INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

ARQUITECTURA PARA LA COMUNICACIÓN E INTEROPERABILIDAD DE DISPOSITIVOS MÓVILES BASADA EN EL CONTEXTO GEOGRÁFICO

T E S I S

QUE PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS EN INGENIERÍA DE CÓMPUTO

P R E S E N T A

ING. DANIEL MARTÍNEZ SALINAS

DIRECTORES DE TESIS: **DR. JOSÉ GIOVANNI GUZMÁN LUGO**

DR. OSCAR CAMACHO NIETO

MÉXICO, D.F., JULIO 2010

Dedicatoria

A Dios.

Por haberme permitido llegar hasta este punto y haberme dado la suficiente energía para lograr mis objetivos, además de su infinita bondad y amor en todos los sentidos.

A mis padres: Jesús Martínez y Martha Salinas.

Que con sus sacrificios y oraciones me han llevado hasta donde estoy ahora, por enseñarme a luchar hacia delante, por su gran corazón y capacidad de entrega, pero sobre todo por enseñarme a ser responsable. Mi triunfo es el de ustedes, ¡los amo!

DANIEL MARTÍNEZ SALINAS

Agradecimientos

Agradezco a Dios que me ha permitido seguir, a pesar de los numerosos problemas que surgieron en el transcurso de este trabajo, su infinito apoyo.

Agradezco a mis padres Jesús Martínez Rojas, Martha Salinas Calderón, por su infinita bondad, por creer siempre en mí, por sus oraciones, por todo su gran apoyo, tanto moral e espiritual. Gracias.

Agradezco al Dr. JOSÉ GIOVANNI GUZMÁN LUGO por su gran apoyo, confianza, paciencia y motivación en todo momento, para la culminación de este trabajo.

Agradezco al Dr. OSCAR CAMACHO NIETO, por alentarme a seguir adelante con este sueño, paciencia y sus sabios consejos para la elaboración de este trabajo.

DANIEL MARTÍNEZ SALINAS

Agradecimiento especial

Al Centro de Investigación en Computación del Instituto Politécnico Nacional, por el soporte y por la oportunidad que me ha brindado de poder materializar un sueño que he tenido y además permitirme crecer como persona.

También quiero hacer un agradecimiento muy especial al Dr. José Luis Oropeza Rodríguez, por su gran esfuerzo y compromiso en la revisión del presente trabajo de tesis.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) porque debido a la beca que se me otorgó he podido culminar de forma exitosa mis estudios de maestría en el Centro de Investigación en Computación del Instituto Politécnico Nacional.

De igual manera quiero agradecer al Programa Institucional de Formación de Investigadores (PIFI) del I. P. N. por haberme apoyado con el otorgamiento de una beca, lo cual ha sido de gran relevancia en el logro de este trabajo de tesis.

De corazón Gracias

DANIEL MARTÍNEZ SALINAS

RESUMEN

El cómputo ubicuo se ha posicionado como una necesidad apremiante en el entorno social, la cual esencialmente pretende dotar de una capacidad de cómputo a los objetos de la vida diaria, así como de una conciencia contextual con el fin de lograr satisfacer las necesidades de los usuarios.

Desafortunadamente los problemas de incompatibilidad tanto en el hardware como en el software, entre los diversos desarrollos de sistemas ubicuos, han dificultado notoriamente el desarrollo de programas o herramientas que permitan llevar a la práctica la creación de entornos ubicuos generalizados.

Es por ello, que la mayor parte de las soluciones actuales tienden a estar limitadas a una plataforma específica. Asimismo, para conseguir la comunicación entre los objetos de dichos entornos se hace uso de medios de transmisión inalámbricos, lo cual pone en riesgo la confidencialidad de la información que se transmite a través del medio. Finalmente, los servicios que se ponen a disposición de las personas en dichos ambientes tienen que contar con información contextual como pueden ser preferencias y particularmente ubicación geográfica del usuario.

Considerando lo antes expuesto en el presente trabajo se ofrece una arquitectura, la cual permite a través de la integración tecnológica de elementos de hardware y software, la materialización de un entorno de cómputo ubicuo, capaz de lograr una interoperabilidad espontánea entre un usuario y dicho entorno de una manera totalmente transparente para el uso de un servicio de impresión. Dicha iteración es llevada a cabo por medio de dispositivos móviles, además se asegura la integridad entre intercambio de información, mediante el desarrollo de un esquema de seguridad basado en técnicas criptográficas. El entorno ubicuo propuesto cuenta con un mecanismo de conciencia contextual (entorno geográfico), para lograr adaptar las respuestas del entorno hacia las necesidades del usuario.

