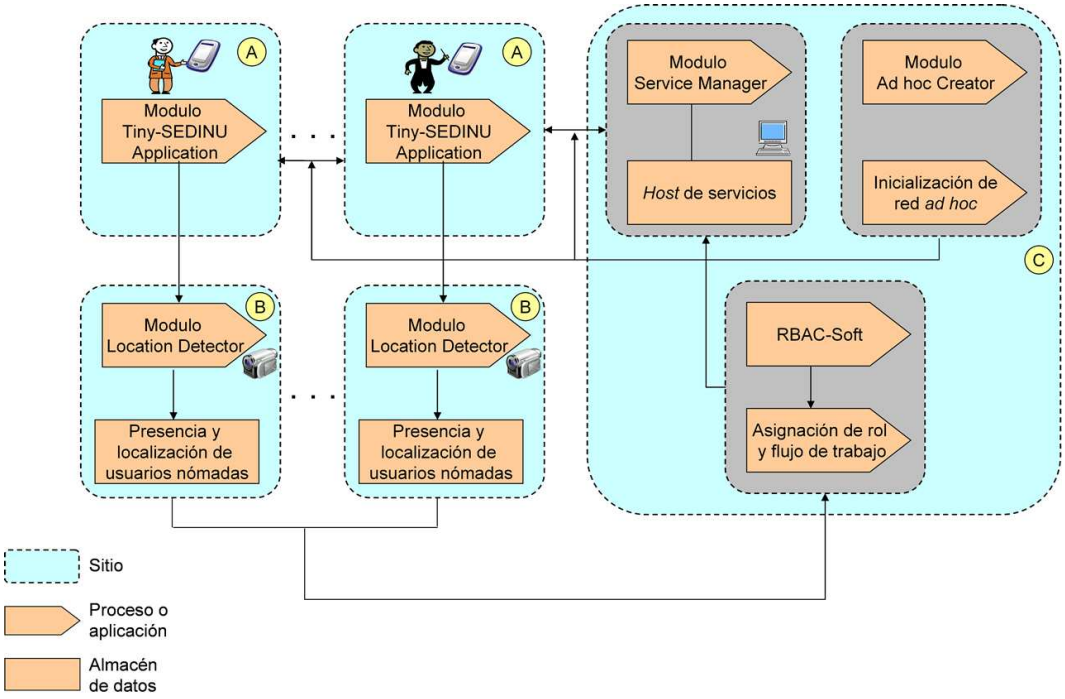

Figura 3.5: Arquitectura de distribución del sistema SEDINU

En el caso de una organización que consta de un solo edificio, podría colocarse un solo sitio de tipo C sin considerar por supuesto la propiedad de tolerancia a fallas. Sin embargo, si el sistema SEDINU es implantado en una organización que cuenta con varios edificios, podría ser necesario colocar un sitio de tipo C en cada edificio. La replicación de los componentes albergados en este tipo de sitio es necesaria, sobre todo, cuando los edificios están físicamente distantes, como es el caso del Departamento de Computación del CINVESTAV-IPN campus D.F. y del Laboratorio de Tecnologías de la Información de Tamaulipas.

3.5 Módulos del sistema SEDINU

En esta sección se describen los módulos que componen el sistema SEDINU: 1) `Location Detector`, encargado de calcular la posición actual de un usuario nómada dentro de un edificio, 2) `Tiny-SEDINU Application`, aplicación embebida en los dispositivos móviles que ayuda a un usuario nómada a colaborar con otros usuarios y a interactuar con los servicios disponibles en el área autónoma, 3) `Service Manager`, responsable de cargar una aplicación cliente o *peer* para que el usuario nómada acceda a dichos servicios y 4) `Ad hoc Creator` encargado de establecer dinámicamente una red *ad hoc*: a) entre el módulo `Tiny-SEDINU Application` y el módulo `Service Manager` o b) entre dos módulos `Tiny-SEDINU Application`.

3.5.1 Módulo *Location Detector*

Este módulo del sistema SEDINU se encarga de detectar las coordenadas actuales de un usuario nómada y de enviarlas al sistema RBAC-Soft para que sean procesadas. Se propone dos técnicas de localización. Una de ellas concierne el sistema WiTrack, que localiza el dispositivo móvil de un usuario nómada dentro de un área autónoma, mediante el método de triangulación de señales WiFi. La otra técnica se refiere a un sistema de reconocimiento de caras, que identifica y ubica a un usuario nómada en un lugar específico.

WiTrack

El éxito del posicionamiento en exteriores y de las aplicaciones basadas en el sistema de posicionamiento global (GPS) [Zhu and Li, 2008] ofrece un incentivo a la investigación y al desarrollo de sistemas de posicionamiento en interiores. Desafortunadamente, el sistema GPS no puede utilizarse en interiores ni en áreas urbanas densas, debido a su débil recepción de señal cuando al menos tres satélites GPS no tienen líneas de visión de un dispositivo móvil. Como resultado, los sistemas de posicionamiento en interiores requieren medios alternativos para detectar la posición de un dispositivo móvil, sin depender de la señal de radio-frecuencia (RF) de los satélites GPS. WiTrack trata de abordar esta cuestión con base en la tecnología WiFi (ver Figura 3.6).

WiTrack es un sistema de código abierto, desarrollado por el equipo EPITECH WiTrack's Team, para el monitoreo en tiempo real del posicionamiento de dispositivos WiFi en interiores. Utiliza la red inalámbrica y las posiciones de los puntos de acceso IEEE 802.11 para poder estimar la ubicación de cualquier dispositivo WiFi, por medio del método de triangulación de señales WiFi [Herborn et al., 2005].

WiTrack puede colocarse en la parte superior de cualquier red WLAN existente, por lo tanto es posible ahorrar el costo de una infraestructura dedicada. Por otra parte, utiliza señales de radio-frecuencia que pueden penetrar la mayoría de los materiales interiores, lo que permite reducir el número de puntos de acceso necesarios para efectos de posicionamiento. Debido a que la intensidad de la señal (RSS) puede medirse a través de las tarjetas de interfaz de red inalámbrica, no es necesaria ninguna tarjeta dedicada para las computadoras portátiles actuales y PDAs con interfaz IEEE 802.11 [Gaertner and Cahill, 2004].

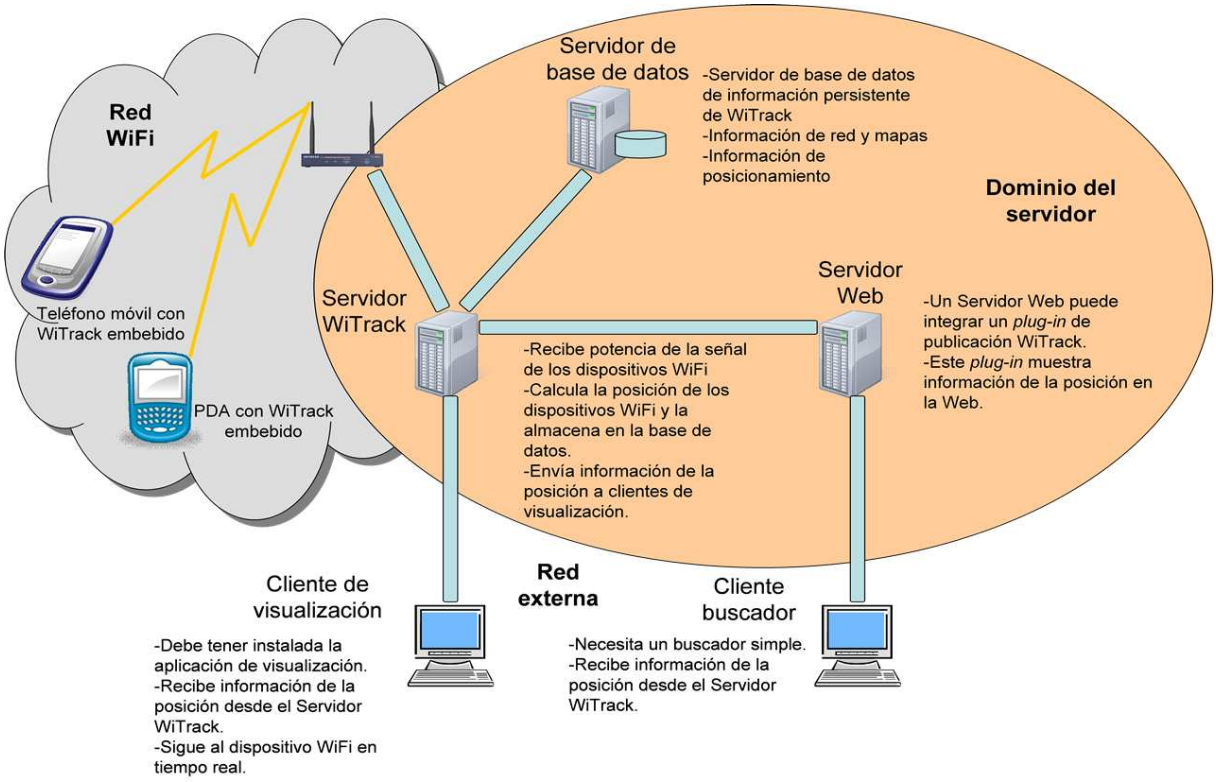

Figura 3.6: Arquitectura del sistema WiTrack

¿Cómo trabaja WiTrack?

Primeramente, un dispositivo WiFi, que tiene embebido el sistema WiTrack, explora ondas de radio de señales WiFi y reúne información de los puntos de acceso. Posteriormente, el dispositivo WiFi envía esta información a la red inalámbrica mediante una señal que se transmite en un intervalo regular. La señal es recibida por puntos de acceso inalámbricos estándar, los cuales la transmiten a un proceso de localización denominado WiTrack *engine* (ver Figura 3.7).

Posteriormente, WiTrack *engine* utiliza un algoritmo de localización que calcula, en tiempo real, la posición del dispositivo WiFi mediante la comparación de los puntos de acceso observados contra una base de datos de puntos de acceso, cuya localización es conocida por WiTrack *manager*. Estos datos de localización son almacenados en la base de datos y por último son utilizados por WiTrack *viewer* para desplegar la ubicación de los dispositivos WiFi en mapas.

WiTrack manager

Witrack *manager* desempeña un papel importante en el proceso de informar a Witrack *engine* acerca de la localización de los dispositivos WiFi. Particularmente, WiTrack *manager* realiza las siguientes tres funciones: 1) la creación de mapas, 2) la configuración de la topología del sistema y 3) el encapsulamiento de la forma en que se propaga la señal

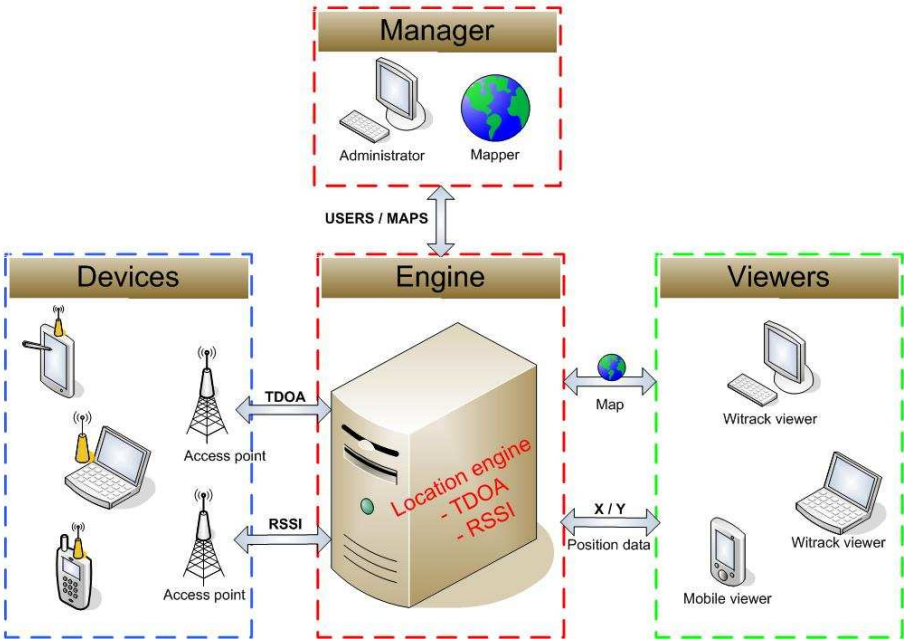

Figura 3.7: Módulos de WiTrack

de radio en relación con la distancia, el entorno físico y la localización.

La mayoría de la información, e.g., el plano del edificio, proviene de organizaciones, e.g., empresas, universidades y departamentos, que conocen la ubicación de los puntos de acceso IEEE 802.11, ya que esta información comúnmente se registra como parte de una estrategia de despliegue y mantenimiento. Un ejemplo de una arquitectura de red propuesta por una organización [Roth, 2006] se menciona a continuación: a) una red comprende uno o varios edificios, b) un edificio se compone de uno o varios pisos y c) un piso contiene una o varias habitaciones. Witrack *manager* también permite cargar elementos (e.g., puntos de acceso y paredes) creados por otro software para representar edificios. Cada elemento está caracterizado por una estructura de datos (ver Tabla 3.8).

Estructura de datos de un punto de acceso	**Estructura de datos de una pared**
Modelo 3D	Modelo 3D
Fuerza de emisión de señal	Coeficiente de reducción de señal
Características de la antena (omnidireccional o multidireccional)	Coeficiente de reflexión de señal
Tipo de punto de acceso	Composición de la pared (e.g., concreto, metal, madera)

Figura 3.8: Estructura de datos de un punto de acceso y de una pared

Witrack viewer

Witrack *viewer* permite mostrar en tiempo real la posición de uno o varios dispositivos WiFi a través de una representación 3D (ver Figura 3.9).

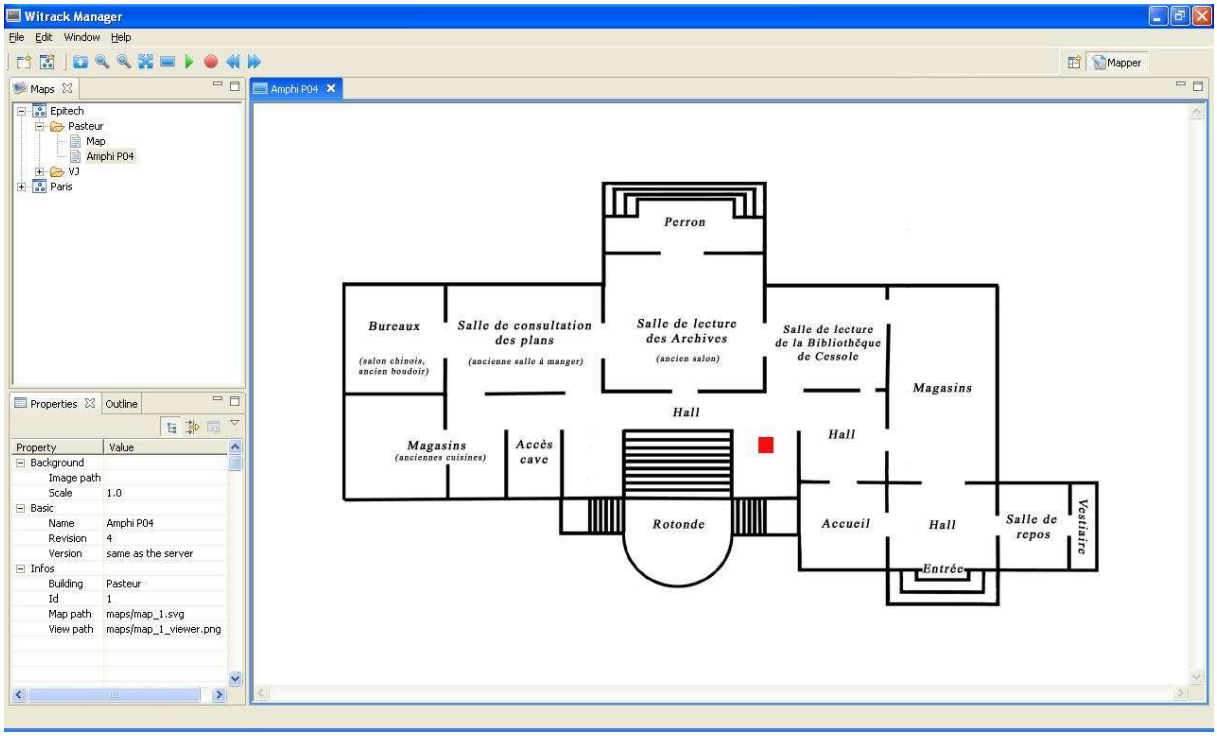

Figura 3.9: Mapa de un edificio generado por WiTrack *viewer*

Witrack device

Es una aplicación embebida en el dispositivo WiFi que ofrece las siguientes funciones: a) identificación del dispositivo, b) adquisición de la potencia de recepción de los puntos de acceso que lo rodean, c) recarga periódica de la información de la señal y d) envío de datos al servidor.

Witrack engine

Witrack *engine* recibe la información de los dispositivos WiFi, luego calcula su posición con el fin de almacenarla en una base de datos y finalmente despliega la posición del dispositivo WiFi por medio del módulo WiTrack *viewer*.

Sistema de reconocimiento de caras

El sistema de reconocimiento de caras es una adaptación de un sistema de reconocimiento de nebulosas planetarias, que fue realizado por el equipo del Dr. Gustavo Olague del Centro de Investigación Científica y de Educación Superior de Ensenada (CICESE) en colaboración con el Instituto de Astronomía de la UNAM. Esta adaptación fue posible gracias al trabajo realizado por la M. en C. Kimberly García del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV-IPN) en con-

junto con el mencionado Dr. Olague y con el Dr. Dominique Decouchant del Centre Nacional de Recherche Scientifique (CNRS) de Francia.

El sistema de reconocimiento de caras pretende identificar a los usuarios nómadas, en vez de los dispositivos móviles que portan (e.g., PDA y *Smartphone*) los cuales son fácilmente transferibles y olvidables [Garcia, 2009]. Por esta razón, si se asume que un usuario nómada siempre se encuentra en el mismo lugar que su dispositivo móvil, esta suposición podría generar información errónea sobre la ubicación del usuario nómada.

El sistema de reconocimiento de caras evita los problemas de transferencia y olvido de dispositivos móviles. Un usuario nómada, que interactúa con sus colegas por medio del sistema SEDINU, no necesita llevar consigo ningún dispositivo móvil para ser localizado, sino únicamente para interactuar con los servicios ofrecidos por el área autónoma en la que se encuentra. El módulo `Location Detector` determina la identificación y la ubicación de un usuario nómada mediante el procesamiento de las imágenes de su cara, las cuales son capturadas por medio de cámaras distribuidas en el espacio físico. El sistema de reconocimiento de caras consta de dos fases:

1. **fase de aprendizaje**: consiste en entrenar al sistema para que reconozca a los colaboradores que se requiere identificar;

2. **fase de reconocimiento o pruebas**: se encarga de hacer predicciones sobre la identidad y la ubicación de los colaboradores.

La primera fase se realiza sólo una vez, mientras que la segunda fase se efectúa cada vez que una cámara capta una cara humana (ver Figura 3.10).

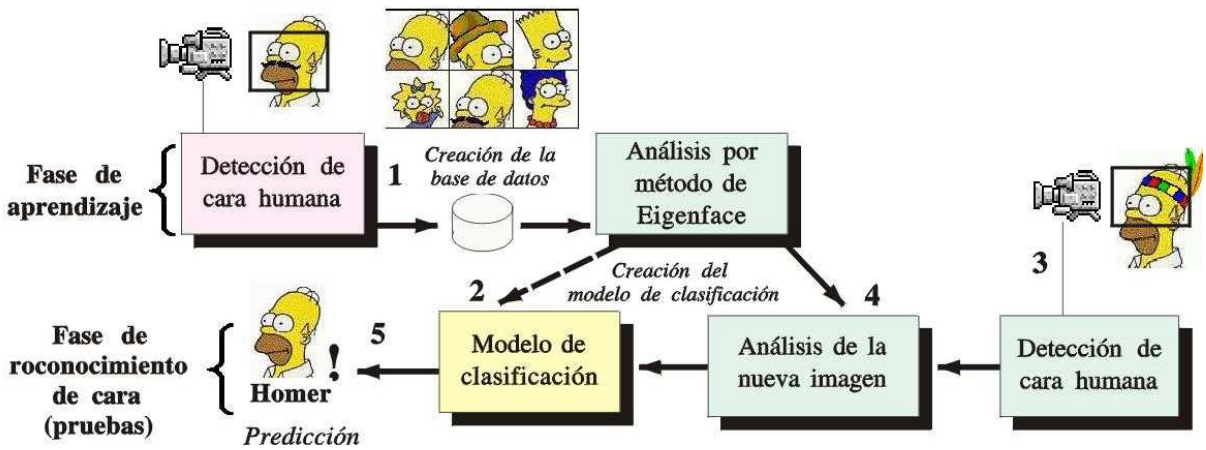

Figura 3.10: Fases del sistema de reconocimiento de caras

La fase de aprendizaje utiliza un algoritmo que permite diferenciar una cara humana del resto de los objetos captados por una cámara. Este algoritmo además permite construir una base de datos (ver Figura 3.10 ref. #1), la cual está compuesta de un conjunto de imágenes de cada colaborador que se desea reconocer. Cada conjunto de imágenes contiene fotografías de la cara de un colaborador en diversas poses y gesticulaciones.

Posteriormente, la base de datos es analizada mediante un método matemático, llamado Eigenfaces [Hernández et al., 2007], el cual permite obtener las características más significativas que diferencian a un colaborador de otro. Los resultados de este análisis matemático permiten crear un modelo de clasificación (ver Figura 3.10 ref. #2), el cual ayudará a identificar a qué colaborador pertenecen las imágenes obtenidas durante la fase de pruebas. Como ya se mencionó, dichas imágenes son capturadas por cámaras localizadas en lugares específicos.

La fase de pruebas inicia cuando el algoritmo, que diferencia caras humanas de otros objetos, detecta una cara humana y captura una imagen (ver Figura 3.10 ref. #3). Esta imagen es analizada tomando en cuenta tanto la información que se obtuvo del análisis de las imágenes de la base de datos, como la información de la imagen recién capturada (ver Figura 3.10 ref. #4). Mediante este análisis se obtienen las características principales de la nueva imagen, las cuales son evaluadas por el modelo de clasificación que se creó en la fase de aprendizaje (ver Figura 3.10 ref. #5). Este modelo se encarga de hacer una predicción sobre la identidad y la ubicación del colaborador al que se le tomó dicha fotografía.

Las dos tecnicas de localización anteriormente descritas son complementarias. Lamentablemente el proyecto WiTrack ha sido abandonado por sus creadores.

3.5.2 Módulo *Tiny-SEDINU Application*

Este módulo está embebido en el dispositivo móvil del usuario nómada. Se trata de una aplicación Java ME (*Micro Edition*) que ayuda al usuario nómada a interactuar con los servicios disponibles en el área autónoma actual o con otros usuarios nómadas por medio de una interfaz de usuario "amigable".

Figura 3.11: Vista de *Tiny-SEDINU Application* al elegir un servicio

Esta aplicación tiene tres funciones principales: 1) guiar al usuario nómada durante su recorrido por la organización; 2) desplegar en la pantalla del dispositivo móvil los servicios ofrecidos por el área autónoma actual y 3) ofrecer las herramientas necesarias para permitir al usuario nómada interactuar con dichos servicios o con otros usuarios nómadas. En la Figura 3.11, se muestra un ejemplo de un estudiante que al entrar al área 3 recibe, en su dispositivo móvil, una notificación que le hace saber que tiene la posibilidad de acceder a un servicio de impresión en blanco y negro.

3.5.3 Módulo *Service Manager*

El módulo `Service Manager` carga las aplicaciones cliente o *peer* en el dispositivo móvil del usuario nómada para permitirle respectivamente: 1) acceder a los servicios ofrecidos por el área autónoma actual o 2) interactuar con otros usuarios nómadas. El módulo `Service Manager` está asociado a un *host*, que contiene todos los servicios disponibles en el área autónoma actual. Dichos servicios son descargados en este *host* por el sistema RBAC-Soft [Mendoza et al., 2009]. El módulo `Service Manager` interactúa con el módulo `Tiny-SEDINU Application` por medio de una red *ad hoc*, creada por el módulo `Ad hoc Creator` (cf. sección 3.4.4).

El tipo de conexión empleado para establecer una red *ad hoc* es Bluetooth. Existen dos tipos de conexiones Bluetooth: 1) conexiones estáticas y 2) conexiones dinámicas, las cuales se describen a continuación.

Conexiones Bluetooth estáticas

El tipo más simple de conexiones Bluetooth es una conexión estática, la cual facilita la transferencia de datos entre dos dispositivos que se conocen entre sí. Un uso común de este tipo de conexiones Bluetooth se presenta cuando un usuario desea sincronizar datos (e.g., citas de una agenda) o transferir archivos/aplicaciones entre una PC y un dispositivo móvil (ver Figura 3.12).

Figura 3.12: Apertura de una conexión estática entre dispositivos conocidos

Cuando un usuario desea transferir archivos o sincronizar datos entre dos dispositivos

conocidos se necesita un proceso de autenticación. Los dispositivos implicados por lo general son emparejados mediante la introducción de la misma clave de acceso en ambos dispositivos.

Si se requiere una autorización (generalmente es el caso), el usuario podrá aceptar la entrada de una conexión Bluetooth si asi lo desea. Sin embargo, una conexión proveniente de un dispositivo conocido puede ser definida con carácter confiable para que sea aceptada de manera automática. Este acuerdo puede establecerse de tal forma que los dispositivos abran automáticamente una conexión cuando se detecten entre sí.

Conexiones Bluetooth dinámicas

Una conexión dinámica es un tipo más avanzado de conexiones Bluetooth que, además de la transferencia de datos entre dispositivos, permite la búsqueda de dispositivos y la selección de servicios. Puede existir varios dispositivos con los cuales es posible establecer una conexión, pero es importante conocer cuál de ellos cuenta con el servicio requerido (ver Figura 3.13).

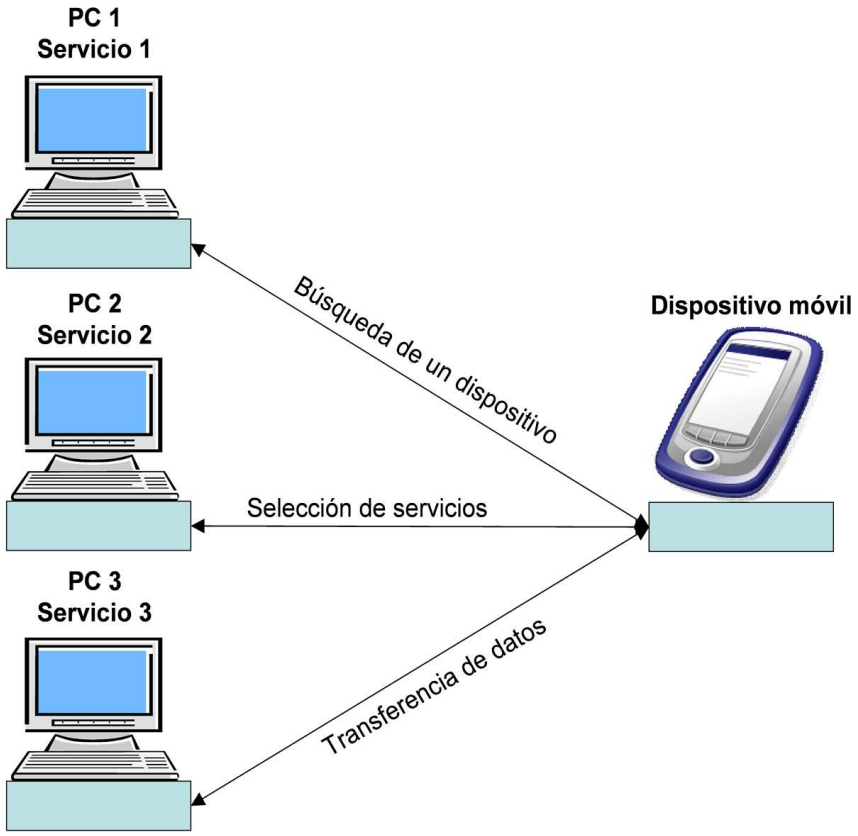

Figura 3.13: Apertura de una conexión dinámica entre dispositivos desconocidos

Un ejemplo interesante de este tipo de conexiones Bluetooth se refiere a un servicio comercial (e.g., la impresión de imágenes desde un teléfono móvil) proporcionado por una empresa. Cuando el usuario entra a un fotoestudio, se inicia el software de impresión

fotográfica en su teléfono móvil. Este software busca un dispositivo Bluetooth (e.g., una PC) que ofrezca un servicio de transferencia de imágenes. Una vez encontrada la PC, se establece una conexión entre esta y el teléfono móvil del usuario. A continuación, el teléfono móvil solicita al usuario que seleccione las imágenes a imprimir. Una vez transferidas todas las imágenes a la PC, se cierra la conexión.

En este caso, la PC y el teléfono móvil no saben uno del otro. El primer paso para establecer una conexión entre ellos es permitir que la PC y el teléfono móvil se encuentren. El siguiente paso consiste en emparejar estos dispositivos para poder establecer la conexión e iniciar la transferencia de datos (e.g., imágenes).

La búsqueda de dispositivos Bluetooth se inicia normalmente de forma manual desde el teléfono móvil o la PC. El emparejamiento de los dispositivos igualmente se realiza de forma manual. Sin embargo, estas tareas pueden ser automatizadas para reducir en gran medida las acciones realizadas por el usuario.

3.5.4 Módulo *Ad hoc Creator*

El módulo `Ad hoc Creator` es responsable de inicializar dinámicamente una red *ad hoc*: 1) entre el módulo `Tiny-SEDINU Application`[1] y el módulo `Service Manager`[2] con el objetivo de facilitar la interacción entre un usuario nómada y los servicios disponibles en un área autónoma y 2) entre dos módulos `Tiny-SEDINU Application` a fin de permitir que dos usuarios nómadas puedan iniciar una sesión de interacción y/o colaboración.

Antes de establecer una red *ad hoc* Bluetooth entre una PC y un dispositivo móvil, es necesario considerar los siguientes problemas: ¿Qué software Bluetooth se debe utilizar en la PC para acceder al dispositivo? y ¿Qué controlador de dispositivo se debe emplear para establecer la comunicación con la PC? Existen dos soluciones a estos problemas: 1) la solución basada en el puerto COM virtual y 2) la solución basada en JSR-82.

Solución basada en el puerto COM virtual

Esta solución se basa en la creación de puertos COM virtuales en ambos dispositivos, los cuales siguen una arquitectura de comunicación cliente/servidor. Esta solución resulta ser la mejor elección cuando se emplea conexiones Bluetooth estáticas, ya que el dispositivo que requiere el servicio conoce previamente el número de puerto COM y el tipo de servicio que ofrece el dispositivo con el que puede establecer comunicación.

Arquitectura

Desde el punto de vista del programador, la arquitectura de la solución basada en el puerto COM virtual es simple y transparente del lado de la PC (ver Figura 3.14). Sin embargo, del lado del dispositivo móvil no existe mucha diferencia entre la solución basada en el puerto COM virtual y la solución basada en JSR-82.

Diversos componentes participan en la solución basada en el puerto COM virtual. A pesar de que estos componentes no son directamente visibles al programador, es conveniente conocer sus características para lograr un mejor entendimiento de la arquitectura

[1] se ejecuta en los dispositivos móviles de los usuarios nómadas
[2] se ejecuta en un servidor

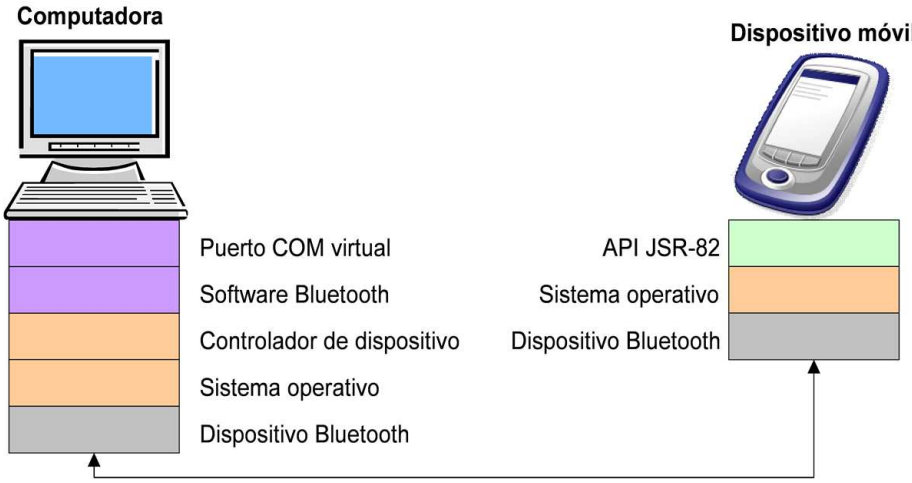

Figura 3.14: Arquitectura de la solución basada en el puerto COM virtual

de esta solución. A continuación, se describe los componentes de la solución basada en el puerto COM virtual, cuya arquitectura general se muestra en la Figura 3.14:

- **Puerto COM virtual:** es un puerto COM creado y manejado exclusivamente mediante software Bluetooth. Por el contrario, los puertos COM reales están asociados a un hardware, de manera que si el programador abre el puerto COM1 o COM2 en una PC, accederá directamente a un puerto COM real.

- **API JSR-82:** está basado en un software Bluetooth y/o en un controlador de dispositivo instalado en el dispositivo móvil (cf. la siguiente sección "Solución basada en JSR-82").

- **Software Bluetooth:** publica API(s) para acceder a los dispositivos Bluetooth y permite la configuración de estos últimos, e.g., dar un nombre a un dispositivo. Un componente importante del software Bluetooth es la interfaz de usuario, ya que facilita la gestión de dispositivos y conexiones, la búsqueda de dispositivos y la creación de *peers*.

- **Controlador de dispositivo:** permite la comunicación con un dispositivo Bluetooth en un nivel más bajo de programación. Un controlador de dispositivo generalmente implementa la pila de protocolos Bluetooth y otros componentes de software de bajo nivel.

- **Sistema operativo:** ejecuta otros componentes software (e.g., administrador de recursos y archivos).

- **Dispositivo Bluetooth:** es hardware habilitado para establecer conexiones Bluetooth. Un dispositivo Bluetooth (e.g., un teléfono móvil) puede comunicarse con una PC o con otro dispositivo Bluetooth (e.g., una PDA).

Software y equipo necesario

El diseño de esta solución requiere: a) un dispositivo Bluetooth compatible con una pila de protocolos Bluetooth (e.g., la de Windows Service Pack 2 y la de Widcomm) y con un software Bluetooth (e.g., el de Windows XP y el de BlueSoleil) y b) el paquete *Javax.com*, que permite a Java 2 Platform, Standard Edition (J2SE) utilizar los puertos COM.

Ventajas y desventajas

Las ventajas de la solución basada en el puerto COM virtual incluyen una fácil implementación, una mejor interoperabilidad y una mayor seguridad de los dispositivos comunicantes, como se describe a continuación:

- **Fácil implementación:** desde el punto de vista del programador, esta solución es fácil de implementar ya que, una vez que la conexión está establecida y activa, el programador solo debe abrir el puerto COM especificado antes de comenzar a leer y escribir datos. La única parte difícil de esta solución consiste en encontrar el número del puerto COM de la PC. El software Bluetooth ofrece normalmente una interfaz de usuario que permite especificar este puerto.

- **Interoperabilidad:** esta solución funciona en cualquier puerto COM virtual disponible. Por lo tanto, una aplicación Bluetooth no necesita tener en cuenta las diferencias entre los dispositivos.

- **Seguridad:** desde el inicio de una conexión Bluetooth, se requiere la participación del usuario, quien sabe de las conexiones activas y conoce exactamente qué dispositivos se están comunicando. Existe una ligera posibilidad de aceptar solicitudes de comunicación de dispositivos desconocidos o que no son de confianza.

- **No es una solución específica para Bluetooth:** los puertos COM virtuales pueden ser utilizados para comunicarse a través de otros medios, capaces de crear puertos COM virtuales (e.g., un cable USB).

Las desventajas de la solución basada en puertos COM virtuales residen en la dificultad de definir puertos virtuales y de crear una conexión, así como en la obligada participación del usuario, como se describe a continuación:

- **Difícil configuración:** como se mencionó anteriormente, el puerto COM virtual debe ser definido por medio de un software Bluetooth. Cada tipo de software tiene su propia interfaz de configuración y sus propios parámetros y publica puertos COM virtuales de diferente manera. Por ejemplo, el software Bluetooth de Widcomm publica el puerto COM virtual, después de haber sido definido en las opciones de configuración, sin embargo el software Bluetooth de Windows XP Service Pack 2 necesita un software que abra el puerto COM virtual, el cual es configurado antes de su publicación.

- **Participación requerida del usuario:** esta solución requiere que el usuario participe activamente en la búsqueda de dispositivos y en el establecimiento de una

conexión entre ellos. Por lo tanto, el usuario debe configurar manualmente el dispositivo, tarea que puede volverse tediosa y frustrante.

Configuración del puerto COM virtual con base en la pila de Bluetooth de Windows XP

Para configurar un dispositivo Bluetooth (e.g., un *smartphone* o una PDA), se requiere que la pila de Bluetooth de Windows XP esté instalada en la PC y sea compatible con dicho dispositivo. Además, el usuario debe asegurarse de que el dispositivo Bluetooth esté conectado a la PC. De otra manera, un gran número de opciones de configuración y aplicaciones no son visibles.

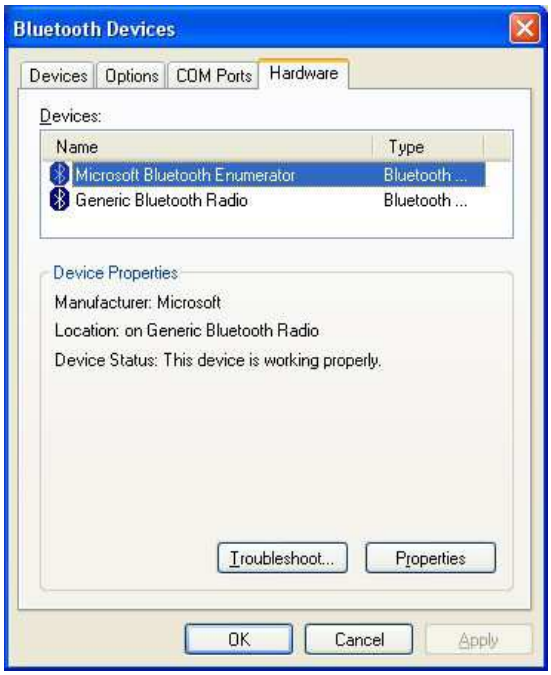

Figura 3.15: Identificación de la pila de Bluetooth de Windows XP

La pila de Bluetooth de Windows XP puede ser reconocida por medio de los siguientes pasos: 1) abrir el panel de control desde el menú "Inicio" (*Start*), 2) abrir el cuadro de diálogo "Dispositivos Bluetooth" (*Bluetooth Devices*) y 3) seleccionar la pestaña "Hardware". La Figura 3.15 muestra el cuadro de diálogo que aparece después de seleccionar esta pestaña. Si las opciones, "*Microsoft Bluetooth Enumerator*" y "*Generic Bluetooth Radio*" están visibles y el estado del dispositivo dice: "Este dispositivo funciona correctamente" (*This device is working properly*), entonces la pila de Bluetooth de Windows XP está configurada.

Para utilizar la solución del puerto COM virtual, primero se debe instalar el paquete *Javax.com* en la PC. A continuación, se debe configurar el puerto COM virtual antes de que pueda ser utilizado. Existen dos opciones de configuración para la pila de Bluetooth de Windows XP: 1) en modo servidor y 2) en modo cliente, las cuales son explicadas a continuación.

La configuración del puerto COM virtual en modo servidor se realiza de la siguiente manera en la PC:

1. Abrir el panel de control desde el menú "Inicio" (*Start*).

Figura 3.16: Lista de Puertos COM virtuales de la pila de Bluetooth de Windows XP

2. Abrir el cuadro de diálogo "Dispositivos Bluetooth" (cf. *Bluetooth Devices* de la Figura 3.16) y seleccionar la pestaña "Puertos COM" (*COM Ports*).

3. Hacer *click* en el botón "Agregar" (*Add*) del cuadro de diálogo "Dispositivos Bluetooth" (cf. *Bluetooth Devices* de la Figura 3.16) con el fin de visualizar el cuadro de diálogo "Agregar puerto COM" (cf. *Add COM Port* de la Figura 3.17) el cual permite al usuario definir qué dispositivo (fijo o móvil) abre la conexión. Por lo general, resulta prudente permitir que el dispositivo móvil abra la conexión (opción "*Incoming*" que ya está seleccionada). Por el contrario, cuando la PC abre la conexión, debe haber un servicio publicado disponible en el dispositivo móvil con el que se puede conectar.