Por otra parte, la arquitectura propuesta tiene la característica de ser escalable, ya que se pueden agregar fácilmente diferentes tipos de servicios, acorde a las necesidades específicas de usuarios finales de esta arquitectura, sin necesidad de modificar el núcleo del entorno ubicuo.

ABSTRACT

Ubiquitous computing is now a days a social necessity. It is aimed to provide every day objects with computing capabilities as well as context awareness. All these have the objective of satisfying the necessities of the human users.

However, compatibility problems both in hardware and software have undermined the potential of this type of tools and environments and have imposed important challenges to practical deployments.

It is for this reason that most of the current solutions are platform dependent. On the other hand, ubiquitous components are mainly connected through wireless links which inherently imposes another set of challenges such as security and confidentiality. Lastly, ubiquitous components must be aware of their context and in particular of their users' preferences and geographic location.

In this context, we present an architecture that supports the integration of software and hardware components and in this way fosters the creation of ubiquitous environments.

The objective of these environments is to provide is to provide spontaneous interaction between a user and her environment and in particular between a user and a printing service. The human-environment interaction is carried out through mobile devices in a secure way. The environment has context awareness that is used to adapt its responses to the necessities of the users.

Or architecture is flexible in the sense that it can easily accommodate other type of services. The latter has the objective of meeting the particular necessities of the end users without the need of further changes to our architecture.

CAPÍTULO 1
INTRODUCCIÓN

Resumen. En este capítulo se presenta una visión general acerca del proyecto propuesto en esta tesis. Se introducen las bases del estudio que se va a realizar en los capítulos posteriores, los objetivos y beneficios del presente trabajo así como un panorama general de la solución propuesta. Finalmente se plantea la organización del resto del documento.

1.1 Antecedentes.

Actualmente, el uso de las redes inalámbricas así como de dispositivos móviles está creciendo significativamente, es difícil concebir el desarrollo de nuestras actividades cotidianas sin alguno de estos dos elementos. Sin embargo, los usuarios enfrentan diversos problemas de heterogeneidad que dificultan considerablemente el intercambio de información a través de dispositivos móviles diferentes. El desarrollo de sistemas móviles ha impulsado paralelamente nuevas versiones de sistemas operativos, los cuales en ocasiones generan una barrera para una adecuada y correcta interoperabilidad de los usuarios.

Aunque existe el esfuerzo, por parte de las compañías que desarrollan estos productos, para simplificar el desarrollo de aplicaciones homogéneas para los diferentes tipos de dispositivos móviles, es común que un individuo dedique gran cantidad de tiempo en procesos de búsqueda y configuración para poder acceder a otros dispositivos o servicios que se encuentran en su entorno. De igual forma, aunque existen diferentes tecnologías de comunicación de forma inalámbrica (WiFi, Bluetooth, Infrarrojos) éstas presentan el inconveniente de que la información puede ser interceptada y analizada, es decir, es vulnerable por la carencia de un mecanismo de protección de los datos (problema que resuelve comúnmente mediante la encriptación).

La Computación Ubicua *(Ubiquitous Computing)* o también nombrada por algunos expertos como Ubicomp, es un término denominado por Mark Weiser [1], para definir a los sistemas cuya meta es el incremento en el uso de sistemas de cómputo a través del ambiente físico, haciéndolos disponibles y a la vez invisibles al usuario. Este trabajo se ha posicionado como un nuevo paradigma en la computación y en años recientes, una gran cantidad de investigadores se han dado a la tarea de materializar las ideas de Mark Weiser y construir sistemas de cómputo ubicuo.

En esencia, un sistema ubicuo es un ambiente saturado con elementos de cómputo y comunicación que están integrados con las tareas de sus usuarios humanos [2] y que tiene como características fundamentales la integración física entre elementos de la vida diaria y elementos de software, y la interoperabilidad espontánea entre componentes ubicuos. Los sistemas ubicuos que están integrados por componentes de hardware o software, los cuales proporcionan una interfaz para ser controlados por otros elementos del sistema sin que tengan conocimiento previo unos de otros, generalmente son denotados como sistemas de control ubicuo.

Para lograr un sistema de control ubicuo es necesario que existan arquitecturas de hardware y software que permitan la integración dinámica de componentes. Esto implica desarrollar protocolos que permitan que componentes heterogéneos sean capaces de

reconocer dinámicamente otros componentes con los que puedan colaborar con un propósito específico.

Por otro lado, las redes que soportan a los sistemas ubicuos, además de tener contemplados a los dispositivos inalámbricos, deben ser altamente dinámicas y permitir la constante entrada y salida de componentes sin la necesidad de realizar tareas de configuración explícita [3].