4. Después de hacer *click* en el botón "OK" del cuadro de diálogo "Agregar puerto COM" (cf. *Add COM Port* de la Figura 3.17) se crea un nuevo puerto al que se le asigna un puerto COM, el cual se muestra el cuadro de diálogo "Dispositivos Bluetooth" (cf. *Bluetooth Devices* de la Figura 3.18).

   Windows no publica el puerto COM hasta que haya alguna aplicación que lo abra. Esto significa que este puerto no puede ser encontrado por un dispositivo móvil hasta que una aplicación Bluetooth se ejecute en Windows. A diferencia de la

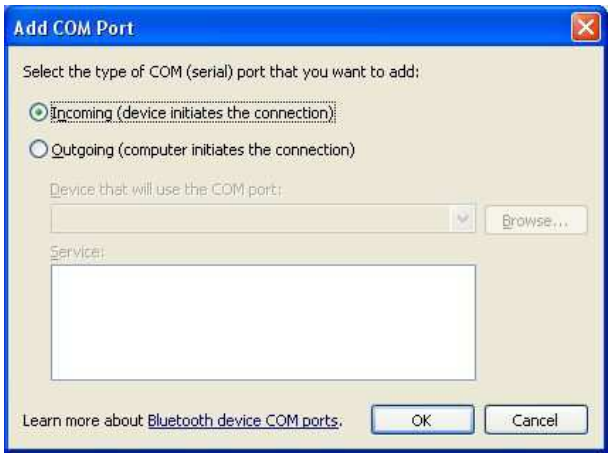

Figura 3.17: Agregación de un puerto COM virtual a la pila de Bluetooth de Windows XP

pila de Bluetooth de Windows XP, la de Widcomm publica el puerto COM virtual después de haber sido agregado. De esta manera, los dispositivos móviles pueden encontrar el puerto COM virtual de inmediato.

Figura 3.18: Puerto COM agregado a la pila de Bluetooth de Windows XP

La configuración del puerto COM virtual en modo cliente se realiza de la siguiente manera:

1. Iniciar una aplicación de perfil de puerto serie en el dispositivo móvil.

Figura 3.19: Sincronización de una PC con un dispositivo móvil que tiene un servicio de puerto COM

2. Abrir el panel de control desde el menú "Inicio" (*Start*) en la PC.

Figura 3.20: Agregación de un nuevo dispositivo Bluetooth mediante el asistente de agregación de dispositivos Bluetooth

3. Abrir el cuadro de diálogo "Dispositivos Bluetooth" (cf. *Bluetooth Devices* de la Figura 3.19) en la PC y seleccionar la pestaña "Dispositivos" (*Devices*). Después, hacer *click* en el botón "Agregar" (*Add*) para iniciar el emparejamiento con el

dispositivo móvil. Sin embargo, si el dispositivo ya está en la lista, se debe retirar seleccionándolo y luego haciendo *click* en el botón "Quitar" (*Remove*).

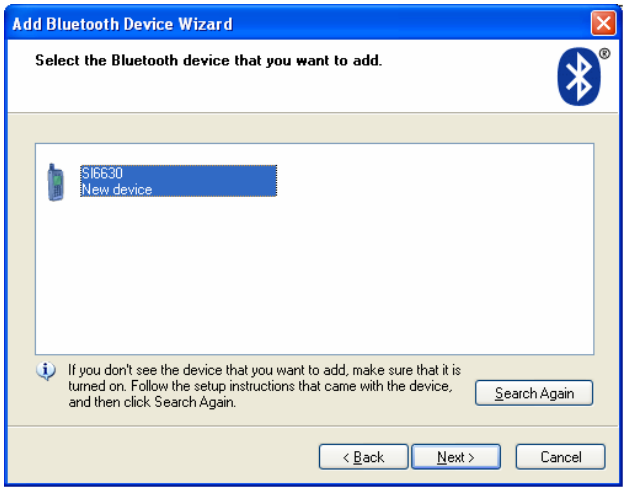

Figura 3.21: Búsqueda de dispositivos mediante el asistente de agregación de dispositivos Bluetooth

4. Seleccionar la casilla "Mi dispositivo está configurado y listo para ser detectado" (*My device is set up and ready to be found*) en el cuadro de diálogo "Asistente de agregación de dispositivos Bluetooth" (cf. *Add Bluetooth Device Wizard* de la Figura 3.20) y hacer *click* en el botón "Siguiente" (*Next*).

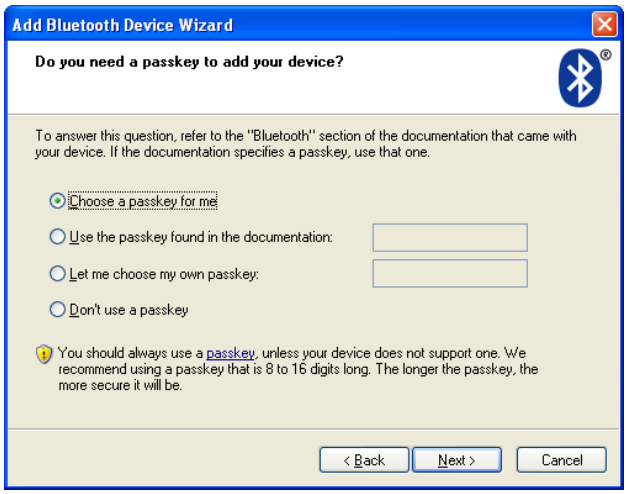

Figura 3.22: Sincronización del dispositivo con la PC mediante una clave de acceso

5. La pila de Bluetooth de Windows XP inicia una búsqueda de nuevos dispositivos. A continuación, se selecciona el dispositivo en la lista del cuadro de diálogo "Asistente de agregación de dispositivos Bluetooth" (cf. *Add Bluetooth Device Wizard* de la Figura 3.21) y se hace *click* en el botón "Siguiente" (*Next*).

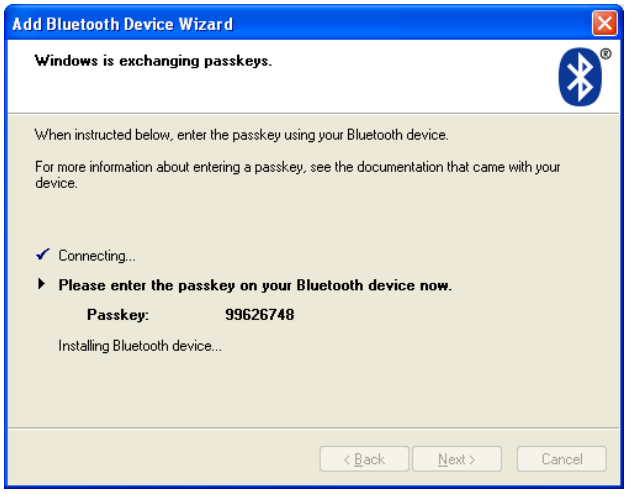

Figura 3.23: Obtención de clave de acceso para sincronizar un dispositivo móvil con una PC

6. Hacer *click* en el botón "Siguiente" (*Next*) del cuadro de diálogo "Asistente de agregación de dispositivos Bluetooth" (cf. *Add Bluetooth Device Wizard* de la Figura 3.22) con el fin de sincronizar la PC con el dispositivo móvil. Es necesario utilizar una clave de acceso para emparejar la PC y el dispositivo móvil, por lo tanto se deja activa la opción "Escoger una clave de acceso" (*Choose a passkey for me*).

Figura 3.24: Finalización del asistente de agregación de dispositivos Bluetooth

7. Dar la clave de acceso que se encuentra en el cuadro de diálogo "Asistente de agregación de dispositivos Bluetooth" (cf. *Add Bluetooth Device Wizard* de la Figura 3.23) mostrado en la PC para aceptar una nueva conexión en el dispositivo móvil.

8. Hacer *click* en el botón "Finalizar" (*Finish*) del cuadro de diálogo "Asistente de

agregación de dispositivos Bluetooth" (cf. *Add Bluetooth Device Wizard* de la Figura 3.24) para completar el proceso de agregación de dispositivos Bluetooth en la PC.

9. Los puertos COM virtuales instalados en la PC pueden ser observados en la pestaña "Puertos COM" (*COM Ports*) del cuadro de diálogo "Dispositivos Bluetooth" (cf. *Bluetooth Devices* de la Figura 3.25).

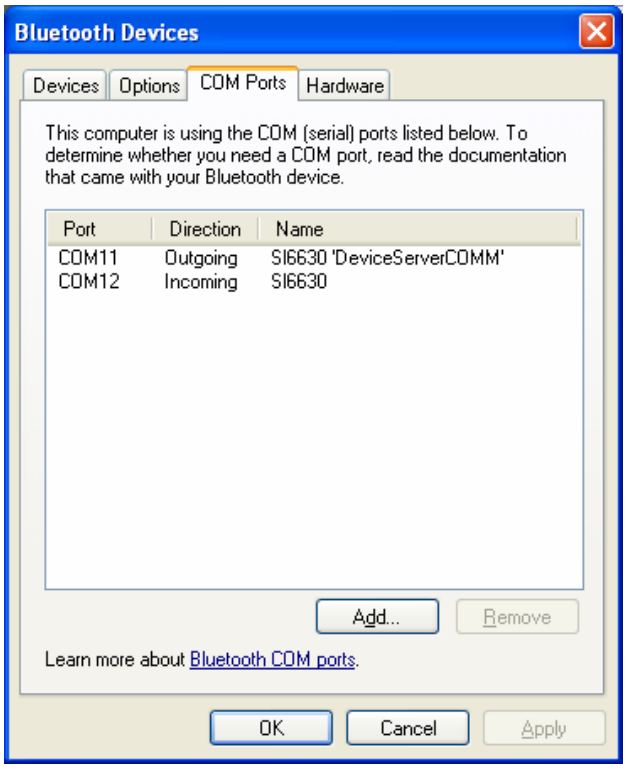

Figura 3.25: Puertos COM virtuales instalados en la PC

Configuración del puerto COM virtual mediante la pila de Bluetooth de Widcomm

El software de Widcomm es más fácil y flexible de configurar en la PC, mediante los siguientes pasos:

1. Abrir el cuadro de diálogo "Configuración de Bluetooth" (cf. *Bluetooth Configuration* de la Figura 3.26) desde el panel de control y seleccionar la pestaña "Servicios Locales" (*Local Services*).

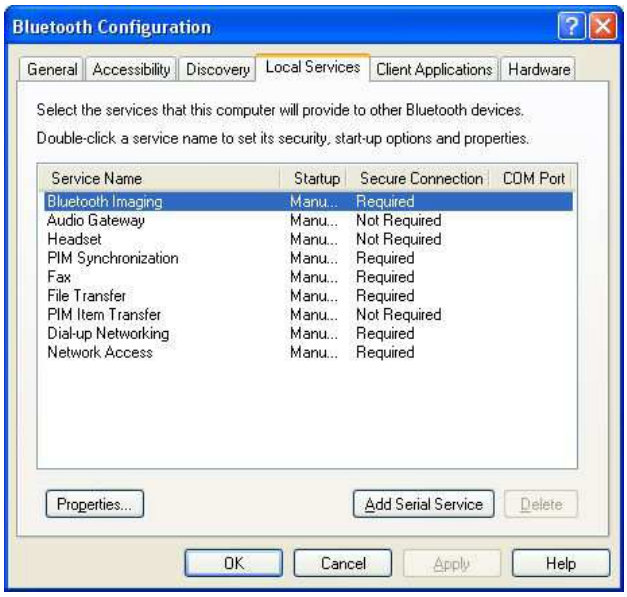

Figura 3.26: Servicios Locales de Widcomm

2. Hacer *click* en el botón "Agregar Servicio Serie" (*Add Serial Service*) del cuadro de diálogo "Configuración de Bluetooth" (cf. *Bluetooth Configuration* de la Figura 3.26) para acceder al cuadro de diálogo "Propiedades Bluetooth" (cf. *Bluetooth Properties* de la Figura 3.27) el cual permite al usuario seleccionar el nombre y número de puerto COM virtual. La casilla de inicio automático debe estar habilitada, a menos que el usuario desee inicializar manualmente el puerto COM virtual.

Figura 3.27: Agregación del puerto COM virtual mediante el software Widcomm

Solución basada en JSR-82

JSR-82 es el API de Java definido para el uso de dispositivos Bluetooth en Java ME. Este API no está configurado de forma predeterminada en las PCs que utilizan J2SE. Sin embargo, existen algunas implementaciones del API JSR-82 para J2SE, que trabajan con las pilas de Bluetooth presentes en las PCs.

Un ejemplo de implementación de JSR-82 para J2SE es BlueCove [Zhang et al., 2006], la cual trabaja con la pila de Bluetooth de Windows XP Service Pack 2 y con la de BlueSoleil. Esta última fue utilizada para realizar las pruebas de las aplicaciones implementadas en este trabajo de maestría.

Arquitectura

La arquitectura de la solución JSR-82 es muy similar a la de la solución del puerto COM virtual. La principal diferencia es que la implementación del API JSR-82 es independiente del software de Bluetooth y del sistema operativo.

El programador debe estar consciente de las posibles limitaciones que cada componente puede presentar cuando utiliza la solución basada en JSR-82. Por lo general, existen restricciones causadas por el controlador de dispositivo o por las implementaciones del software Bluetooth. Las limitaciones en un nivel afectan a todos los niveles superiores. Por ejemplo, las limitaciones del controlador de dispositivo son visibles en los niveles de software Bluetooth y JSR-82. El API JSR-82 está implementado sobre el software Bluetooth y/o el controlador del dispositivo. La Figura 3.28 ilustra la arquitectura general de esta solución.

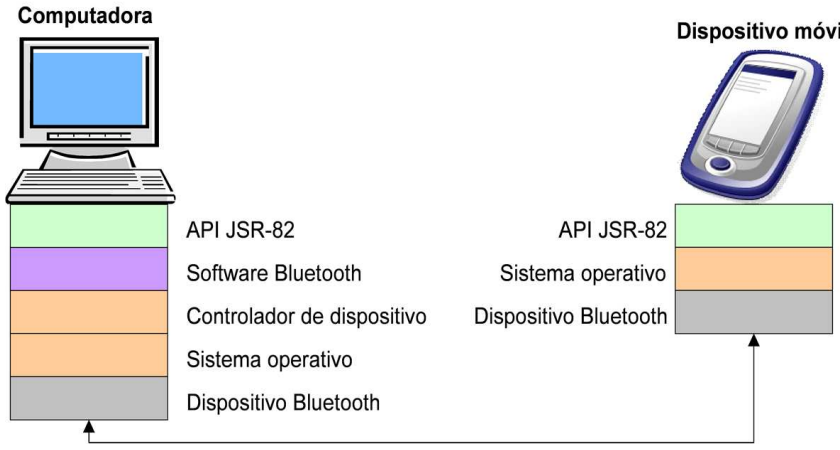

Figura 3.28: Arquitectura de la solución basada en JSR-82

Equipo y software necesario

La solución basada en JSR-82 requiere: a) un dispositivo Bluetooth compatible con la pila de Bluetooth (e.g., la pila de Bluetooth de Windows XP y la de Widcomm) y con el software de la PC y b) una implementación JSR-82.

La mayoría de las implementaciones JSR-82 disponibles son específicas de una determinada pila de Bluetooth, e.g., existen dos implementaciones JSR-82 libres para la pila

de Bluetooth de Windows XP Service Pack 2: 1) BlueCove[3] y 2) BlueSock[4].

Ventajas

Las ventajas de la solución basada en JSR-82 radican en la automatización del decubrimiento y de la conexión de dispositivos, así como en la portabilidad y robustez de esta API, como se describe a continuación:

- **Automatización:** el software basado en JSR-82 facilita 1) el descubrimiento de dispositivos, 2) la creación automática de una conexión entre los dispositivos autorizados y 3) la transferencia de datos necesarios. Este proceso no requiere ninguna intervención del usuario.
- **API robusta:** la solución basada en JSR-82 puede 1) manejar una lista de dispositivos, 2) realizar una búsqueda en ella y después 3) elegir un dispositivo para la transferencia de datos.
- **Portabilidad:** al utilizar el API JSR-82, una gran cantidad de código puede ser compartido entre las aplicaciones de J2SE y Java ME.

Desventajas

Es importante recordar que el software Bluetooth no implementa todas las características del API JSR-82. BlueCove, que utiliza el software Bluetooth de Windows XP Service Pack 2, permite solamente comunicaciones mediante el protocolo RFCOMM. Además, BlueCove carece de soporte para el protocolo L2CAP, lo cual representa una restricción para las aplicaciones que requieren retransmisión de datos.

BlueCove

El módulo de `Ad hoc Creator` utiliza, del lado de la PC, la implementación JSR-82 BlueCove cuya fase de instalación se menciona a continuación:

1. Descargar la distribución de BlueCove[5]
2. Descomprimir en cualquier carpeta temporal el archivo ZIP descargado.
3. Copiar el archivo *intelbth.dll* a un lugar donde el software pueda encontrarlo, e.g., la carpeta Windows/System32 o cualquier otra carpeta que esté declarada en las variables de entorno.
4. Copiar el archivo *Bluecove.jar* a una carpeta y agregar la ruta de acceso correspondiente al *classpath*.

[3]http://bluecove.sourceforge.net/
[4]https://bluesock.dev.java.net/
[5]http://sourceforge.net/projects/bluecove/

Capítulo 4

Implementación de *Ad hoc creator*

En este capítulo, primeramente se presenta los lineamientos que se siguieron para la selección de la plataforma empleada en la implementación de aplicaciones Bluetooth (sección 4.1). A continuación, se describe los pasos necesarios para desarrollar aplicaciones Java ME (*MIDlets*) las cuales están especialmente destinadas a ejecutarse en dispositivos móviles (sección 4.2). Posteriormente, se detalla la implementación de dos aplicaciones que permiten la explotación de una red *ad hoc* Bluetooth (sección 4.3). Estas aplicaciones constituyen la base para implementar el módulo `Ad hoc Creator` del sistema SEDINU, el cual se describe al final del presente capítulo (sección 4.4).

4.1 Selección de la plataforma

Java [Feng and Zhu, 2001] es un lenguaje de programación orientado a objetos, que fue desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de la sintaxis de C y C++, pero tiene un modelo de objetos más simple que elimina algunas construcciones de bajo nivel (e.g., la manipulación directa de punteros o de la memoria) que suelen inducir a muchos errores por parte de los programadores no experimentados. Java tiene las siguientes características:

- **Simple**: si bien Java es bastante parecido al lenguaje de programación C++, han sido eliminadas algunas complejidades con el fin de facilitar y reducir el costo del desarrollo de software. Particularmente, Java no contempla la manipulación de punteros, la sobrecarga de operadores ni la herencia múltiple. Sin embargo, Java dispone de un sistema automático de asignación y liberación de memoria (recolector de basura) que elimina las instancias de objetos que han dejado de ser utilizadas en un programa. De esta manera, el programador se despreocupa de la destrucción de objetos y evita la sobrecarga de la memoria RAM;
- **Orientado a objetos**: los objetos Java agrupan en estructuras encapsuladas, tanto sus datos, como los métodos (o funciones) que manipulan estos datos, lo cual facilita su reutilización;
- **Distribuido**: Java proporciona una colección de clases (e.g., para abrir *sockets*, establecer y aceptar conexiones con servidores o clientes remotos) que facilitan el

desarrollo de aplicaciones distribuidas;

- **Interpretado**: el compilador Java traduce cada archivo fuente de clases en código de bytes (*bytecode*), el cual puede ser interpretado por todas las máquinas que den soporte a una Máquina Virtual Java. El *bytecode* no es específico de una máquina determinada, por lo que no se compila ni enlaza como en el ciclo clásico de compilación, sino que se interpreta;

- **Sólido**: Java no permite escribir en áreas arbitrarias de la memoria ni realizar operaciones que corrompan el código. Java fue diseñado para crear software altamente confiable, por lo que realiza numerosas comprobaciones en tiempo de ejecución;

- **Seguro**: cuando se ejecuta código Java, la máquina virtual realiza comprobaciones de seguridad. Incluso, el diseño del propio lenguaje carece de características potencialmente inseguras, e.g., la manipulación de punteros;

- **Arquitectura neutral**: Java soporta aplicaciones que se ejecutan en los más variados entornos de red (desde Unix a Windows NT, pasando por Mac y estaciones de trabajo) sobre arquitecturas distintas y sistemas operativos diversos. Para acomodar requisitos de ejecución tan variados, el compilador de Java genera *bytecodes*, diseñados para transportar el código, de manera eficiente, a múltiples plataformas de hardware y software.

- **Portable**: los programas Java son iguales en todas las plataformas, lo que refleja parte de su portabilidad. Además, Java especifica el tamaño de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos;

- **Alto rendimiento**: al ser código interpretado, la ejecución no es tan rápida como el código compilado para una plataforma particular. En consecuencia, cuando se necesita capacidades de procesamiento intensivas, puede utilizarse llamadas a código nativo para proporcionar un mayor grado de eficiencia;

- **Multihilos**: Java soporta la sincronización de múltiples hilos de ejecución (*multithreading*) a nivel de lenguaje, que son especialmente útiles en la creación de aplicaciones distribuidas. Así, mientras un hilo se encarga de la comunicación, otro puede interactuar con el usuario, mientras otro presenta una animación en pantalla y otro realiza cálculos;

- **Dinámico**: el sistema de ejecución en tiempo real de Java es dinámico en la fase de enlazado, ya que las clases se ligan a medida que se requieren. Bajo demanda, se puede enlazar nuevos módulos de código procedentes de fuentes muy variadas, incluso de la red.

El diseño de Java, su robustez, el respaldo de la industria y su fácil portabilidad han hecho de Java uno de los lenguajes con un mayor crecimiento y amplitud de uso en distintos ámbitos de la industria de la informática. Por esta razón, se propuso la especificación Java ME, una versión reducida y altamente optimizada de la plataforma Java, que fue especialmente desarrollada para el mercado de los dispositivos móviles.

4.1.1 ¿Qué es Java ME?

Java ME [Haque and O'Connor, 2002] es una de las tres ediciones de la plataforma Java, diseñada específicamente para dispositivos móviles, e.g., teléfonos celulares y PDAs. Las otras dos ediciones son Java Enterprise Edition (Java EE) para servidores y computadoras empresariales y Java Platform Standard Edition (Java SE) para computadoras de escritorio. Una tecnología relacionada es Java Card, cuya especificación permite que la tecnología Java sea implantada sobre tarjetas inteligentes y otros dispositivos de memoria más limitada que la de un dispositivo móvil. Estas ediciones son necesarias para adaptar la tecnología Java a diferentes ámbitos de la industria de la computación actual. La Figura 4.1 ilustra las ediciones de la plataforma Java y sus mercados de destino.

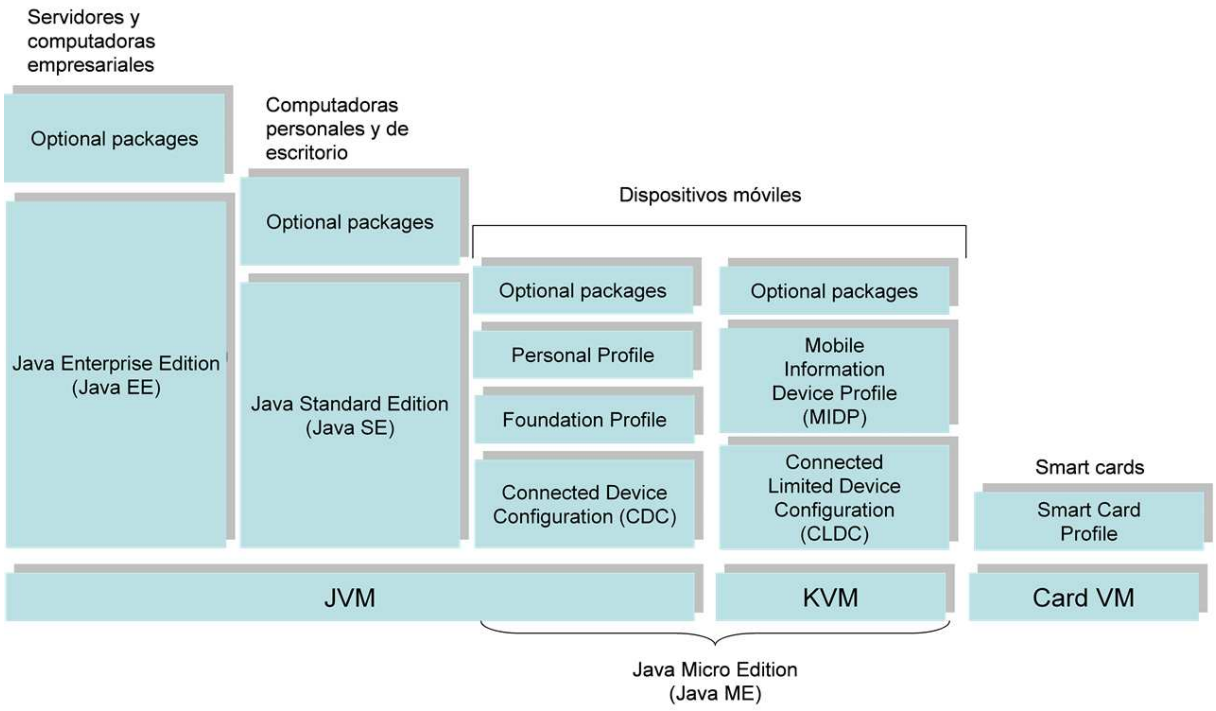

Figura 4.1: Plataformas Java

La plataforma Java ME otorga los beneficios de la tecnología Java (e.g., portabilidad de código, paradigma orientado a objetos y un ciclo de desarrollo rápido) al desarrollo de aplicaciones para dispositivos móviles. El objetivo principal de Java ME es permitir a los dispositivos descargar, de forma dinámica, aplicaciones que maximicen las capacidades nativas de cada dispositivo.

La arquitectura Java ME define configuraciones, perfiles y paquetes opcionales para facilitar la modularidad y la personalización. La Figura 4.2 muestra la relación de alto nivel entre las capas de la arquitectura de Java ME.

Figura 4.2: Capas de la arquitectura de Java ME

Configuraciones de Java ME

Una Máquina Virtual Java (MVJ) interpreta los *bytecodes* que son generados cuando se compilan los programas Java. De esta manera, un programa Java puede ejecutarse en cualquier dispositivo que tenga una MVJ adecuada y un conjunto de bibliotecas de clases Java.

Las configuraciones de Java ME comprenden una MVJ y un conjunto mínimo de bibliotecas de clases [Juntao Yuan, 2003]. La MVJ normalmente se ejecuta sobre un sistema operativo anfitrión, que es parte del software del dispositivo destino. La configuración define dos características: a) la funcionalidad mínima para una determinada categoría o grupo de dispositivos y b) las capacidades y los requisitos mínimos para una MVJ y las bibliotecas de clases disponibles en todos los dispositivos de la misma categoría o grupo. En la actualidad, existen dos configuraciones de Java ME: *Connected Limited Device Configuration* (CLDC) y *Connected Device Configuration* (CDC), las cuales se detallan a continuación:

- ***Connected, Limited Device Configuration***: se centra en dispositivos de bajo consumo tales como agendas electrónicas, teléfonos móviles y buscapersonas. Los dispositivos CLDC: a) tienen procesadores lentos y memoria limitada, b) funcionan con baterías y c) soportan solamente conexiones de red intermitentes. Una aplicación CLDC generalmente incluye una Kilobyte Virtual Machine (KVM)[1] la cual está especialmente diseñada para dispositivos de memoria limitada;

- ***Connected Device Configuration***: se centra en dispositivos de alto consumo que tienen más memoria, procesadores más rápidos y mayor ancho de banda de red. Algunos ejemplos típicos de dispositivos CDC son los decodificadores de televisión y los comunicadores de gama alta. La configuración CDC incluye: 1) una máquina virtual que se ajusta plenamente a la MVJ y 2) un subconjunto mucho más grande de la plataforma Java SE que la configuración CLDC.

[1] Recibe su nombre debido a su pequeña memoria (del orden de kilobytes)

Perfiles de Java ME

Las configuraciones de Java ME no suelen proporcionar una solución completa. Los perfiles [Huang and Rudolph, 2007] agregan las APIs necesarias para completar un entorno de ejecución completamente funcional para una clase de dispositivos. Las configuraciones deben combinarse con los perfiles que definen las APIs de alto nivel para proporcionar las capacidades requeridas por un mercado o industria específica.

Un dispositivo puede soportar varios perfiles. Se debe aclarar que los perfiles de Bluetooth, que han sido definidos en el capítulo 2, no deben confundirse con los perfiles de Java ME que se describen en la presente sección. Los dos tipos de perfiles no están relacionados. Un perfil Bluetooth se refiere a un conjunto de funcionalidades de los protocolos de Bluetooth para un caso de uso particular, en tanto que un perfil de Java ME es un conjunto de APIs que amplían la funcionalidad de una configuración Java ME. Algunos ejemplos de perfiles Java ME son *Mobile Information Device Profile* (MIDP), *Foundation Profile* (FP) y *Personal Profile* (PP), los cuales se describen a continuación:

- ***Mobile Information Device Profile***: es el primer perfil que está diseñado para teléfonos móviles, buscapersonas y agendas electrónicas. El perfil MIDP y la configuración CLDC ofrecen conjuntamente funcionalidades fundamentales, e.g., interfaz de usuario, capacidad de red y almacenamiento persistente. El perfil MIDP proporciona un ambiente Java de ejecución completo para dispositivos móviles. Una aplicación MIDP, llamada *MIDlet*, es tanto una clase definida en MIDP como la superclase de todas las aplicaciones MIDP.

- ***Foundation Profile***: es el perfil de más bajo nivel de la configuración CDC. Otros perfiles se pueden añadir en la parte superior, según sea necesario, para proporcionar funcionalidad a una aplicación. El perfil FP está diseñado para dispositivos embebidos sin interfaz de usuario, pero con capacidad de red.

- ***Personal Profile***: es empleado en dispositivos, tales como agendas electrónicas, comunicadores y consolas de juegos, que requieren una interfaz de usuario y un soporte de Internet. El perfil PP sustituye a la tecnología *PersonalJava* y ofrece a las aplicaciones una ruta de migración clara hacia la plataforma Java ME.

  Además existe un perfil, llamado *Personal Basis Profile* (PBP), que es un subconjunto de PP, dirigido a dispositivos que requieren sólo un nivel básico de presentación gráfica (e.g., decodificadores de televisión).

Paquetes opcionales de Java ME

Una gran variedad de dispositivos Java ME incluyen tecnologías adicionales, tales como la tecnología inalámbrica Bluetooth, multimedia, mensajes inalámbricos y la conectividad con bases de datos. Los paquetes opcionales se crearon para emplear estas tecnologías mediante las APIs estándares de Java. Los fabricantes de dispositivos pueden incluir estos paquetes opcionales, según sea necesario, para aprovechar plenamente las características de cada tipo de dispositivo.

Además de las configuraciones, perfiles y paquetes opcionales, los fabricantes de dispositivos definen nuevas clases Java para utilizar las características específicas de cada tipo de dispositivo. Existen dos tipos de clases: 1) las de licencia abierta (LOC[2] por sus siglas en inglés) y 2) las de licencia cerrada. LOC define las clases disponibles para cualquier desarrollador. Por el contrario, las clases de licencia cerrada definen clases disponibles sólo para el fabricante del dispositivo. Los programas que utilizan estas clases pueden no ser portables entre dispositivos que tienen la misma configuración y los mismos perfiles.

4.1.2 ¿Por qué la tecnología Java para dispositivos Bluetooth?

La utilización de la tecnología inalámbrica Bluetooth, varía de persona a persona. Dos personas que tienen el mismo modelo de teléfono habilitado con Bluetooth pueden utilizarlo para finalidades distintas. Por ejemplo, una persona puede descargar juegos de video en el teléfono o utilizarlo como control remoto de televisión, mientras que otra persona puede utilizar el mismo modelo de teléfono para abrir las puertas de su automóvil, operar aparatos electrodomésticos o abrir y cerrar las puertas de su *garaje*. Una forma de que ambas personas alcancen sus objetivos consiste en descargar aplicaciones Bluetooth en sus dispositivos. Para realizar la descarga de aplicaciones, se necesita una API estándar que permita a los programadores escribir aplicaciones Bluetooth que funcionen en diversas plataformas de hardware. El lenguaje Java es la elección ideal para definir esta API estándar. Una API Java permite que las aplicaciones se ejecuten en diferentes tipos de hardware, sistemas operativos y clases de dispositivos. En resumen, además de la portabilidad, el lenguaje Java proporciona otros beneficios:

- rápido desarrollo de aplicaciones gracias a la abstracción y a la programación de alto nivel, proporcionadas por un lenguaje de programación orientado a objetos;
- posibilidad de ampliar dinámicamente la funcionalidad de un programa durante su ejecución, mediante la carga de clases en tiempo de ejecución;
- verificación de archivos de clases y características de seguridad que proporcionan protección contra aplicaciones maliciosas, la cual es necesaria para permitir a los dispositivos la descarga de aplicaciones;
- estándares con mejores interfaces de usuario que soporten interacción sofisticada;
- gran comunidad de desarrolladores, ya que el número de personas que programan en lenguaje Java está en continuo crecimiento.

Por estas razones, surgió la necesidad de desarrollar una API estándar para la tecnología inalámbrica Bluetooth, utilizando el lenguaje de programación Java. Esta estandarización dio lugar al conjunto de APIs denominado JABWT (*Java APIs for Bluetooth Wireless Technology*), el cual complementa las tecnologías existentes en lugar de reemplazarlas. JABWT se construye en la parte superior de la ya establecida y ampliamente utilizada pila de protocolos Bluetooth.

[2]Licensee Open Classes

4.1.3 JABWT

En esta sección, se describe la motivación y los objetivos de JABWT, con la finalidad de conducir a una mejor comprensión de las características y capacidades de este conjunto de APIs.

Objetivos de JABWT

JABWT garantiza la compatibilidad entre dispositivos Bluetooth de diferentes fabricantes, pero no ofrece una API estándar específica para utilizar pilas Bluetooth. Sin embargo, JABWT define el primer conjunto estándar de APIs para desarrollar aplicaciones Bluetooth. Los objetivos de JABWT son: 1) reducir al mínimo el número de clases[3], 2) mantener las APIs simples y fáciles de aprender y programar y 3) permitir el acceso a la tecnología inalámbrica Bluetooth mediante el lenguaje Java. Las abstracciones y facilidades de programación del lenguaje Java ofrecen las herramientas necesarias para desarrollar programas complejos.

Dispositivos JABWT

JABWT está dirigido principalmente a dispositivos móviles que utilizan baterías y que están limitados en poder de procesamiento y en memoria. Estos dispositivos pueden ser fabricados en grandes cantidades, sin embargo el objetivo principal de los fabricantes es construir dispositivos de bajo costo y consumo de energía. Algunos artefactos, e.g., un automóvil, una laptop o un punto de acceso, no son dispositivos Java ME, ya que funcionan con Java SE o CDC. Algunos fabricantes de estos artefactos, ya están incorporando JABWT en sus nuevos diseños.

Conservación de los perfiles Bluetooth

La idea inicial del grupo de expertos de JSR-82 era definir una API basada en los perfiles Bluetooth, pero se dio cuenta de que el número de perfiles estaba en constante crecimiento y que no sería posible mantener al día la especificación JABWT. Por lo tanto, el grupo de expertos de JSR-82 decidió proporcionar soporte solo a los protocolos y perfiles básicos, en lugar de introducir nuevos elementos por cada perfil Bluetooth. La intención del diseño JABWT es permitir que los nuevos perfiles Bluetooth sean construidos encima de esta API mediante el lenguaje de programación Java. En particular, los perfiles Bluetooth están siendo desarrollados sobre los protocolos de comunicación OBEX, RFCOMM y L2CAP. Por esta razón, estos protocolos han sido incorporados en JABWT. Actualmente, se escriben perfiles Bluetooth en lenguaje Java, con el fin de promover la portabilidad entre los principales sistemas operativos y pilas de protocolo Bluetooth.

Además de la API para acceder a los protocolos, existen APIs para utilizar algunos perfiles Bluetooth. Particularmente, JABWT maneja los perfiles GAP, SDAP, SPP, y GOEP. La información detallada sobre los perfiles Bluetooth y sus relaciones con los protocolos OBEX, RFCOMM y L2CAP está definida en la especificación de cada perfil.

[3]el número total de clases es 21

4.1.4 Casos de uso de JABWT

Cualquier tecnología mejora cuando se crean más aplicaciones para ella. Las APIs estándares (e.g., la API JABWT) fomentan un ambiente para crear una variedad de aplicaciones que soporta diversos casos de uso, como los que se mencionan a continuación.

Caso de uso #1: Creación de redes *peer to peer*

Las redes *peer to peer* pueden ser definidas e interpretadas de muchas maneras. Para efectos de esta explicación, una red *peer to peer* se define como una red *ad hoc* entre dos o más dispositivos en la que cada dispositivo puede ser tanto un servidor como un cliente. JABWT soporta redes *peer to peer* que emplean la tecnología inalámbrica Bluetooth. Un ejemplo de una aplicación *peer to peer* es un juego entre dos o más dispositivos conectados a través de comunicación Bluetooth.

Los dispositivos (fijos y móviles) involucrados pueden pertenecer a clases totalmente distintas, como un teléfono celular y un receptor GPS, que utilizan hardware y sistemas operativos diferentes. Si estos dispositivos disponen de JABWT, el software del juego puede estar escrito en lenguaje Java y ejecutarse en dichos dispositivos. Además, la propiedad de independencia de dispositivo, que caracteriza a estas aplicaciones, permite compartir y descargar estos juegos en dispositivos heterogéneos.

Caso de uso #2: Venta de software por quiosco

No es práctico para un quiosco que vende software, tener una aplicación ejecutable por cada tipo de dispositivo Bluetooth. Mediante JABWT, una aplicación puede ser escrita una vez, comprada y ejecutada en todos los dispositivos Bluetooth que implementan JABWT. Esta capacidad permite a los establecimientos, tales como aeropuertos, estaciones de metro y centros comerciales, tener aplicaciones personalizadas que funcionen adecuadamente en su entorno. Los dispositivos Bluetooth habilitados con JABWT pueden descargar estas aplicaciones desde quioscos.

Caso de uso #3: Compra de productos y aplicaciones Bluetooth por medio de máquinas expendedoras

Otro ejemplo de los beneficios de la API JABWT se muestra en un escenario en el que las personas compran o descargan aplicaciones en su dispositivo Bluetooth, por medio del mismo dispositivo que utilizan para comprar un refresco en una máquina expendedora. La API permite que las aplicaciones se ejecuten en plataformas Bluetooth diferentes. De esta manera, las máquinas expendedoras venden estas aplicaciones y las transfieren a través de una red *ad hoc* Bluetooth a los dispositivos móviles de los clientes. Un fabricante de juegos podría comprar un espacio publicitario en las máquinas expendedoras para promocionar un juego de muestra. Los clientes que compran un refresco podrían seleccionar la opción de descargar en su dispositivo el juego de muestra, el cual puede ser mejorado más tarde (e.g., aumento de niveles) cuando se compre.

4.1.5 Ventajas y desventajas de redes *ad hoc* Bluetooth

Las redes *ad hoc* pueden formarse mediante diferentes tecnologías inalámbricas, pero Bluetooth se ha vuelto popular en la creación de dichas redes por las numerosas ventajas que ofrece sobre las tecnologías competidoras.

La razón principal por la que la tecnología Bluetooth es ampliamente utilizada para la creación de redes *ad hoc*, se debe a que está diseñada específicamente para este fin. Por el contrario, el estándar 802.11 (WiFi) tiene como objetivo proporcionar un reemplazo de las infraestructuras de redes cableadas. Mientras que otras redes pueden configurarse para operar de una manera *ad hoc*, sin ningún dispositivo central de "control", Bluetooth ofrece funcionalidades de configuración automática de red, autenticación y descubrimiento de servicios. De esta manera, la creación de una red *ad hoc* basada en Bluetooth es sencilla para el usuario promedio, quien probablemente no está familiarizado con protocolos de red y configuraciones de *routers*.

Bluetooth está diseñado para consumir la menor energía posible en el dispositivo más pequeño posible. Se puede dar por hecho que si los dispositivos utilizan cualquier tecnología inalámbrica (e.g., WiFi, Bluetooth o Infrarrojos) entonces son suministrados de energía por una batería. WiFi y otras tecnologías consumen mucha energía en comparación con Bluetooth, la cual funciona con solo 2.5mW de potencia, frente a 100mW que utiliza WiFi.