En los últimos quince años se ha profundizado en el estudio de sistemas de igual a igual (P2P), cuya principal característica es que son sistemas distribuidos en donde el software que se ejecuta en cada nodo, proporciona funciones equivalentes y no existe ningún control centralizado u organización jerárquica. Estas propiedades dotan a los sistemas de igual-a-igual con características como la escalabilidad, flexibilidad, tolerancia a fallas y simplicidad en la administración, que pueden ser sumamente útiles en el diseño de una infraestructura para el desarrollo de sistemas de cómputo ubicuo y en particular para la arquitectura de control ubicuo que se propone en este trabajo de tesis.

Otra de las claves importantes en la implementación de un sistema ubicuo es el uso de un caso particular de una ontología, el cual es conocido como jerarquía de clases, y lenguajes basados en las ontologías (OWL), que principalmente representarán el conocimiento, definiendo formalmente los conceptos de los diferentes dominios y sus relaciones, con capacidad para realizar deducciones con este conocimiento. En un sistema ubicuo los lenguajes basados en ontologías permiten hacer descripciones semánticas de recursos y servicios.

Asimismo se pretende integrar arquitecturas de igual a igual (P2P) con una jerarquía de clases, para resolver varios puntos clave en el desarrollo de sistemas ubicuos, en específico: los procesos de inicialización, el de interacción y el de descubrimiento. Dichos procesos son ubicados como condiciones previas para lograr la “interoperabilidad espontánea”, que es uno de los problemas abiertos más importantes en cómputo ubicuo [3].

1.2 Panorama de aplicación.

Se puede considerar el caso de que un visitante llega a las instalaciones de una institución educativa para asistir a una conferencia en la Sala de Usos Múltiples. Mientras está a la espera del inicio de la plática correspondiente, el visitante tiene el deseo de usar su dispositivo móvil para acceder a Internet y consultar su correo electrónico; suponiendo que su dispositivo tiene soporte para conexiones WiFi, primeramente puede hacer la búsqueda de un punto de acceso para posteriormente revisar su cuenta de correo electrónico, véase la Figura 1.1. Hasta este punto parecería que no existe ningún inconveniente; pero a esta suposición se le agrega el hecho de que el visitante requiere imprimir urgentemente un memorándum de su jefe inmediato respecto a algunas tareas que le fueron encomendadas

durante su estancia en esta institución. Entonces surge la primera interrogante: ¿Dónde puede imprimir el documento correspondiente?, para este problema existen diferentes opciones, pero el punto de interés radica en la forma en que se puede ofrecer un servicio, o conjunto de servicios, para que el visitante no tenga que invertir una gran cantidad de tiempo y esfuerzo para realizar una tarea simple (usar un servicio de impresión).

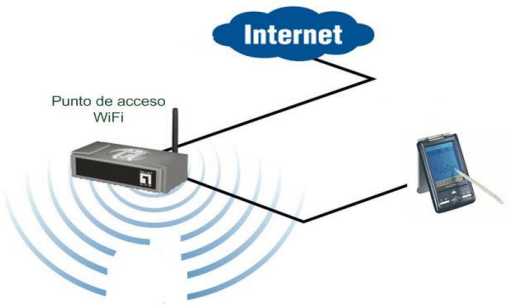


Figura 1.1 Acceso a un servicio mediante red inalámbrica.

Pineda [4] desarrolló e implementó una arquitectura (denominada ArCU) la cual permite simplificar notoriamente el uso de servicios (proyección, impresión; entre otros) mediante una arquitectura de cómputo ubicuo.

1.3 Planteamiento del problema.

En el escenario que anteriormente se ha planteado, existen diversos aspectos importantes los cuales no fueron resueltos en ArCU [4] y que en este trabajo de tesis se pretenden abordar. Dichos problemas son:

- No existe un mecanismo que permita controlar y restringir el uso de los servicios, en consecuencia cualquier usuario puede acceder al entorno de cómputo ubicuo.

- La transferencia de información se realiza empleando redes públicas, por lo cual no hay una forma de garantizar la privacidad de la información.

- Cuando existen diferentes opciones para un mismo servicio, la arquitectura no define algún mecanismo que permita recomendar un servicio en particular. En consecuencia, se requiere de un mecanismo (de preferencia basado en el contexto geográfico) que permita sugerir el uso de una opción en particular para un determinado servicio. Por ejemplo, retomando el escenario descrito anteriormente, lo ideal sería que el visitante

pudiera imprimir el documento de interés en la impresora más cercana a su ubicación geográfica.

Lo que se persigue con el presente trabajo de tesis es: mediante una integración tecnológica, desarrollar un esquema que permita crear una arquitectura de cómputo ubicuo en el cual pueda haber interacción de distintos usuarios, por medio de dispositivos móviles, con un conjunto de servicios. Lo anterior de forma rápida (empleado el entorno geográfico) y de forma segura (haciendo uso de encriptación). Inicialmente, se consideró tomar como base el trabajo desarrollado por Pineda, sin embargo, se tuvo que hacer una reingeniería completa de ArCU.

Lo anterior tuvo como justificante los cambios significativos en algunas de las herramientas utilizadas para el desarrollo de ArCU. En particular, ArCU requiere el uso de los protocolos JXTA *(Juxtapose)* versión 2.1, el cual para el intercambio de información utiliza tuberías (pipes), sin embargo, éstas están limitadas a un bloque de datos con un tamaño máximo de 64K, lo cual es sumamente ineficiente. En el Capítulo 4 se describe a detalle este inconveniente así como otros que fueron resueltos mediante una nueva versión de JXTA (versión 2.5). Este cambio en la versión de JXTA trajo diversas mejoras para la implementación de la arquitectura de cómputo ubicuo, pero debido a que se hizo una modificación considerable en JXTA, muchos de los métodos utilizados por Pineda en su versión de ArCU, fueron eliminados en la nueva versión de JXTA. Por lo anterior, se tuvieron que reprogramar la mayoría de las clases para poder hacer uso de las nuevas características de los protocolos de Juxtapose. Esta reingeniería nos permitirá lograr los objetivos que se pretenden alcanzar en esta propuesta de tesis; de forma general, la arquitectura propuesta se resume en la siguiente figura.

Figura 1.2 Esquema general propuesto.

1.4 Objetivos.

1.4.1 Objetivo general.

Diseñar e implementar una arquitectura de cómputo ubicuo orientada al contexto geográfico que permita la interacción entre un conjunto de dispositivos móviles y servicios específicos basándose en el contexto geográfico.

1.4.2 Objetivos específicos.

- Implementar una arquitectura de cómputo ubicuo basada en sistemas operativos de código abierto con requisitos computacionales mínimos. Es importante mencionar que la arquitectura propuesta puede ser usada en una plataforma computacional robusta, el objetivo de hacer uso de recursos mínimos en lo que se refiere a la relación hardware-software, es conseguir que para la implementación de dicha propuesta no se requiera de una inversión significativa, para la creación de múltiples entornos de cómputo ubicuo enfocados a las necesidades particulares de diferentes grupos de trabajo.

- Desarrollar e implementar un motor de búsqueda de servicios ubicuos considerando información de contexto geográfico.

- Desarrollar e implementar un mecanismo de autentificación de usuarios para el uso de componentes ubicuos.

- Implementar un conjunto de algoritmos de encriptación que permitan codificar la información que se intercambia a través de infraestructuras inalámbricas.

- Definir un mecanismo por medio del cual se logre recabar información de contexto geográfico.

- Realizar un conjunto de pruebas que permitan el análisis cuantitativo, para determinar la funcionalidad y alcances de la arquitectura desarrollada.

1.5 Justificación.

Existen diversos elementos que permiten justificar el desarrollo de esta investigación, los cuales, se discutirán brevemente. Primeramente, es importante enfatizar el crecimiento del uso de dispositivos móviles por parte de los usuarios. De acuerdo a cifras recientes

publicadas por la Comisión Federal de Telecomunicaciones (COFETEL), indican que tan solo en México existe un total de 100 millones de usuarios activos de dispositivos móviles, esto nos permite reforzar la idea de la importancia que han cobrado dichos dispositivos en nuestras actividades diarias.

Por otra parte, este tipo de dispositivos no son diseñados ni empleados únicamente con el objeto de servir como medios de comunicación telefónica, ya que ahora es posible navegar por Internet, reproducir contenido multimedia, capturar imágenes o video, entre otros servicios; lo cual obliga a que se cuente con mecanismos que permitan simplificar el intercambio de datos entre este tipo de dispositivos y otros medios (computadoras de escritorio, impresoras, servicios de proyección multimedia, por citar algunos).

Adicionalmente, la relevancia de este trabajo radica en el propósito de simplificar la interacción entre un conjunto de dispositivos móviles y el entorno, ya que muchas veces los diferentes tipos de tecnologías resultan ser complejas para usuarios comunes. Por ejemplo, cuando dos personas desean intercambiar un archivo de imagen o de video, la experiencia ha mostrado, que tecnologías como Bluetooth resultan ser molestas y desesperantes por la latencia en su proceso de operación: búsqueda, descubrimiento, apareamiento, sincronización, e intercambio.