Además, los usuarios de la tecnología Bluetooth no se enfrentan con obstáculos legales al utilizar las frecuencias de radio requeridas por los dispositivos. Como Bluetooth utiliza la banda libre ISM a 2.4 GHz, ofrece a los usuarios la ventaja de no tener que obtener licencias para su uso.

Lamentablemente, Bluetooth tiene algunas limitaciones, respecto al rango de alcance y al ancho de banda. Esta tecnología ofrece normalmente un rango de alcance de 10 metros, debido a que se planeó que sustituiría el cable y crearía redes *ad hoc*. En consecuencia, era poco probable que, en la práctica, fuera necesaria una gran distancia entre los nodos. Cabe señalar que la comunicación entre dispositivos a distancia es posible si existen otros dispositivos capaces de retransmitir los mensajes. Aunque algunas versiones de Bluetooth soportan un rango de alcance mucho mayor (hasta 100 metros), estas versiones consumen mucha más energía (hasta 100mW).

La otra desventaja de Bluetooth se refiere al reducido ancho de banda que maneja. Esta tecnología es capaz de transmitir y recibir datos hasta 1 Mbps, lo cual se queda muy atrás comparado con los 54 Mbps ofrecidos por las redes WiFi. De esta manera, en el caso de requerir la transferencia de grandes archivos, e.g., el flujo de audio, el usuario debería optar por el estándar 802.11 (WiFi).

4.2 Programación en Java ME

En esta sección, se presenta los requerimientos necesarios para implementar aplicaciones Java ME. La plataforma Java ME proporciona al desarrollador de aplicaciones los medios necesarios para construir aplicaciones Java destinadas a ejecutarse en dispositivos de recursos limitados, como teléfonos móviles y PDAs.

4.2.1 Requerimientos de los dispositivos Java ME

JABWT no pretende ser una solución completa. Se trata de una API opcional, basada en el marco GCF (*Generic Connection Framework*), que extiende la plataforma Java con soporte para la tecnología inalámbrica Bluetooth. JABWT se enfoca en las siguientes características generales de los dispositivos Java ME:

- 512K de memoria mínima para la plataforma Java (ROM/Flash y RAM); los requerimientos de memoria de la aplicación son adicionales;
- pila y hardware de comunicación Bluetooth;
- aplicación compatible con la configuración CLDC (*Connected, Limited Device Configuration*) o un superconjunto de APIs CLDC, tales como CDC o cualquier plataforma Java que contenga la API JSR-197.

Los requerimientos de hardware de los diferentes perfiles y configuraciones de Java ME pueden obtenerse directamente de las especificaciones correspondientes[4].

4.2.2 MIDlets

Como se mencionó anteriormente, el perfil MIDP (*Mobile Information Device Profile*) se apoya en la configuración CLDC para ofrecer las clases y los paquetes necesarios para el desarrollo de aplicaciones. El perfil MIDP está orientado principalmente a teléfonos móviles, aunque también existe una implementación para PalmOS y PocketPC.

Se denomina *MIDlet* a una aplicación Java implementada mediante la especificación del perfil MIDP, i.e., un *MIDlet* es una aplicación que puede utilizar la funcionalidad aportada por MIDP y CLDC. Un *MIDlet* está compuesto, al menos, por una clase principal que hereda directamente de la clase *javax.microedition.midlet.MIDlet*. Los métodos de esta clase permiten a la aplicación AMS (*Application Management Software*) crear, iniciar, pausar y destruir un *MIDlet*, como se describe a continuación.

Application Management Software

El término *Application Management Software* se refiere básicamente a una aplicación de software que viene con el perfil MIDP. La aplicación AMS también es conocida como *MIDlet Management Software* o *Java Application Manager* porque controla la instalación, la ejecución y la eliminación de *MIDlets*. AMS puede variar de un dispositivo a otro, pero los servicios básicos que debe proporcionar son los siguientes:

- Ofrece funciones a los usuarios para instalar y desinstalar *MIDlets* desde sus dispositivos inalámbricos, ya sea mediante un cable serie conectado a una PC o por medio de Internet;

[4] que están disponibles en www.jcp.org

- Provee un entorno de ejecución a los *MIDlets* ya que, después de haberlos inicializado, AMS pone a su disposición los recursos del sistema, tales como la configuración CLDC, el perfil MIDP y la máquina virtual KVM; en tiempo de ejecución, AMS también pone a disposición clases, archivos de recursos y sus descriptores;

- Maneja todos los errores que se producen durante la instalación y la ejecución de los *MIDlets* para evitar la falla del sistema.

Ciclo de vida de un *MIDlet*

La ejecución de un *MIDlet* incluye tres estados válidos: `activo`, `pausado` y `destruido`. Las transiciones entre los diferentes estados son controladas por la aplicación AMS (*Application Management Software*) mediante los métodos `startapp ()`, `pauseApp ()` y `destroyApp ()` implementados por el *MIDlet*. La Figura 4.3 ilustra las transiciónes entre los tres estados posibles de un *MIDlet* a través de la invocación de dichos métodos.

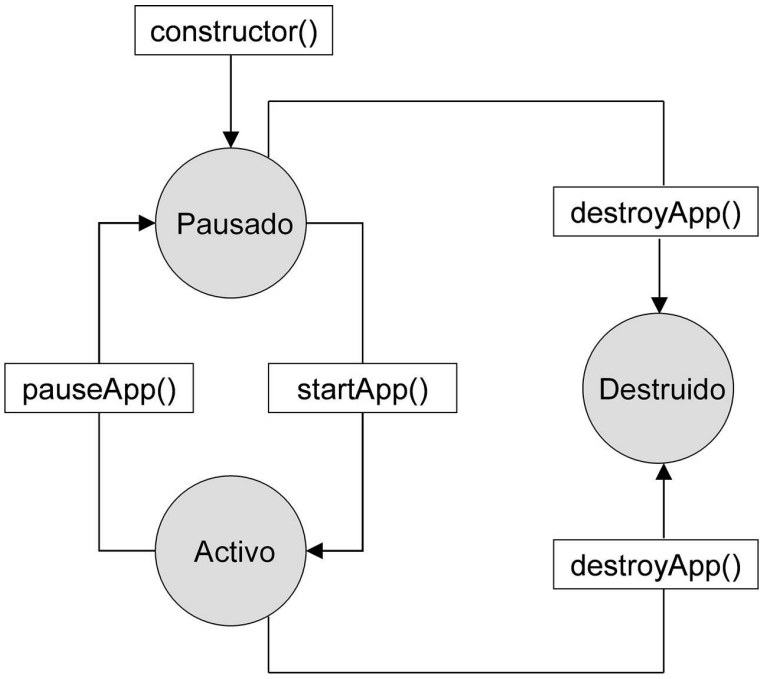

Figura 4.3: Ciclo de vida de un *MIDlet*

Cuando un *MIDlet* está listo para ejecutarse, AMS primero crea una instancia del *MIDlet* mediante su constructor público sin argumentos. En respuesta, el *MIDlet* entra al estado `pausado`.

Después, AMS invoca al método `startapp ()`, con el fin de que el *MIDlet* entre al estado `activo`, en el cual adquiere los recursos necesarios para prestar sus servicios. En este estado, el *MIDlet* está en ejecución.

Cuando AMS determina que el *MIDlet* ya no necesita estar en el estado `activo`, invoca al método `pauseApp ()`. Como resultado, el *MIDlet* se detiene para entrar al estado

`pausado`, en el cual libera algunos de los recursos adquiridos y se vuelve inactivo. El *MIDlet* puede regresar al estado `activo` siempre que AMS llame al método `startapp ()`.

Por último, cuando AMS decide que ya no necesita el *MIDlet* o requiere liberar espacio en la memoria para albergar a un programa de mayor prioridad, marca al *MIDlet* como un candidato a ser destruido, mediante el método `destroyApp ()`. El *MIDlet* entra finalmente al estado `destruido`, en el cual libera todos los recursos y guarda cualquier dato persistente antes de terminar.

Empaquetamiento de un *MIDlet*

Los *MIDlets* necesitan ser empaquetados antes de ser instalados en el dispositivo destino. Para llevar a cabo la fase de empaquetamiento, se utiliza un archivo .JAR (*Java ARchive*) donde se almacena el *MIDlet* principal y todas aquellas clases, imágenes o archivos que pueden ser necesarios en tiempo de ejecución. El archivo .JAR (particularmente, el archivo *manifest*) también incluye información referente al contenido del fichero .JAR. Esta misma información también está incluida en el fichero .JAD (*Java Application Descriptor*).

Las clases del *MIDlet*, que están empaquetadas en el archivo .JAR, deben estar compiladas en archivos *.class* y verificadas, antes de su ejecución en un dispositivo, para evitar que realicen alguna operación no permitida. De hecho, la única operación "no permitida" que autoriza la API de MIDP es el método `exit()` de las clases *System* o *Runtime*, las cuales requieren incluir la excepción *SecurityException* para poder ser utilizadas. Esta verificación debe llevarse a cabo manualmente, ya que si se agrega a la máquina virtual puede resultar muy costosa en memoria debido a las limitaciones de los dispositivos móviles. Por esta razón, se debe verificar los archivos .JAR que se instalan en dichos dispositivos, ya que si provienen de una fuente no fiable podrían contener código malicioso que salga del control de la máquina virtual.

Un archivo .JAR puede contener varios *MIDlets*. A este conjunto de *MIDlets* se le denomina *MIDlet suite*, el cual facilita el uso compartido de recursos. Esta opción es aconsejable para minimizar el uso de los recursos del dispositivo y para lograr una mayor reutilización de los componentes del *MIDlet suite*.

Estructura de un *MIDlet*

Un *MIDlet* tiene que extender la clase *javax.microedition.midlet.MIDlet*, la cual contiene todos los métodos necesarios para controlar su ciclo de vida. Un *MIDlet* debe tener un constructor público por defecto, i.e., un constructor sin argumentos. El programador puede implementar este constructor, si hace falta inicializar algunos atributos del *MIDlet*, o bien el compilador Java puede implementarlo si no encuentra ninguno. La estructura de un *MIDlet* se observa en la Figura 4.4.

El *MIDlet* empieza en el estado `pausado`, después se carga en memoria y se ejecuta el constructor; si hasta este punto no ha habido ningún problema, entonces el *MIDlet* pasará al estado `activo` por medio del método `startApp()`. A continuación, se detallan los tres métodos principales de la clase *javax.microedition.midlet.MIDlet*:

```
//clase MIDlet
import javax.microedition.midlet.*;

//La clase debe heredar de la clase MIDlet
public class Ejemplo1 extends MIDlet {

   //Constructor
   public Ejemplo1( ) {
   }

   //Método que se llama cuando cambia un MIDlet de Pausado a
Activo
   protected void startApp( ) {
   }

   //Método que se llama cuando cambia un MIDlet Activo a
Pausado
   protected void pauseApp( ) {
   }

   //Método que se llama cuando se destruye el MIDlet
   protected void destroyApp(boolean incondicional) {
   }
}
```

Figura 4.4: Estructura de un *MIDlet*

- **Método *protected abstract void startApp ()***: es un método abstracto de la clase *javax.microedition.midlet.MIDlet*, por lo que tiene que ser implementado en el *MIDlet*. Debido a que es un método protegido (*protected*) solo puede ser utilizado, si se importa el paquete *javax.microedition.midlet.MIDlet*. Algunas veces, este método suele ser redefinido como público (*public*) para que pueda ser invocado desde otro lugar. Si no hay ningún problema (*exception* o *error*) el *MIDlet* se estará ejecutando mientras no pase al estado `pausado` o `destruido`. En cualquier momento, el entorno de ejecución puede forzar al *MIDlet* a pasar al estado `pausado`. Por ejemplo, cuando se recibe una llamada en un teléfono celular, el dispositivo necesita los recursos "pantalla" y "teclado" para tratar esta llamada entrante; como resultado, se invoca al método `pauseApp()`.

- **Método *protected abstract void pauseApp ()***: la implementación de este método depende en gran medida del *MIDlet*, pero básicamente guarda su estado actual y libera los recursos que reservó. Durante la ejecución de este método, el *MIDlet* pasa al estado `pausado` a partir del cual su ejecución puede reanudarse en cualquier momento. En dicho estado, el *MIDlet* deja de tener acceso a la pantalla, pero sus hilos y sus temporizadores (*timers*) siguen activos. Cuando el *MIDlet* regresa al estado `activo` después de salir del estado `pausado`, AMS llama al método `startApp()`. Este método tiene que distinguir entre la primera vez que es invocado (al iniciarse el *MIDlet*) y cuando se llama después de una pausa. Si el entorno de ejecución o el usuario decide cerrar la aplicación, entonces AMS llamará al método `destroyApp()`.

- **Método *protected abstract void destroyApp (boolean unconditional)***: este método finaliza el *MIDlet*, el cual pasa al estado `destruido` en el que tiene que liberar todos los recursos empleados y guardar los datos persistentes. Este método puede ser invocado desde los estados `activo` y `pausado`. El *MIDlet* tiene la opción de no entrar al estado `destruido` lanzando la excepción *MIDletStateChangeException*, la cual solo es válida si el parámetro del método (*boolean unconditional*) es falso. Por el contrario, si es verdadero, el *MIDlet* terminará inevitablemente su ejecución. Si ocurre alguna excepción durante la ejecución de este método, el *MIDlet* la ignorará completamente y pasará al estado `destruido`.

4.3 Aplicaciones de prueba de redes *ad hoc*

En esta sección, se presenta dos aplicaciones Bluetooth: 1) una aplicación denominada Eco Cliente-Servidor que utiliza el perfil de puerto serie y 2) una aplicación de *Instant Messenger* que ofrece una interfaz de usuario. Estas aplicaciones pueden utilizarse tanto en PCs como en dispositivos móviles y sirven de base para la implementación del módulo `Ad hoc Creator`, el cual se describe en la sección 4.4. Como las aplicaciones de PC utilizan la API JSR-82 con J2SE, estas requieren que BlueCove (o una implementación similar de la API JSR-82) esté instalada previamente en la PC.

4.3.1 Aplicación Eco Cliente-Servidor Bluetooth

Esta aplicación Bluetooth consiste de un módulo servidor (aplicación Java SE) para una PC y un módulo cliente (aplicación *MIDlet* Java ME) para dispositivos móviles. La comunicación entre el cliente y el servidor Bluetooth se realiza a través del perfil de puerto serie (SPP), por lo cual se necesita una dirección URL de conexión. Esta URL consiste de un identificador de protocolo, el UUID (*Universally Unique IDentifier*) del servicio a ofrecer y otros atributos opcionales (e.g., seguridad habilitada). Dado que se trata de un servicio SPP, el identificador de protocolo es "btspp", mientras que el UUID se define como 1101.

Módulo servidor

Una vez que el módulo `servidor` está activo, este inicia un ciclo en espera de una nueva solicitud de conexión por parte de un cliente. Cuando recibe dicha solicitud, la comunicación entre el `cliente` y el `servidor` queda establecida. A continuación, el módulo `servidor` empieza a enviar una serie de *tokens* (e.g., 1, 3, 5, 7 y 255) al módulo `cliente`, el cual le responde transfiriéndole los mismos *tokens*. El módulo `servidor` se detiene cuando envía el último *token*, i.e., 255 al módulo `cliente` (ver Figura 4.5). Después de manejar esta sesión, el módulo `servidor` reinicia el ciclo en espera de una nueva conexión.

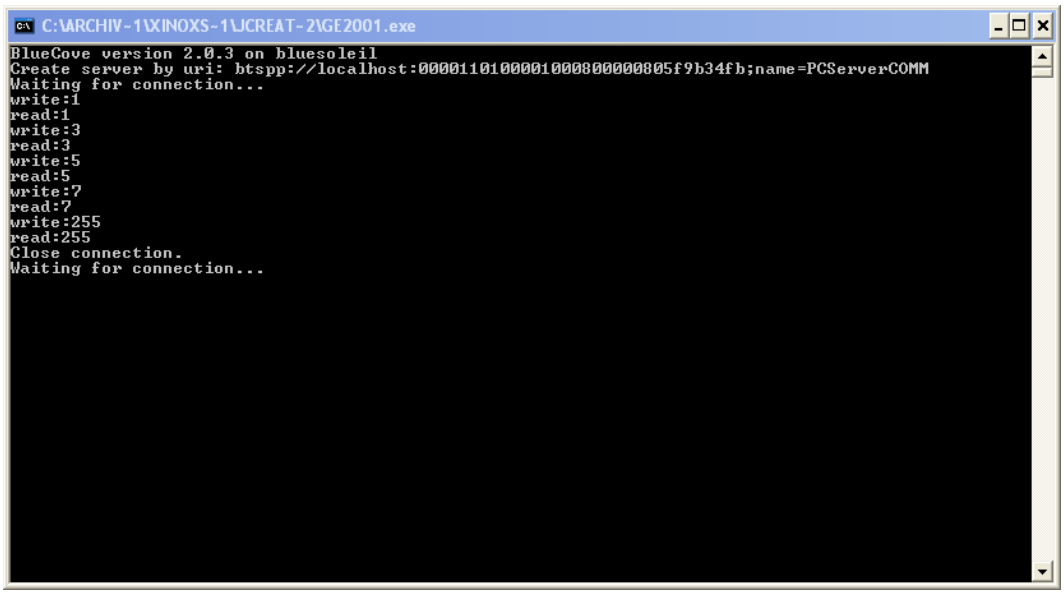

Figura 4.5: Módulo servidor de la aplicación Eco Cliente-Servidor Bluetooth

Módulo cliente

Cuando el módulo `cliente` inicia su ejecución, el área de información del dispositivo móvil muestra los datos referentes a la búsqueda de dispositivos. A continuación, se presenta una lista de los dispositivos Bluetooth que están cerca del dispositivo móvil que alberga al módulo `cliente`. A partir de esta lista, el usuario selecciona el dispositivo (en este caso, una PC) que actúa como `servidor`. Una vez que la conexión se estableció con éxito, el módulo `cliente` funciona como un eco, i.e., todo lo que se recibe del módulo `servidor` se lo reenvía (ver Figura 4.6).

El módulo `cliente` implementa tres operaciones diferentes: 1) la búsqueda de dispositivos, 2) el descubrimiento de servicios y 3) el manejo de *streams*. Las operaciones deben ser ejecutadas en orden de dependencia, como a continuación se menciona:

1. **Búsqueda de dispositivos**: se utiliza para buscar todos los dispositivos Bluetooth dentro del alcance del dispositivo que alberga al módulo `cliente`;

2. **Descubrimiento de servicios**: busca el servicio de perfil de puerto serie en el dispositivo seleccionado; el descubrimiento de servicios inicia después de que el usuario seleccionó un dispositivo;

3. **Manejo de *streams***: se comunica mediante una conexión Bluetooth con el módulo `servidor`, el cual se mantiene en un ciclo de espera hasta que encuentra la señal de parada, i.e., el *token* 255.

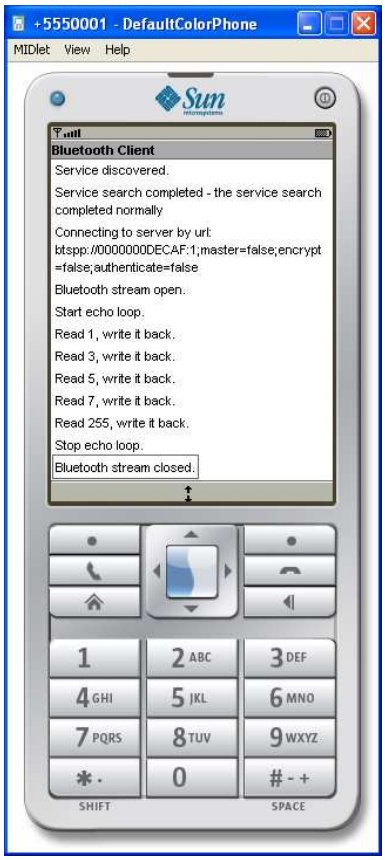

Figura 4.6: Módulo cliente de la aplicación Eco Cliente-Servidor Bluetooth

4.3.2 Aplicación *Instant Messenger*

Esta es una aplicación de *chat* descentralizada que opera sobre conexiones *peer to peer* Bluetooth. La aplicación *Instant Messenger* está inspirada en *Internet Relay Chat* (IRC). La sesión de *chat* comienza una vez que se inicia la aplicación. La interfaz de usuario tiene un área de entrada para desplegar los mensajes que se reciben de todos los usuarios y un área de salida para enviar mensajes a los demás usuarios.