Por otra parte, en lugares donde existe un alto número de dispositivos, estas tecnologías presentan serios problemas de congestionamiento. Por ejemplo, supóngase un lugar como el auditorio de una institución educativa o de un consorcio empresarial donde existan alrededor de 60 dispositivos móviles; si una persona desea enviar un archivo a otro compañero de trabajo, tendrá serios problemas para poder descubrir al otro dispositivo móvil. En consecuencia, es necesario ofrecer una arquitectura simple y de bajo costo que permita crear un efecto de comunicación directa entre dos o más dispositivos o servicios, garantizando privacidad en el envío y recepción de dicha información.

1.6 Beneficios esperados.

Se considera que los aspectos positivos del desarrollo de este trabajo de tesis se pueden resumir y describir en los siguientes puntos:

- **Simplificación de búsqueda.** Al contemplar el contexto geográfico se pueden realizar búsquedas que permitan ordenar los servicios resultantes acorde a su ubicación específica, permitiendo reducir el tiempo en que un usuario tiene que trasladarse a donde se encuentra el dispositivo que ofrece el servicio requerido.

- **Seguridad.** Al hacer uso de algoritmos de encriptación, se garantizará que los usuarios puedan transferir y compartir información confidencial, sin temor a que estos

datos puedan ser alterados o procesados por personas ajenas, evitando así el mal uso de cualquier dato que se esté transfiriendo por canales inalámbricos.

- **Control de usuarios.** Como parte de esta investigación se considera la implementación de un sistema de autentificación, con el objeto de poder restringir el acceso de los usuarios a entornos específicos.

- **Múltiples ambientes.** Por las características que ofrece la arquitectura a desarrollar se pueden proporcionar uno o más ambientes ubicuos, que pueden convivir sin ningún problema. Inclusive, se puede presentar el caso en que un dispositivo pueda acceder a más de un ambiente, o bien, que un servicio esté disponible en múltiples ambientes.

1.7 Alcances y límites.

Las principales limitaciones que se presentaron en el desarrollo de este trabajo, se describen a continuación:

- **Restricción en algunos tipos de dispositivos móviles.** Los dispositivos que no cuenten con acceso a medios inalámbricos de información no podrán hacer uso de la arquitectura propuesta, es decir, se usará solo un dispositivo con tecnología moderna.

- **Latencia de intercambio de información.** Debido al proceso de codificación y decodificación de la información, se podrán generar tiempos de espera para procesar toda la información, en especial cuando la cantidad de datos a procesar sea elevado, para lo cual, se puso especial énfasis en la optimización de los algoritmos correspondientes.

- **Falsa ubicación de dispositivos o servicios.** Para una correcta detección de los dispositivos y servicios, lo idóneo sería contar con dispositivos de posicionamiento global (GPS, *Global Positioning System*) en los teléfonos móviles, sin embargo este tipo de medios aún no ofrece una plataforma totalmente abierta lo cual limita de manera significativa su correcta manipulación además de tener un costo elevado. Por lo anterior, se propone el uso de un mecanismo basado en XML para poder especificar la ubicación geográfica de cada dispositivo o servicio. Sin embargo, de no hacer una correcta actualización en caso de desplazamiento se podrían obtener resultados no adecuados, al momento de hacer búsquedas basadas en el contexto geográfico.

- **Saturación del entorno.** Con base en una serie de pruebas, se determinará la capacidad del sistema para atender las solicitudes por parte de los dispositivos móviles, en un mal escenario.

1.8 Organización de la Tesis.

El documento se encuentra organizado de la siguiente manera:

El Capítulo 1 es una introducción del trabajo de investigación; en el Capítulo 2 se describen los principales trabajos de investigación que se han desarrollado y que están relacionados con el trabajo de tesis propuesto. En el Capítulo 3 se exponen los fundamentos teóricos así como aspectos fundamentales de temas necesarios para el desarrollo de este trabajo como son: redes ad-hoc (MANET) y P2P, la seguridad y algoritmos de encriptación. Posteriormente en el Capítulo 4 se plantea la propuesta del presente trabajo acerca del entorno de cómputo ubicuo. En el Capítulo 5 se realizan las pruebas del sistema implementado. Finalmente el Capítulo 6 describe las conclusiones, así como los trabajos futuros del presente trabajo de investigación.

CAPÍTULO 2
ESTADO DEL ARTE

Resumen: En este capítulo se describen los fundamentos sobre los que se sustenta la tecnología de cómputo ubicuo, presentando sus aspectos más importantes, así como las ventajas que ofrece esta tendencia. Posteriormente, se analizarán las propuestas más importantes que tratan de ofrecer una solución a la implementación de arquitecturas ubicuas.