Las versiones para PC y para dispositivos móviles presentan pequeñas diferencias. En la versión de PC, la aplicación se inicia por medio del método `main()` de Java, mientras que la versión para dispositivos móviles extiende la clase *MIDlet*. Las principales diferencias radican en la interfaz de usuario (ver Figura 4.7). La versión para PC utiliza la clase *Swing UI* (*User Interface*), mientras que la versión para dispositivos móviles utiliza la clase *LCDUI* (*Limited Capability Device User Interface*).

La aplicación *Instant Messenger* forma una red de usuarios a través de conexiones Bluetooth. Por cada charla se establece una conexión Bluetooth, i.e., no existe un servidor centralizado.

La aplicación tiene dos hilos: un hilo cliente y un hilo servidor. El hilo cliente, mejor conocido como hilo de descubrimiento, busca nuevos dispositivos. Se requiere un proceso de búsqueda por cada dispositivo encontrado. Si se descubre el servicio de *chat* en la

PC, se establecerá una conexión entre esta y el dispositivo móvil. Una vez finalizada esta sesión, el hilo servidor queda en espera de nuevas conexiones.

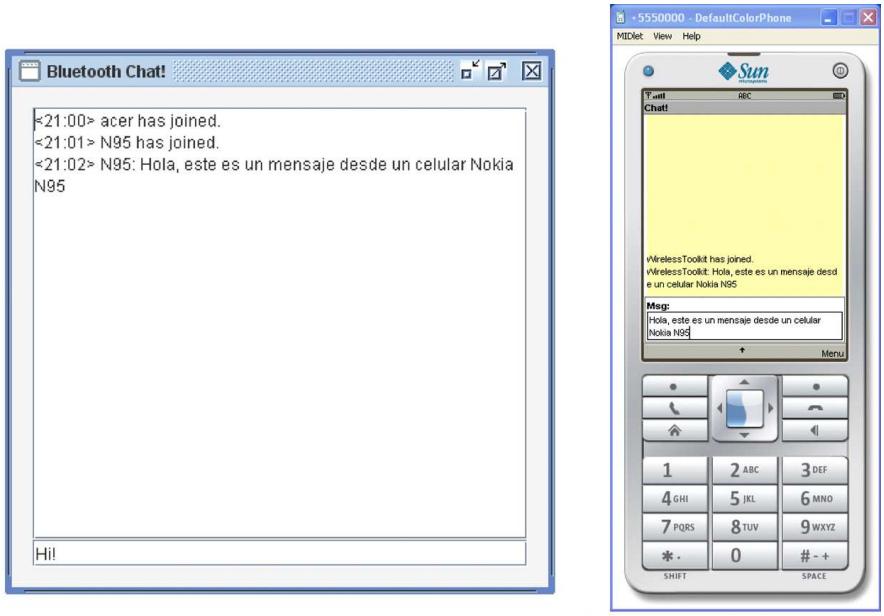

Figura 4.7: Interfaces de usuario de la aplicación *Instant Messenger* para PC y dispositivos móviles

A continuación se detallan los componentes de la aplicación *Instant Messenger*:

- **Protocolo de mensajes**: el protocolo de mensajes es muy sencillo. Las cadenas son codificadas mediante el método *DataOutputStream.writeUTF* y decodificadas por medio del método *DataInputStream.readUTF* para enviar y recibir datos, respectivamente.

- **Hilo servidor**: este hilo envía primeramente el nombre del usuario del *chat* a todos los hilos clientes conectados. Todas las cadenas transmitidas posteriormente son mensajes de un usuario dirigidos a los demás usuarios.

  Las dos primeras cadenas a recibir son la dirección Bluetooth del dispositivo y el nombre del usuario, el cual debe ser inferior a 8 caracteres. Todas las demás cadenas recibidas se refieren a los mensajes de *chat* de todos los usuarios.

- **Hilo cliente**: este hilo envía primeramente la dirección Bluetooth del dispositivo y el nombre del usuario. Todas las cadenas transferidas ulteriormente se refieren a los mensajes de *chat* del usuario asociado al hilo cliente.

  La primera cadena a recibir es el nombre del usuario. Todas las demás cadenas se refieren a los mensajes de *chat* de todos los usuarios.

Pruebas de la aplicación *Instant Messenger*

La Figura 4.8 muestra el proceso de conexión entre una PC y un dispositivo móvil para ejecutar conjuntamente la aplicación *Instant Messenger*. Esta figura ilustra el lado de la PC, donde se puede observar que se utiliza la implementación BlueCove versión 2.0.3 (sobre la pila Bluetooth de BlueSoleil) para establecer una conexión Bluetooth.

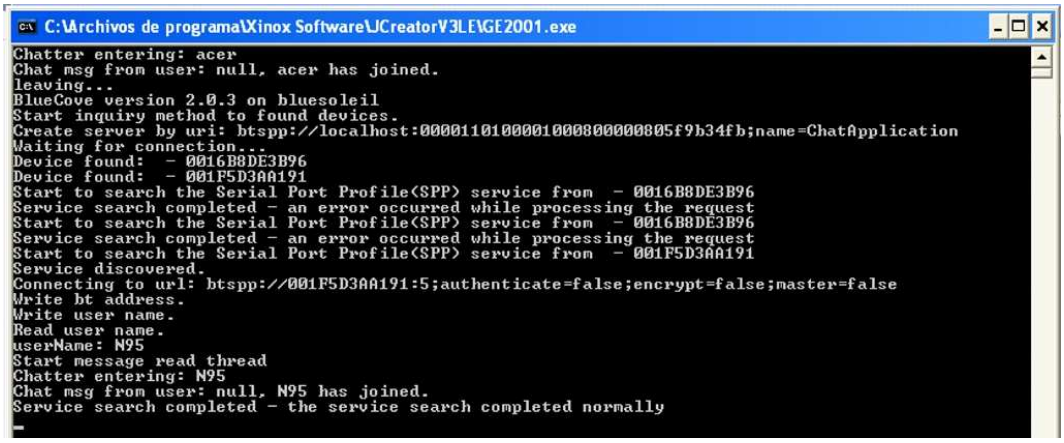

Figura 4.8: Aplicación *Instant Messenger* en una PC y en un dispositivo móvil

Primeramente, se inicia la búsqueda de dispositivos Bluetooth. A continuación, se abre una conexión de tipo servidor mediante una URI (*Uniform Resource Identifiers*) la cual está formada de una cadena de caracteres que identifica a un servicio de perfil SPP (*Service Port Profile*). La estructura de la URI que identifica a este servicio es la siguiente:

btspp://localhost: UUID;name=ChatApplication

Posteriormente, el hilo servidor espera una conexión. Por su parte, el hilo cliente muestra los dispositivos Bluetooth encontrados y, en cada uno de ellos, busca el servicio SPP. Cuando el hilo cliente descubre este servicio, establece una conexión con el hilo servidor. Después, este último envía, a todos los hilos cliente, la dirección Bluetooth del dispositivo y el nombre del usuario asociados a dicho hilo cliente. A partir de este momento, el envío de mensajes entre los usuarios implicados puede tomar lugar.

Problemas encontrados durante la implementación

Durante las pruebas realizadas a la aplicación *Instant Messenger*, se presentó un problema de despliegue en pantalla en uno de los dispositivos móviles de prueba, específicamente un Nokia N95.

Descripción:
Cuando se adjunta un objeto de la clase *CustomItem* a un formulario, el contenido de este objeto permanece en blanco, mientras el formulario no se haga visible. Si un formulario contiene un objeto *CustomItem*, este último se convierte en el contenido actual

visible. Sin embargo, el contenido del objeto *CustomItem* permanece en blanco, a pesar de que debería observarse su contenido.

Pruebas:

De acuerdo a las pruebas de despliegue llevadas a cabo en los telefonos celulares Sony Ericsson W300 y Nokia N90 S60 2da edición, se observó que el despliegue de los gráficos si funciona correctamente. Asimismo, los resultados de las pruebas realizadas en los dispositivos de la serie S60 de Nokia 3era edición (N95, N73, N82, E65) mostraron un despliegue blanco en la pantalla.

Hipótesis:

Parece ser que existe un problema de incompatibilidad con alguna función de despliegue de información en la pantalla. Los sistemas operativos de ambas series son diferentes (Symbian 8.1a para la serie S60 2da edición y Symbian 9.1 para la serie S60 3era edición).

Solución:

Actualmente no existe una solución. Sin embargo, el problema se puede evitar si se invoca al método `Display.getDisplay ()` por lo menos una vez durante el ciclo de vida del *MIDlet*, antes de crear el objeto *CustomItem* (e.g., en el inicio del constructor del *MIDlet*) como se muestra a continuación:

Paso #1: Se creó un método en la clase LCDUI:

```
public void appendItems()
{
 append(messageArea);
 append(field);
}
```

Paso #2: En el *MIDlet* se invocó a este método, antes de crear el objeto *CustomItem*:

```
ui.setCurrent();
ui.appendItems();
(Objeto CustomItem)
```

4.4 Implementación del módulo *Ad hoc Creator*

En esta sección, se describe la implementación del módulo `Ad hoc Creator` mediante dos aplicaciones de prueba: 1) AdhocCreatorServices y 2) AdhocCreatorNomadasMIDlet. La aplicación AdhocCreatorServices, se encarga de simular la creación de redes *ad hoc* entre el dispositivo móvil de un usuario nómada y el *host* de servicios de un área autónoma. Por su parte, la aplicación AdhocCreatorNomadasMIDlet simula la creación de redes *ad hoc* entre dos dispositivos móviles para facilitar la interacción y/o colaboración entre dos usuarios nómadas.

4.4.1 Programación mediante el protocolo OBEX

El protocolo IrOBEX (*Infrared Object Exchange Protocol*) fue definido por la organización IrDA (*Infrared Data Association*) como una alternativa al protocolo de transferencia de hipertexto (HTTP por sus siglas en inglés) para dispositivos que tienen menor capacidad de procesamiento. Dado que HTTP define una única solicitud y una sola respuesta, el protocolo IrOBEX permite: 1) que los dispositivos dividan las solicitudes y respuestas en trozos más pequeños, 2) que los datos sean procesados conforme se van recibiendo y 3) que una solicitud o respuesta sea abortada en cualquier momento.

Al igual que el protocolo HTTP, el protocolo IrOBEX es de transporte neutro, i.e., IrOBEX funciona sobre casi cualquier otro protocolo de la capa de transporte. Las implementaciones iniciales de IrOBEX utilizaban infrarrojos como medio de transporte, pero las implementaciones actuales se ejecutan sobre TCP, serial y conexiones RFCOMM.

El protocolo IrOBEX se volvió más popular desde que el grupo Bluetooth SIG (*Special Interest Group*) autorizó el protocolo de IrDA. Cuando el protocolo IrOBEX se utiliza con la tecnología inalámbrica Bluetooth, la sílaba "Ir" se omite y el protocolo se conoce como OBEX[5]. El grupo Bluetooth SIG define OBEX como uno de los protocolos de la pila de protocolos Bluetooth, de acuerdo con la cual OBEX trabaja sobre el protocolo RFCOMM.

Casos de uso de OBEX

El protocolo OBEX puede ser utilizado para una variedad de propósitos. Este protocolo es utilizado en PDAs como un medio para intercambiar tarjetas de negocios electrónicas y sincronizar dispositivos móviles con computadoras de escritorio. La API OBEX, definida en el lenguaje de programación Java, permite que el protocolo OBEX pueda utilizarse para implementar una gama aún más amplia de aplicaciones, e.g., sincronización e impresión de información.

Caso de uso #1: Sincronización de información

Un problema común de los dispositivos móviles, tales como los teléfonos celulares, se refiere a cómo sincronizar la información de un dispositivo con la de una PC. Mediante la tecnología inalámbrica Bluetooth, el dispositivo no necesita estar conectado a la PC a través de cables. El grupo Bluetooth SIG define el perfil de sincronización para dar soporte a este caso de uso. Este perfil utiliza OBEX para el intercambio de datos entre el dispositivo móvil y la PC.

Caso de uso #2: Impresión de información

Java ME ha comenzado a ser utilizado por las empresas como una manera de mantenerse en contacto con los empleados. Un usuario puede enviar y recibir el correo electrónico por medio de su dispositivo móvil, así como actualizar y revisar su agenda de trabajo y hacer una lista de sus actividades. Existe un inconveniente cuando se utiliza un dispositivo móvil para realizar estas tareas. Como la mayoría de los dispositivos móviles tiene un

[5]De aquí en adelante, OBEX e IrOBEX se usan indistintamente

tamaño de pantalla muy limitado, a los usuarios les resulta más útil enviar su correo electrónico o su agenda a una impresora desde su dispositivo móvil.

4.4.2 Aplicación *AdhocCreatorServices*

`AdhocCreatorServices` está diseñada como una aplicación cliente/servidor OBEX. El servidor simula una impresora a la cual los clientes (usuarios nómadas) le envían datos (e.g., una imagen, un correo electrónico o una agenda) que deben ser impresos.

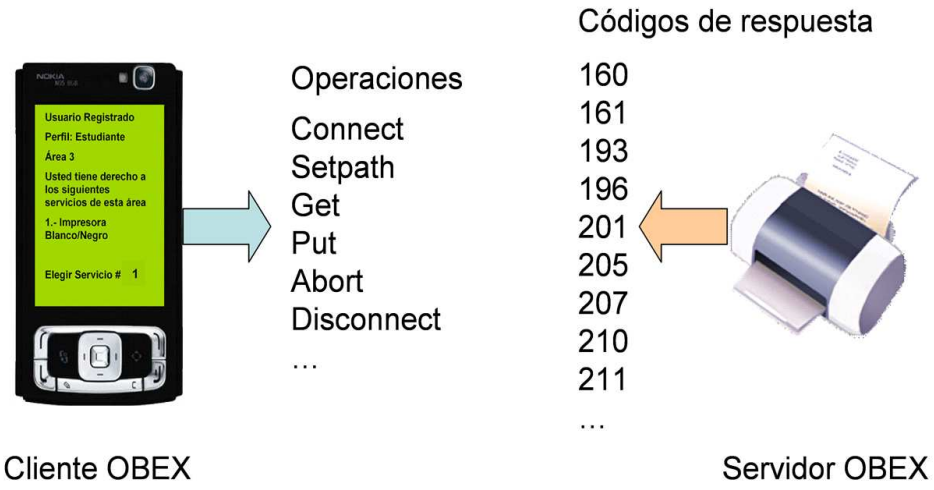

Figura 4.9: Cliente/Servidor OBEX

Cuando los clientes y el servidor se comunican mediante una sesión OBEX, sus interacciones se denominan operaciones o peticiones. Por cada operación enviada desde un cliente, el servidor da una respuesta que indica el estado de la operación (ver Figura 4.9).

Antes de crear una sesión OBEX, se debe establecer una conexión en la capa de transporte por medio de Bluetooth. Una vez establecida la conexión, el cliente emite la operación CONNECT. Si el servidor (i.e., la impresora) está disponible para aceptar un nuevo cliente que utilice su servicio (i.e., impresión), entonces responderá al cliente con el código OK, SUCCESS (i.e., 160). De lo contrario, el servidor le responderá con un código "Servicio OBEX no disponible" (i.e., 211).

Suponga que la impresora acepta la operación CONNECT. Como resultado, se crea una sesión OBEX entre el cliente y el servidor. Por medio de esta sesión, el cliente puede enviar peticiones al servidor. En particular, las operaciones GET y PUT transfieren datos entre el servidor y el cliente. Sin embargo, la operación SETPATH se utiliza principalmente cuando el servidor tiene un sistema de archivos. Por ejemplo, suponga que un cliente envía una petición SETPATH para solicitar al servidor un cambia en el directorio de trabajo. La operación SETPATH generalmente seguida de una operación GET o PUT. Para finalizar la sesión OBEX, el cliente envía la petición DISCONNECT. Si tiene éxito, el servidor le responderá con el código OK, SUCCESS (i.e., 160).

Cliente OBEX

Primeramente, el cliente llama al método `Connector.open ()` para establecer una conexión en la capa de transporte. Después, el cliente invoca al método `connect ()` para establecer una sesión OBEX. Si el servidor acepta la solicitud de conexión, el cliente puede invocar los métodos `put ()`, `get ()`, `setPath ()` o `delete ()` para emitirle una solicitud. Una vez finalizada la comunicación con el servidor, el cliente llama al método `disconnect ()` para terminar la sesión OBEX y posteriormente invoca al método `close ()` para cerrar la conexión en la capa de transporte.

Servidor OBEX

Las conexiones tipo servidor OBEX trabajan ligeramente diferente. Para procesar las solicitudes de un cliente, el servidor proporciona una clase, que extiende la clase *ServerRequestHandler*, al método `acceptAndOpen ()`. Las solicitudes de los clientes pasan al servidor a través de los eventos de la clase *ServerRequestHandler*. En particular, el servidor invoca los métodos `onConnect ()`, `onPut ()`, `onGet ()`, `onSetPath ()`, `onDelete ()` y `OnDisconnect ()` cuando recibe una solicitud del cliente. Por medio de estos métodos, el servidor puede configurar cualquier cabecera para enviar la respuesta junto con su código correspondiente.

Interacción entre el cliente y el servidor OBEX

Cada sesión OBEX comienza cuando el cliente emite la petición CONNECT al servidor. Si el cliente desea, puede incluir cabeceras adicionales en la solicitud.

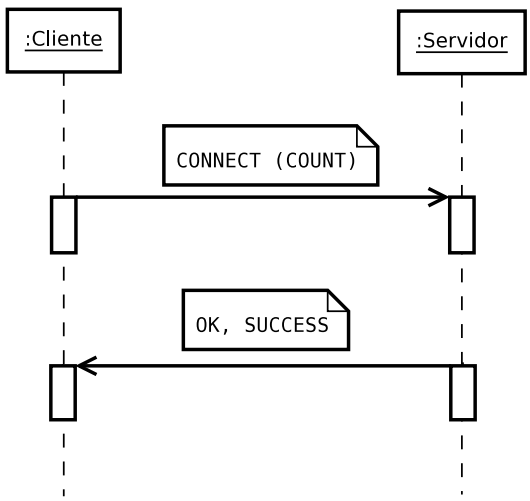

Figura 4.10: Establecimiento de una sesión OBEX

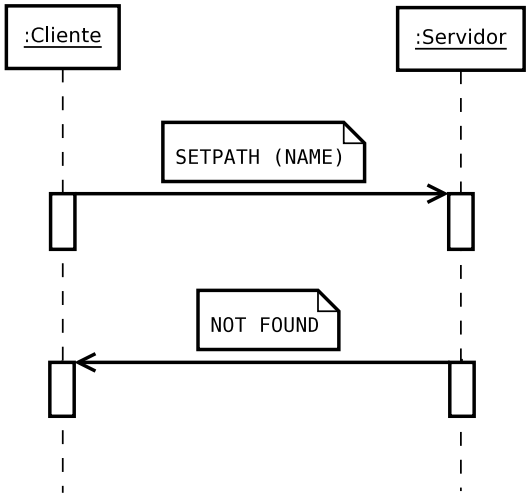

Figura 4.11: Cambio de directorio en el sistema de archivos del servidor OBEX

Cuando el servidor recibe una solicitud de conexión, procesa las cabeceras antes de decidir si la acepta o no. Si el servidor acepta la conexión, entonces responderá con el código OK, SUCCESS. En caso contrario, el servidor responderá con uno de los códigos HTTP que especifica la razón por la cual la solicitud fue rechazada.

El cliente envía al servidor la operación CONNECT junto con la cabecera COUNT para especificar el número de objetos (e.g., imágenes) que requiere transferir (ver Figura 4.10). Después de procesar la petición, el servidor responde con el código OK, SUCCESS.

Una vez establecida la conexión, el cliente necesita moverse a un directorio diferente en el sistema de archivos del servidor. Entonces, el cliente envía al servidor la operación SETPATH en la que especifica el nombre del directorio destino mediante la cabecera NAME (ver Figura 4.11). Cuando el servidor reciba la petición, podrá decidir si autoriza o rechaza el cambio de directorio. El servidor puede negar la solicitud por diversas razones, incluyendo la respuesta NOT FOUND, si el directorio no existe.

Aún si el servidor no fuera capaz de realizar la operación SETPATH, la sesión OBEX permanece activa, de manera que el cliente puede seguir haciendo peticiones. Por ejemplo, el cliente puede enviar un archivo al servidor. En este caso, el cliente emite una operación PUT. Si el archivo es grande, el cliente puede dividirlo en fragmentos más pequeños. Si este es el caso, el cliente envía la operación PUT con las cabeceras NAME y BODY, que contienen respectivamente el nombre y el primer fragmento del archivo (ver Figura 4.12). Cuando el servidor recibe esta operación, almacena el primer fragmento del archivo y responde con la petición CONTINUE. Como resultado, el cliente envía el siguiente fragmento del archivo mediante otra solicitud PUT con la cabecera BODY. Después de almacenar este fragmento del archivo, el servidor envía otra respuesta CONTINUE. Esta interacción continúa hasta que el cliente envía al servidor el último fragmento del archivo. En este caso, el cliente envía nuevamente la operación PUT pero incluye, en vez de la cabecera BODY, la cabecera END OF BODY la cual notifica al servidor que se trata del último fragmento del archivo. Finalmente, el servidor responde con el código OK, SUCCESS, el cual indica al cliente que el archivo se recibió sin problemas.

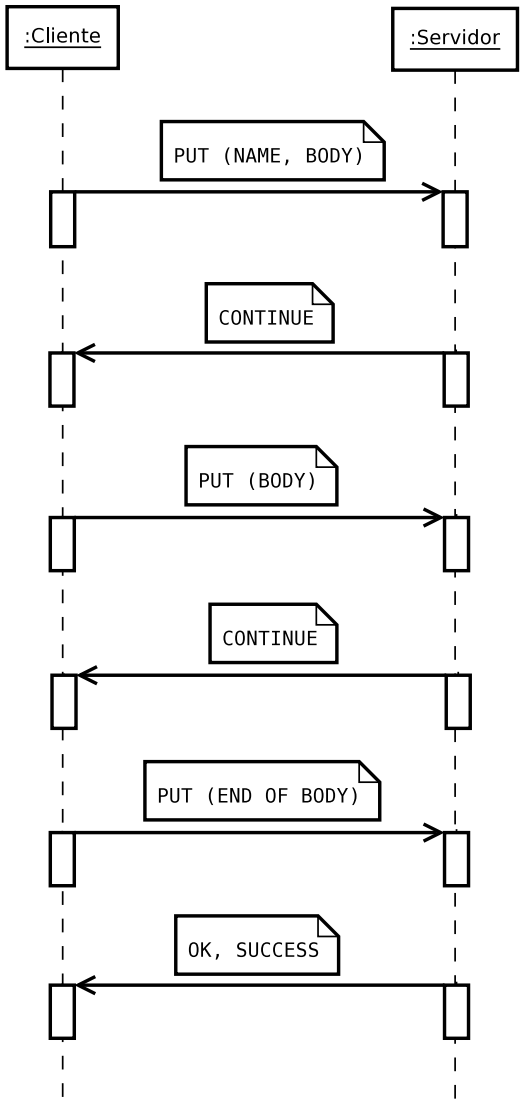

Figura 4.12: Envío de un archivo del cliente al servidor mediante una red *ad hoc*

Para finalizar la sesión OBEX, el cliente emite al servidor la operación DISCONNECT la cual, por lo general, no contiene ninguna cabecera adicional. Cuando el servidor recibe esta operación, libera los recursos que pudo haber reservado y envía al cliente el código OK, SUCCESS (ver Figura 4.13). Cuando el cliente recibe esta respuesta, la sesión OBEX finaliza.

4.4.3 Aplicación *AdhocCreatorNomadasMIDlet*

El protocolo OBEX es adecuado principalmente para la transferencia de información, como tarjetas de negocio y vales electrónicos. El propósito de la aplicación *AdhocCreatorNomadasMIDlet* es el intercambio de tarjetas de negocio entre usuarios nómadas.

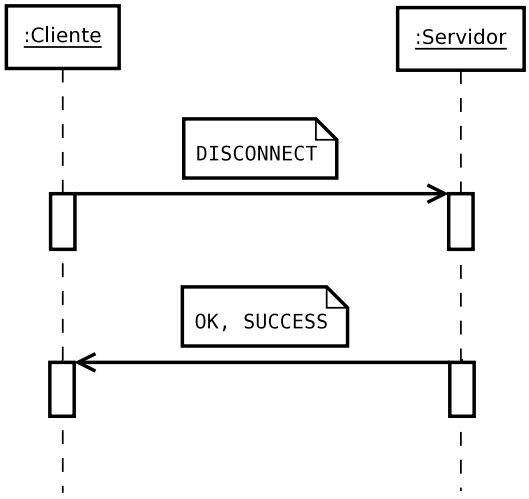

Figura 4.13: Terminación de una sesión OBEX

Un escenario típico de uso de esta aplicación es el siguiente: cuando el *MIDlet* inicia por primera vez, el usuario nómada selecciona su propia tarjeta de negocio a partir de la libreta de direcciones. Así, en el momento en que dos usuarios nómadas se encuentren, estos pueden iniciar el *MIDlet* en sus respectivos dispositivos móviles para poder intercambiar sus tarjetas de negocio. Una vez intercambiadas las tarjetas, estas son almacenadas en el directorio telefónico de cada dispositivo.

Figura 4.14: La libreta de direcciones debe tener al menos un contacto

Cada dispositivo debe tener al menos un contacto (tarjeta de negocio) en la libreta de direcciones. De lo contrario, el *MIDlet* no funcionará correctamente (ver Figura 4.14). Las tarjetas de negocio se entregan directamente a la aplicación, no a la bandeja de entrada. Por lo tanto, el intercambio de tarjetas de negocio requiere que el *MIDlet* se esté ejecutando en el dispositivo móvil de ambos usuarios nómadas.

Diagrama general de la aplicación *AdhocCreatorNomadasMIDlet*

El diagrama general de esta aplicación se ilustra en la Figura 4.15. Inmediatamente después de haber iniciado la aplicación, el usuario selecciona su propia tarjeta de negocio y después pasa al menú principal. Si la tarjeta ya estaba seleccionada, la aplicación inicia directamente desde el menú principal. Mientras el usuario está en el menú principal, es posible: a) seleccionar otra tarjeta, b) iniciar el intercambio de tarjetas, c) aceptar el intercambio de tarjetas o d) salir de la aplicación.

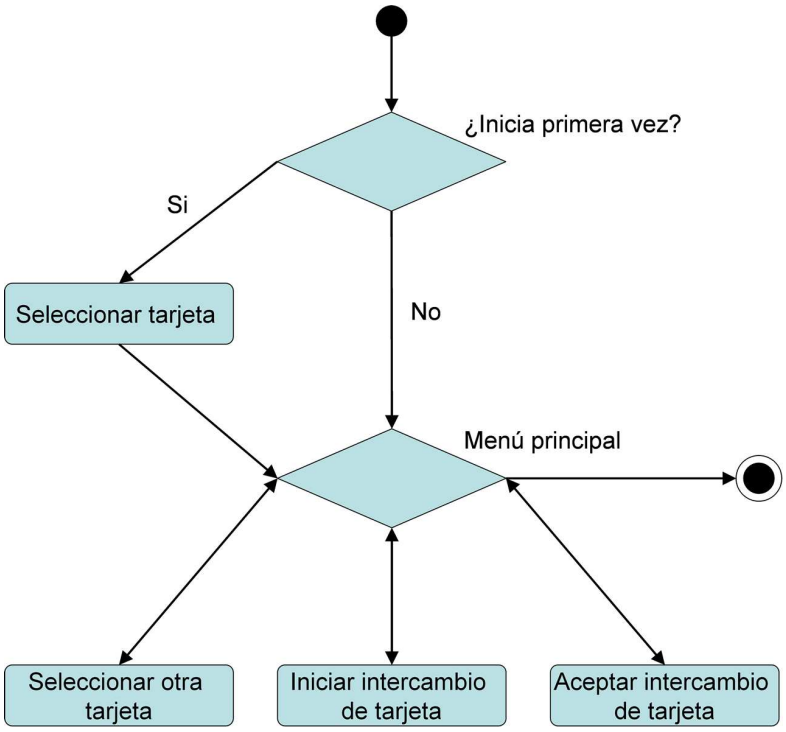

Figura 4.15: Diagrama general de la aplicación *AdhocCreatorNomadasMIDlet*

La Figura 4.16 muestra el diagrama del proceso "Iniciar intercambio de tarjetas de negocio". Cuando un usuario nómada inicia el intercambio de tarjetas desde el menú principal, la aplicación inicia una serie de sub-procesos Bluetooth: a) la búsqueda de dispositivos, b) el descubrimiento de servicios, c) el envío de una tarjeta de negocio, d) la recepción de una tarjeta de negocio y e) el almacenamiento de dicha tarjeta. Además, si varios servicios Bluetooth declarados por la aplicación `AdhocCreatorNomadasMIDlet` son descubiertos, la aplicación muestra la lista de servicios encontrados y pide al usuario que seleccione uno. Cada procedimiento puede ser finalizado o cancelado si se produce

un error. En ambos casos se debe poner fin a la ejecución del sub-proceso y regresar al menú principal.

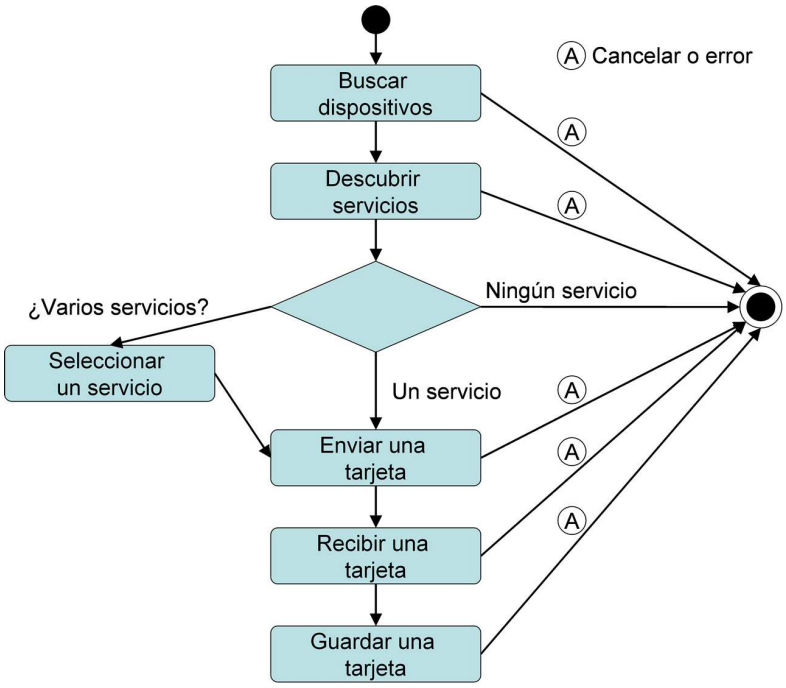

Figura 4.16: Diagrama del proceso "Iniciar intercambio de tarjetas de negocio"

La Figura 4.17 muestra el diagrama del proceso "Aceptar intercambio de tarjetas de negocio". Cuando la aplicación actúa como un *peer* receptor, simplemente recibe la tarjeta de otro usuaio nómada, después la almacena y finalmente envía su propia tarjeta. Además, cada sub-proceso puede cancelarse o fallar. El usuario nómada, que inicia la operación de intercambio de tarjetas, recibe una notificación sobre el éxito o fracaso en la realización de dicha operación.

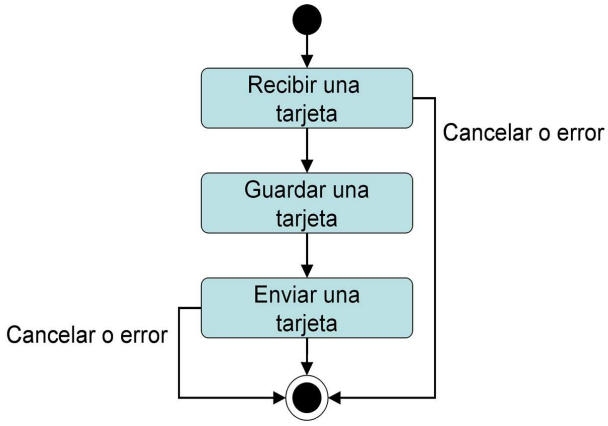

Figura 4.17: Diagrama del proceso "Aceptar intercambio de tarjetas de negocio"

Componentes de la aplicación *AdhocCreatorNomadasMIDlet*

Los componentes de esta aplicación se describen a continuación (ver Figura 4.18):

- El componente `MIDlet` es el coordinador de la aplicación *AdhocCreatorNomadasMIDlet*, ya que administra los componentes `UI`, `Comm`, `Libreta de direcciones` y `Almacenamiento`.
- El componente `UI` es el responsable de la interfaz de usuario de la aplicación. Contiene clases que implementan diversas pantallas del *MIDlet* y las transiciones entre ellas.
- El componente `Comm` se encarga de manejar la comunicación, i.e., todas las operaciones relacionadas con Bluetooth OBEX.
- El componente `Libreta de direcciones` permite acceder a la libreta de direcciones del dispositivo móvil.
- El componente `Almacenamiento` guarda la información acerca de la tarjeta de negocio elegida como propia.

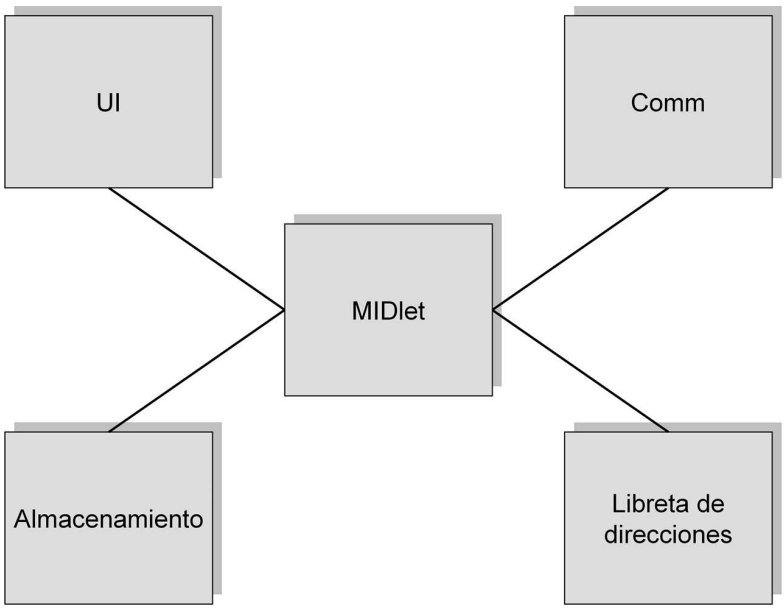

Figura 4.18: Componentes de la aplicación *AdhocCreatorNomadasMIDlet*

Los componentes `Almacenamiento` y `Libreta de direcciones` contienen sólo una clase cada uno. La única clase del componente `Almacenamiento` utiliza el perfil MIDP 2.0 *Record Management System* (RMS), mientras que la única clase del componente `Libreta de direcciones` emplea el API PIM (*Personal Information Management*) [JSR-75].

Los componentes `UI` y `Comm` son más sofisticados. El componente `Comm` es responsable de diversas operaciones de comunicación basadas en Bluetooth y OBEX, en tanto que

el componente `UI` representa la interfaz de usuario de la aplicación. Cada componente contiene clases que implementan una máquina de estados.

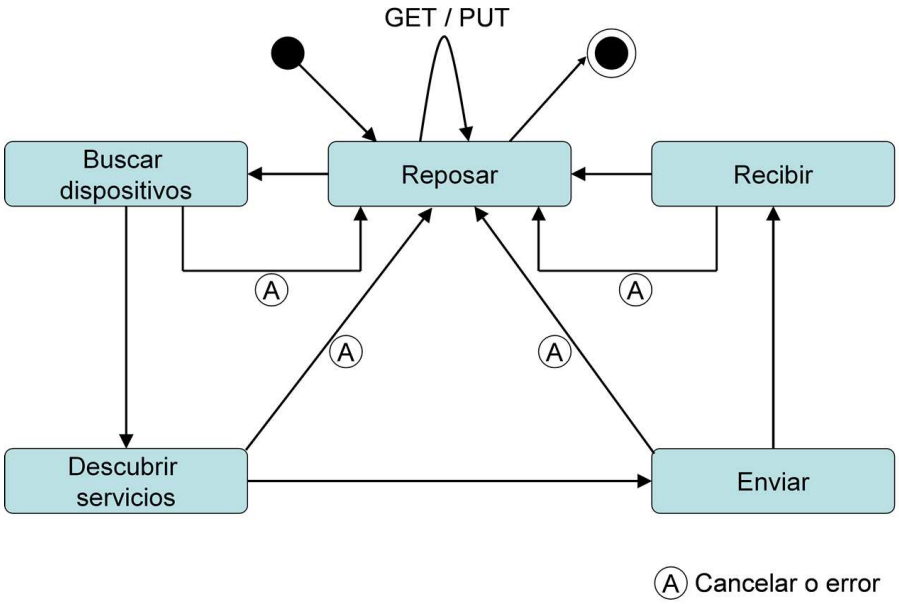

Figura 4.19: Diagrama de estados del componente Comm

La Figura 4.19 muestra el diagrama de estado del componente `Comm`. Cuando la aplicación inicia el intercambio de tarjetas de negocio, el flujo normal de cambio de estados es `Reposar` → `Búscar dispositivos` → `Descubrir servicios` → `Enviar tarjeta` → `Recibir tarjeta` → `Reposar`. Sin embargo, cada operación puede fallar, lo que resulta en la cancelación del proceso de intercambio de tarjetas y en el regreso al estado `Reposar`.

La aceptación de una conexión se implementa como servicio de peticiones OBEX GET y PUT, por lo que se realiza sin cambio de estado. Sin embargo, se debe tener en cuenta que la aplicación debe iniciar en el estado `Reposar` para poder realizar las peticiones OBEX. En otras palabras, la aplicación está siempre lista para aceptar una solicitud de intercambio de tarjetas proveniente de un dispositivo remoto, si no está realizando otro intercambio en ese momento.

El componente `UI` presenta un diagrama de estado similar al que se muestra en la Figura 4.20. Cada estado del componente `UI` representa una pantalla separada. Al iniciar la aplicación, el estado inicial es `Pantalla libreta de direcciones` (si la aplicación inicia por primera vez) o `Pantalla principal` (si la aplicación ha iniciado antes). Mientras la aplicación esté en el estado `Pantalla principal`, esta puede moverse al estado `Pantalla libreta de direcciones`, si el usuario necesita cambiar su tarjeta, y luego transferirse de nuevo al estado `Pantalla principal`, después de haber seleccionado la tarjeta. Se debe tener en cuenta que la aplicación puede terminar solo si se encuentra en los estados `Pantalla principal` o `Pantalla libreta de direcciones`.

Cuando el usuario inicia el intercambio de tarjetas desde el estado `Pantalla principal`, se lleva a cabo varios cambios de estado (pantallas) como se muestra en la Figura 4.20. Cada pantalla de progreso (Progreso búsqueda de dispositivos, Progreso descubrimiento

de servicios, Progreso enviando y Progreso recibiendo) se transfiere al siguiente estado si finaliza con éxito o al estado `Pantalla principal` en caso de fallo o cancelación. La única excepción es el estado `Progreso descubrimiento de servicios`, ya que si la aplicación encuentra más de un servicio, la interfaz de usuario cambiará al estado `Pantalla lista de servicios` que permite al usuario nómada seleccionar un único servicio.

Otra pantalla de la aplicación, denominada `Pantalla mensajes de alerta`, se utiliza para mostrar los resultados de una operación (e.g., tarjeta de negocios enviada o búsqueda cancelada de dispositivos) durante un período limitado de tiempo (mediante el perfil MIDP 2.0 *Alert class*). Por lo tanto, no se considera como un estado del componente UI.