2.1 Introducción.

La materialización del cómputo ubicuo, es resultado de los avances tecnológicos, que han logrado que lo que hasta hace pocos años fuera imposible de lograr, hoy sea una realidad absoluta. En su momento, fueron pocos los que lograron imaginar las posibilidades de semejante avance, siendo probablemente Mark Weiser quién más claro tuvo esta visión. En sus publicaciones, Weiser [5] planteó un cambio de paradigma, el cual es conocido como la tercera era de la computación. La primera era de la computación [6] inició con el auge de los grandes sistemas informáticos, que fueron diseñados con el fin de proporcionar servicios a múltiples usuarios, pero que se caracterizaban por tener gran complejidad de uso así como un costo demasiado elevado. Posteriormente (en la segunda era), con la aparición de las computadoras personales, cada individuo tuvo acceso a su propia computadora de escritorio. Finalmente, en la tercera era (la de computación ubicua), la relación hombre / computadora se invierte, ya que cada individuo tiene a su disposición multitud de dispositivos de cómputo, los cuales no son manipulados directamente por el usuario, ya que éstos trabajan de manera autónoma, es decir, de forma independiente del usuario.

Uno de los principales objetivos de la computación ubicua es hacer desaparecer a los dispositivos computacionales, haciéndolos situarse en un segundo plano. Este objetivo de crear dispositivos que se mezclen en la vida cotidiana hasta que lleguen a ser indistinguibles, supone una potencial revolución que puede hacer cambiar el modo de vida diario de los seres humanos. Las personas se centrarán en las tareas que deben hacer, no en las herramientas que utilizan, porque se pretende que esas herramientas pasen desapercibidas.

El significado de enviar la computación a un segundo plano, hace referencia a dos conceptos diferentes pero que tienen relación entre sí. El primero es, el significado literal de que la tecnología de la computación se debe integrar en los objetos, cosas, tareas y entornos cotidianos. Y la segunda es, que esta integración se debe realizar de forma que la introducción de capacidades de cómputo en estas cosas u objetos no interfieran con las actividades en las que son empleadas, y que siempre proporcionen un uso más cómodo, sencillo y útil de esos objetos.

Estos objetos cotidianos en los que se integra la tecnología de la computación adoptan una serie de propiedades que permiten la creación de un entorno ubicuo, siendo las más relevantes, las que se describen a continuación.

- **Comunicación entre dispositivos.** Todos estos objetos dotados de capacidad de cómputo también tienen capacidad de comunicación, y no sólo con el usuario, sino con los demás objetos integrados que haya a su alrededor.

- **Estos objetos tienen memoria.** Además de poder comunicarse entre ellos, interactuarán con los usuarios, estos dispositivos tienen capacidad de memoria y pueden utilizar esta memoria para una mejor interacción con el resto de dispositivos.

- **Son sensibles al contexto.** Los objetos son sensibles al contexto, es decir, se adaptan a las posibles situaciones, como la ubicación geográfica, los dispositivos que hay a su alrededor, las preferencias de los usuarios, y actúan dependiendo del entorno que los rodea.

- **Son reactivos.** Los objetos reaccionan al ocurrir determinados eventos, que pueden percibir en su entorno mediante sensores o a través de la interacción con otros dispositivos.

2.2 Definición de cómputo ubicuo.

Antes de presentar un concepto que permita definir formalmente el cómputo ubicuo, es importante escrudiñar dentro de dos campos estrechamente relacionados, los sistemas distribuidos [7] y la computación móvil.

- Sistemas Distribuidos: Es una colección de ordenadores totalmente autónomos conectados por una red y soportados por aplicaciones que hacen que la colección actúe como un servicio integrado.

- Computación Móvil. Es una tecnología la cual permite el acceso a recursos digitales en cualquier lugar, en cualquier momento [3].

Es gracias a la ampliación de los conceptos anteriormente mencionados que se puede dar una definición de la computación ubicua, y que se resume en el siguiente párrafo:

"La computación ubicua es aquella que involucra la integración entre nodos de cómputo y elementos del mundo físico. Desde el punto de vista del desarrollo de software, esta integración física implica que algunas veces los programas tendrán que ejecutarse en ambientes sumamente restringidos en cuanto a los recursos de hardware disponibles." [2]**.**

Este enfoque trae consigo una serie de nuevos conceptos, los cuales se enunciarán a continuación:

- **Uso eficiente de espacios.** Su objetivo es poder lograr una adecuada detección del estado de un individuo, con lo que es posible hacer una deducción de las necesidades

del mismo independientemente de su ubicación, ya sea en la oficina, sala de reuniones, salón de clase, domicilio, coche; entre otros.