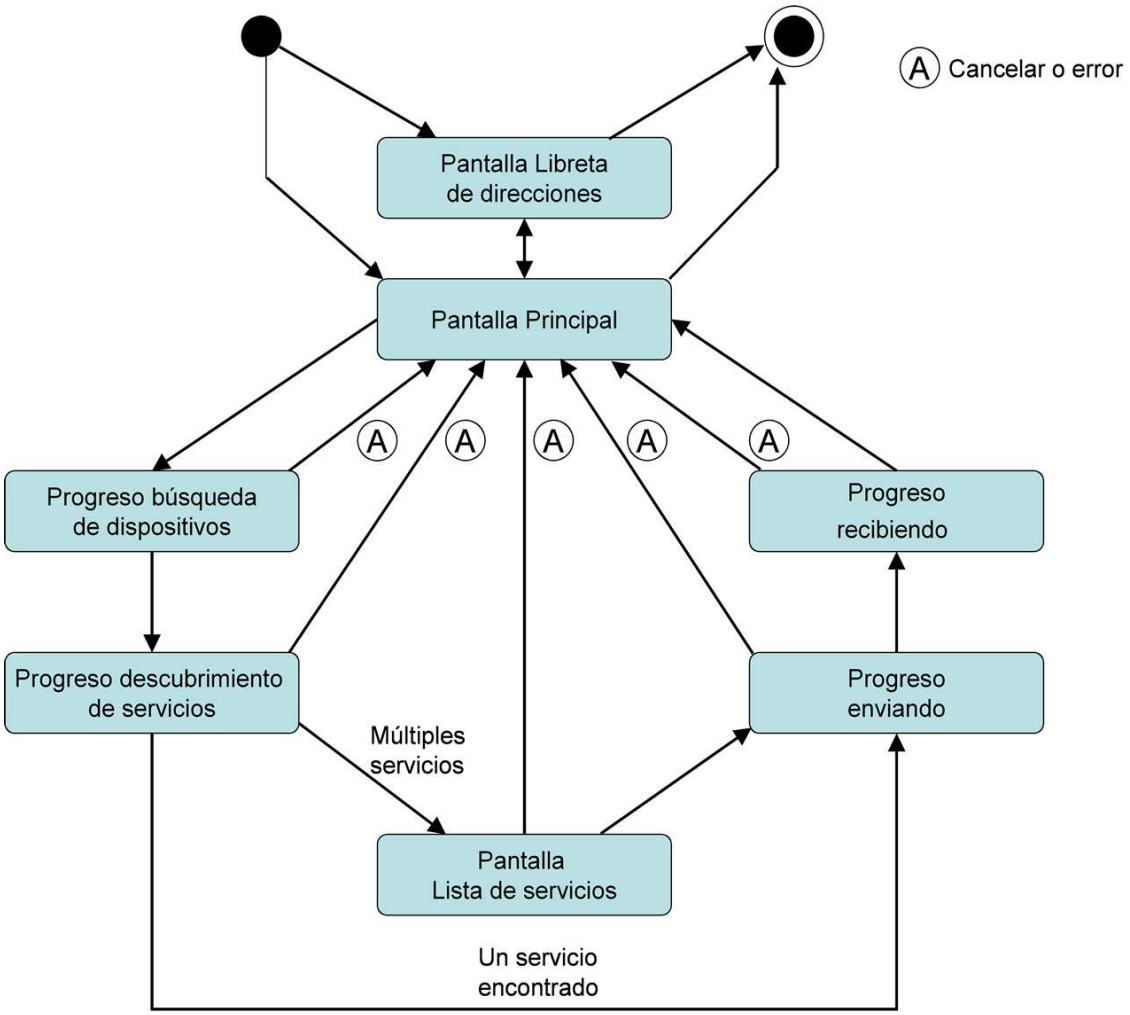

Figura 4.20: Diagrama de estados del componente UI

Capítulo 5

Conclusiones y trabajo futuro

El objetivo de esta tesis de maestría fue diseñar e implementar un sistema denominado SEDINU (*SErvice DIscovery for Nomadic Users*) el cual permite a los usuarios nómadas, inmersos en un área autónoma, interactuar con otros usuarios y con los servicios disponibles en dicha área. En este capítulo, primeramente se hace una recapitulación del contexto de investigación y de la problemática abordada (sección 5.1). Enseguida, se presenta una síntesis de las contribuciones de este trabajo de investigación (sección 5.2). Finalmente, se presenta algunas limitaciones y posibles extensiones futuras del sistema SEDINU (sección 5.3).

5.1 Recapitulación de la problemática

Este trabajo de tesis está inmerso en el campo de investigación del cómputo ubicuo, cuyo objetivo es incrementar el uso de sistemas computacionales a través del entorno físico al hacerlos disponibles, pero también invisibles al usuario. Por lo tanto, el cómputo ubicuo busca desarrollar sistemas inteligentes que se adapten al usuario y que se utilicen de manera intuitiva.

El cómputo ubicuo da soporte a la información contextual y al descubrimiento de servicios para realizar sus funciones e.g., detectar los cambios de ubicación de un usuario nómada y descubrir los servicios disponibles en el entorno en función de su ubicación.

El descubrimiento de servicios es el proceso mediante el cual una entidad es notificada de forma espontánea de la disponibilidad de servicios en una red, i.e., es un mecanismo para referenciar dinámicamente a un servicio. Una red hace disponibles a los servicios a través de su registro para que puedan ser localizados, mediante la búsqueda de un servicio de directorio o la notificación periódica de anuncios en la red. Por su parte, los clientes establecen criterios que describen los servicios en los que están interesados. Estos criterios son utilizados por el sistema de descubrimiento para determinar los servicios adecuados que satisfacen la solicitud de un cliente. Desafortunadamente, los clientes de estos servicios son principalmente programas. Por lo tanto, estos sistemas no soportan interacciones entre los usuarios y los servicios presentes en un entorno ni mucho menos interacciones entre usuarios. Por esta razón, es necesario un soporte computacional y comunicacional que facilite la interacción entre un usuario nómada y los servicios ofrecidos por el área actual donde él se encuentra, así como la interacción entre usuarios nómadas inmersos en dicha

área. Estos tipos de interacción dependen de un contexto específico, definido en términos de rol, localización y objetivos de los usuarios.

Este trabajo de investigación esta basado en el concepto de "área autónoma", la cual se define como una entidad lógica que proporciona servicios especializados y que maneja sus recursos de forma independiente, con el fin de ayudar a los usuarios nómadas en la realización de sus objetivos parciales y globales. Un área autónoma gestiona los recursos de su dominio, mientras observa las políticas de la organización (e.g., seguridad y facturación). La gestión de los recursos de un área autónoma incluye: información de los servicios disponibles, información contextual del área, roles de usuario, intercambio de información con otras áreas de la organización y plan de actividades.

5.2 Conclusiones

En esta tesis de maestría se propuso el sistema SEDINU el cual ha sido diseñado con la finalidad de ser implantado en organizaciones estructuradas con base en una jerarquía de áreas autónomas (e.g., empresas, instituciones públicas o administraciones). Su objetivo principal es facilitar la interacción entre los usuarios nómadas y los servicios proporcionados por las áreas autónomas, así como la colaboración entre los usuarios nómadas bajo contextos específicos. Cuando un usuario nómada entra a un edificio, el módulo `Location Detector` del sistema SEDINU localiza su dispositivo móvil (e.g., un PDA, teléfono celular, etc.) a fin de: 1) determinar el área autónoma correspondiente y 2) asignar un rol al usuario nómada. Este módulo de localización, basado en triangulación de señales WiFi, permite localizar el dispositivo móvil del usuario, no al usuario mismo.

Debido a que el usuario puede separarse temporalmente de su dispositivo móvil se propuso, como complemento del módulo de localización por triangulación (el cual aún sigue en desarrollo) un sistema de reconocimiento de caras, cuyos objetivos son identificar y ubicar al usuario dentro de la organización. Sin embargo, este sistema incluye una fase de aprendizaje que requiere la adquisición de varias imágenes de la cara de cada usuario sujeto a reconocimiento. Es evidente que este sistema no puede ser empleado para gestionar usuarios ajenos a la organización (e.g., visitantes o repartidores). Este sistema es más adecuado para administrar los recursos humanos de la organización (e.g., personal administrativo, profesores, jefes y directivos). No obstante sus respectivas limitaciones, la combinación de estas dos técnicas de localización (triangulación WiFi y reconocimiento de caras) resulta conveniente.

De acuerdo con la localización actual y el rol del usuario, el sistema SEDINU determina los servicios disponibles al usuario nómada y crea dinámicamente, por medio del sistema RBAC-Soft, el flujo de trabajo que va a guiar sus interacciones/colaboraciones. Las aplicaciones requeridas para interactuar con los servicios disponibles se cargan automáticamente al dispositivo móvil del usuario nómada. Tan pronto como él seleccione un servicio, el sistema SEDINU puede crear una red *ad hoc*: a) entre el dispositivo móvil del usuario y el *host* de servicios para permitir la interacción servicio-usuario o b) entre los dispositivos móviles de los usuarios para facilitar la colaboración entre ellos.

5.3 Trabajo futuro

La versión actual del sistema SEDINU crea redes *ad hoc* entre dispositivos móviles mediante el protocolo Bluetooth, el cual ofrece varias ventajas: a) tiene una infraestructura de bajo costo; b) los usuarios no requieren de un profundo conocimiento de administración de redes y c) la mayoría de los dispositivos móviles tienen conexión Bluetooth. Por lo tanto, el sistema SEDINU es compatible con varios dispositivos móviles.

Los módulos del sistema SEDINU que aún no han sido implementado son:

- Módulo `Tiny-SEDINU Application`: aplicación embebida en los dispositivos móviles que ayuda a un usuario nómada a interactuar con los servicios disponibles en el área autónoma actual;
- Módulo `Service Manager`: responsable de cargar una aplicación cliente o *peer* para acceder a los servicios disponibles en dicha área.

Como futuras extensiones de este sistema, se desarrollarán módulos para crear redes *ad hoc* por medio del protocolo ZigBee [Labiod et al., 2007], el cual puede ser utilizado por aplicaciones que no requieran gran transmisión de datos. Además, el consumo de batería de un dispositivo ZigBee es mucho más bajo que el de los dispositivos Bluetooth o WiFi. De esta manera, el sistema SEDINU podrá crear redes *ad hoc* para controlar el acceso a ciertas áreas autónomas restringidas (e.g., un laboratorio de radioactividad) i.e., solo los usuarios nómadas autorizados serán capaces de penetrar a esas áreas. Asimismo, el sistema SEDINU constituye la base para la definición de un *framework* que dará soporte a la colaboración nómada y ubicua.

Adicionalmente, se propone el desarrollo de las siguientes aplicaciones:

- una interfaz de comunicación entre el sistema SEDINU y el sistema de administración de flujos de trabajo (RBAC-Soft) de manera que el primero pueda:
  1. obtener el plan de actividades del usuario nómada, correspondiente al área autónoma actual e,
  2. informar al sistema de administración de flujos de trabajo sobre los cambios de estado del plan de actividades y notificar dichos cambios contextuales al área autónoma actual.
- Construir un mecanismo para detectar cambios contextuales en el área autónoma actual y en otras áreas, que puedan tener influencia en el plan de actividades del usuario nómada.

Publicaciones del autor

[Gómez et al., 2009] V. Gómez, S. Mendoza, D. Decouchant and J. Rodríguez, *Nomadic User Interaction/Cooperation within Autonomous Areas*, In Proceedings of the 15th Collaboration Researchers International Workshop on Groupware, pp. 32-40, LNCS 5784 Springer, Peso da Régua (Portugal), September 13-17 2009

[Mendoza et al., 2009] S. Mendoza, V. Gómez, M. Navarrete, D. Decouchant, K. García, G. Olague and J. Rodríguez, *Area-based Collaborative Ubiquitous Work within Organizational Environments*, In Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, pp. 140-144, IEEE Computer Society, Milan (Italy), September 15-18 2009

Referencias

[Adjie-Winoto et al., 1999] Adjie-Winoto, W., Schwartz, E., Balakrishnan, H., and Lilley, J. (December 1999). *The Design and Implementation of an Intentional Naming System.* In Proc. 17th ACM Symp. Operating System Principles (SOSP 99), pp. 186-201, ACM Press, Charleston, South Carolina (USA).

[Bardram, 2007] Bardram, J. (January 2007). *Pervasive Computing Support for Hospitals: An overview of the Activity-Based Computing Project.* IEEE Pervasive Computing, Vol. 6, No. 1, pp. 44-51.

[Campo et al., 2006a] Campo, C., Almenarez, F., Diaz, D., Garcia-Rubio, C., and Marín, A. (March 2006a). *Secure Service Discovery based on Trust Management for ad-hoc Network.* Journal of Universal Computer Science, Vol. 12, No. 3, pp. 340-356.

[Campo et al., 2006b] Campo, C., F., A., Diaz, D., Garcia-Rubio, C., and Lopez, A. (March 2006b). *Secure Service Discovery based on Trust Management for ad-hoc Network.* Universal Computer Science, Vol. 12, No. 3, pp. 340-356.

[Chan et al., 2003] Chan, W.-C., Chen, J.-L., Lin, P.-T., and Yen, K.-C. (December 2003). *Quality of service in IP services over Bluetooth ad-hoc networks.* Mobile Networks and Applications, Vol. 8, No. 6, pp. 699-709.

[Chang et al., 2007] Chang, C., Sahoo, P., and Lee, S. (January 2007). *A Location-Aware Routing Protocol for the Bluetooth Scatternet.* Wireless Personal Comm., Vol. 40, No. 1, pp. 117-135.

[Chlamtac et al., 2003] Chlamtac, I., Conti, M., and J.N. Liu, J. (July 2003). *Mobile ad hoc networking: imperatives and challenges.* Ad Hoc Networks, Vol. 1, No. 1, pp. 13-64, Elsevier.

[Dey, 2000] Dey, A. K. (November 2000). *Providing Architectural Support for Building Context-Aware Applications.* PhD. Thesis, Georgia Institute of Technology.

[Edwards, 2006] Edwards, W. K. (April-June 2006). *Discovery Systems in Ubiquitous Computing.* IEEE Pervasive Computing, Vol. 5, No. 2, pp.70-77.

[Feng and Zhu, 2001] Feng, Y. and Zhu, J. (May 2001). *Wireless Java Programming with J2ME.* 1st Edition, Sams, Indianapolis, IN, USA.

[Freebersyser and Leiner, 2001] Freebersyser, J. A. and Leiner, B. (2001). *A DoD perspective on mobile ad hoc networks.* Ad hoc networking, Addison-Wesley Longman Publishing Co., Boston, MA, pp. 29-51.

[Gaertner and Cahill, 2004] Gaertner, G. and Cahill, V. (January 2004). *Understanding Link Quality in 802.11 Mobile Ad Hoc Networks.* IEEE Internet Computing, Vol. 8, No. 1, pp. 55-60.

[Garcia, 2009] Garcia, K. (January 2009). *Disponibilidad de recursos compartidos en un ambiente colaborativo con carácter ubicuo.* Master Thesis, Departamento de Computación, CINVESTAV.

[Garcia et al., 2008] Garcia, K., Mendoza, S., Olague, G., Decouchant, D., and Rodriguez, J. (September 2008). *Shared Resource Availability within Ubiquitous Collaboration Environments.* In Proc. of CRIWG 2008, The 14th International Collaboration Researchers Conference on Workshop on Groupware, pp. 25-40, LNCS, Springer, Omaha NE (USA).

[Han et al., 2006] Han, J., Cho, Y., Kim, E., and Choi, J. (May 2006). *A Ubiquitous Workflow Service Framework.* Computational Science and its Applications - ICCSA 2006,pp. 30-39, Springer LNCS 3983, Glasgow (UK).

[Haque and O'Connor, 2002] Haque, I. and O'Connor, B. (April 2002). *J2ME Enterprise Development.* 1st Edition, Wiley, New York, NY, USA.

[Herborn et al., 2005] Herborn, S., Lopez, Y., and Seneviratne, A. (November 2005). *A Distributed Scheme for Autonomous Service Composition.* In Proc. of the First ACM Int. Workshop on Multimedia Service Composition, pp. 21-30, ACM Press, Hilton, Singapore.

[Hernández et al., 2007] Hernández, B., Olague, G., Hammoud, R., Trujillo, L., and Romero, E. (May 2007). *Visual learning of texture descriptors for facial expression recognition in thermal imagery.* Computer Vision and Image Understanding, Vol. 106, No. 2-3, pp. 258-269.

[Huang and Rudolph, 2007] Huang, A. and Rudolph, L. (September 2007). *Bluetooth Essentials for Programmers.* 1st Edition, Cambridge University Press, New York, NY, USA.

[Jamalipour, 2003] Jamalipour, A. (March 2003). *The Wireless Mobile Internet: Architectures, Protocols and Services.* John Wiley and Sons, Inc., New York, NY (USA).

[Jeronimo and Weast, 2003] Jeronimo, M. and Weast, J. (May 2003). *UPnP Designe by Example: A Software Developer's Guide to Universal Plug and Play.* Intel Press.

[Ji and Corson, 2003] Ji, L. and Corson, M. S. (October 2003). *Explicit multicasting for mobile ad hoc networks.* Mobile Networks and Applications, Vol. 8, No. 5, pp. 535-549.

[Juntao Yuan, 2003] Juntao Yuan, M. (November 2003). *Enterprise J2ME: Developing Mobile Java Applications*. 1st Edition, Prentice Hall PTR, Upper Saddle River, NJ, USA.

[Kesselman et al., 2005] Kesselman, A., Kowalski, D., and Segal, M. (January 2005). *Energy efficient communication in ad hoc networks from user's and designer's perspective*. ACM SIGMOBILE Mobile Computing and Communications Review, Vol. 9, No. 1, pp. 15-26.

[Kim et al., 2001] Kim, S., Chung, Y., Jung, S., and Hwang, C.-S. (June 2001). *Optimistic Transaction Processing Algorithms in Pure-Push and Adaptive Broadcast Environments*. In Proc. of the Eighth International Conference on Parallel and Distributed Systems, pp. 61-68, IEEE Computer Society, Kyongju City, Korea.

[Labiod et al., 2007] Labiod, H., Afifi, H., and De Santis, C. (June 2007). *Wi-Fi, Bluetooth, ZigBee and WiMax*. 1st Edition, Springer, Dordrecht (The Netherlands).

[Mendoza et al., 2009] Mendoza, S., Gómez, V., Navarrete, M., Decouchant, D., Garcia, K., Olague, G., and Rodriguez, J. (September 15-18, 2009). *Area-based Collaborative Ubiquitous Work within Organizational Environments*. In Proc. of the 2009 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2009), pp. 140-144, IEEE Computer Society, Milan (Italy).

[Nidd, 2001] Nidd, M. (August 2001). *Service Discovery in DEAPspace*. IEEE Personal Comm., pp. 39-45.

[Oaks and Wong, 2000] Oaks, S. and Wong, H. (March 2000). *Jini in a Nutshell: A Desktop Quick Reference*. O'Reilly and Associates, Inc., Sebastopol, CA, USA.

[Roth, 2006] Roth, J. (April 2006). *Detecting identifiable areas in mobile environments*. In Proc. of the 2006 ACM Symposium on Applied Computing, pp. 986-991, ACM, Dijon, France.

[Thompson et al., 2008] Thompson, T. J., Kumar, C. B., and Kline, P. (February 2008). *Bluetooth Application Programming with the Java APIs*. Morgan Kaufmann Publishers, Burlington,MA., USA.

[Weiser, 2002] Weiser, M. (January-March 2002). *The Computer for the 21st Century*. Scientific American, Sept., 1991, pp. 94-104, reprinted in IEEE Pervasive Computing, pp. 19-25.

[White, 2007] White, K. (2007). *Apple Training Series Mac OS x Support Essentials*. Peachpit Press, Berkeley, CA (USA).

[Zhang et al., 2006] Zhang, Y., Zhang, S., Vuong, S., and Malik, K. (July 2006). *Mobile learning with bluetooth-based E-learning system*. In Proc. of the 2006 International Conference on Wireless Communications and Mobile Computing, pp. 951-956, ACM, Vancouver, British Columbia, Canada.

[Zhao and Schulzrinne, 2005] Zhao, W. and Schulzrinne, H. (February 2005). *Enhancing Service Location Protocol for Efficiency, Scalability and Advanced Discovery.* Journal of Systems and Software, Vol. 75, No. 1-2, pp. 193-204.

[Zhu et al., 2005] Zhu, F., W. Mutka, M., and M.Ni, L. (Oct.-Dec. 2005). *Service discovery in pervasive computing environments.* IEEE Pervasive Computing, Vol. 4, No. 4, pp. 81-90.

[Zhu and Li, 2008] Zhu, G. and Li, D. (December 2008). *Method for land consolidation progress assessment based on GPS and PDA.* In Proc. of the 13th International Conference on Applied Mathematics, pp. 291-296, World Scientific and Engineering Academy and Society (WSEAS), Puerto De La Cruz, Spain.

[Zhu and Chlamtac, 2006] Zhu, H. and Chlamtac, I. (August 2006). *Admission control and bandwidth reservation in multi-hop ad hoc networks.* Computer Networks, Vol. 50, No. 11, pp. 1653-1674.