- **Invisibilidad**. Uno de los requisitos para lograr un ambiente ubicuo es poder lograr la absoluta desaparición de todo lo relacionado con la tecnología, de la percepción del usuario, sin embargo, es una propiedad difícil de alcanzar, debido a que para poder lograr una invisibilidad, se necesita hacer un cambio drástico en lo referente al tipo de interfaces usadas para comunicarse con un sistema de cómputo, es decir, se deben manejar conceptos como son reconocimiento de voz y de gestos, comprensión del lenguaje natural y del texto manuscrito, en la dirección hombre-máquina y en el sentido contrario, síntesis de lenguaje hablado, escrito y de representaciones gráficas.

- **Interoperabilidad espontánea**. En un sistema de cómputo ubicuo, los componentes deben interactuar espontáneamente en ambientes cambiantes, el cual es uno de los problemas más complicados actualmente dentro de la computación ubicua. Por lo anterior, se requiere de un conjunto de protocolos que permitan a componentes de naturaleza totalmente diferente, que sean capaces de reconocerse dinámicamente entre sí, de tal manera que puedan colaborar para un propósito específico sin la necesidad de realizar la instalación de software o la modificación de algunos de los parámetros de configuración. Ver Figura 2.1.

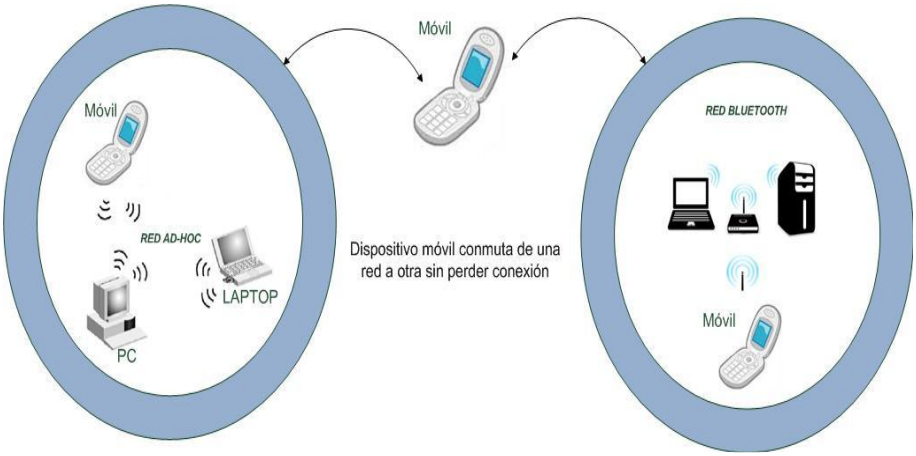


Figura 2.1 Interoperabilidad espontánea.

2.3 Requisitos tecnológicos.

A continuación se analizarán los requisitos tecnológicos (tanto de hardware como de software) que se requieren para poder embeber un entorno de cómputo ubicuo. En la Figura 2.2 se muestra el modelo seguido por la computación ubicua. De este modelo se puede resaltar que el conjunto de dispositivos que se encargan de dosificar los servicios de computación define un mundo virtual simultáneo, paralelo y reflejo del mundo real en el que se encuentran.

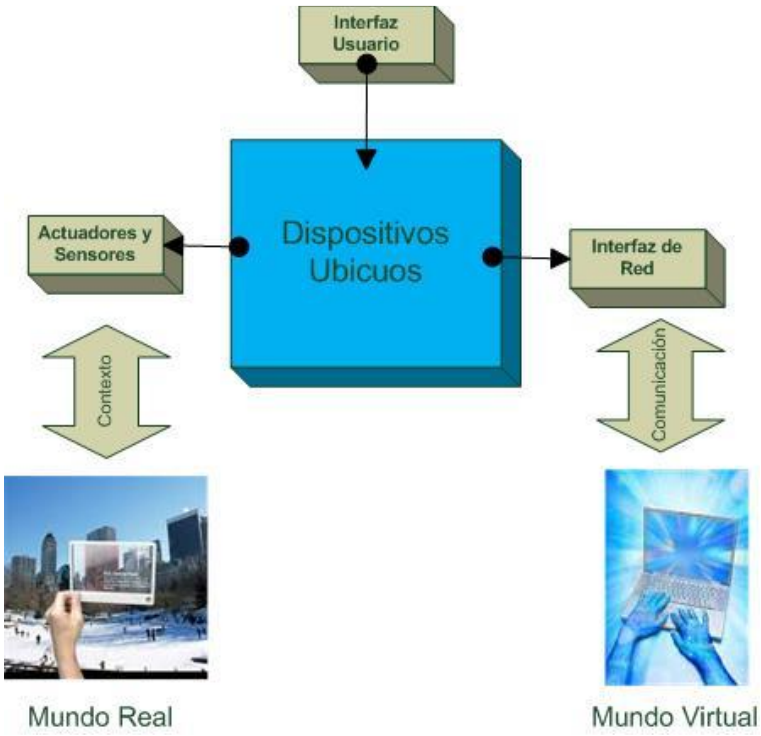


Figura 2.2 Modelo Seguido por la computación ubicua.

Fundamentalmente, son tres los requisitos impuestos al hardware que operará en el entorno ubicuo, los cuales se enuncian a continuación:

- **Miniaturización**. Para poder lograr dotar a todos los dispositivos que nos rodean, de capacidades de cómputo, así como de invisibilidad, es de suma importancia la miniaturización de los microprocesadores, lo cual se traduce como un requisito esencial en las exigencias impuestas al hardware necesario para establecer lo que se denomina como sistema embebido. Es importante resaltar que la situación actual de esta tecnología cubre las necesidades requeridas por la computación ubicua [8].

- **Baja potencia**. Uno de los principales objetivos que siempre han sido objeto de estudio en lo concerniente a los procesadores, es el de aumentar su rendimiento, de tal manera que la potencia consumida sea mínima. Para las aplicaciones que tienen que

ver con el cómputo ubicuo, el hecho de lograr un consumo bajo de energía es prioritario.

- **Conexión sin hilos**. Debido a lo mencionado anteriormente, el hecho de que la interconexión de todos los elementos constituye un entorno altamente dinámico (donde se permite la entrada y salida constante de componentes) y la incorporación de un paradigma establecido por el modelo de la computación móvil, trae como consecuencia la necesidad de establecer un control, el cual no debe de ser centralizado (comunicación sin hilos). Es aquí donde surge un gran desafío para desarrollar protocolos de conexión, mismos que engloban conceptos como son: las comunicaciones, el soporte físico y un gran ancho de banda.

En cuanto a las características con que debe contar el software de los dispositivos que conforman un ambiente ubicuo, se pueden dividir esencialmente entre aquellas requeridas para establecer las conexiones entre los dispositivos (software de soporte de la red), y las que gobiernan las acciones que tienen que ver con la interacción entre el entorno físico y el usuario.

Para dar soporte adecuado a los problemas derivados de las conexiones de dispositivos, en el cómputo ubico, es necesario abordar cambios en cuatro áreas, las cuales están involucradas con el funcionamiento de la misma red:

- **Entrada al soporte informático descentralizado y dinámico.** A diferencia de un enfoque centralizado (conexión con hilos), los dispositivos sin conexión física y en el caso de la computación ubicua, se requiere de una topología de tipo celular, ya que éstos frecuentemente entran y se desconectan de la red. Por lo tanto, no se asegura que un dispositivo pueda estar escuchando a todos los demás, lo cual puede ocasionar problemas en la detección de colisiones.

  Se han realizado varias propuestas, por ejemplo el protocolo MACA (*Multiple Access with Collision Avoidance)* de Weiser [9]) para dar solución a este problema.

- **Amplio rango de ancho de banda**. Para garantizar una trasmisión de voz e imagen con un alto grado de calidad, se requieren redes con una tasa de transferencia del orden de Gigabits[1] por segundo. Por lo tanto, es de suma importancia desarrollar nuevos tipos de paquetes de datos que utilicen los recursos de la red y se adecuen a las características de los sistemas de computación ubicua.

[1] Un **gigabit** es una unidad de medida de almacenamiento informático normalmente abreviada como **Gb** o a veces **Gbit**, que equivale a 10^9 bits.

- **Protocolos para tiempo real.** Este punto está muy ligado al anterior ya que el objetivo de ambos es el mismo, y plantea que se requiere del desarrollo de protocolos de conmutación de paquetes que permitan aplicaciones multimedia en tiempo real.

- **Enrutamiento de paquetes**. El protocolo de Internet, el IP, no provee suficiente infraestructura para dispositivos con alta movilidad. El motivo es que las direcciones están asociadas a redes estáticas, de manera que cuando un paquete debe dirigirse a un dispositivo, se supone que éste es accesible localmente a la red que pertenece. Se han realizado grandes avances para superar este problema y existen numerosas propuestas (por ejemplo; Virtual IP o Mobile IP), desarrolladas en paralelo para la utilización de redes inalámbricas.

Existe una gran variedad de redes y protocolos para el control y las comunicaciones de los dispositivos (sensores, actuadores, entre otros), en un entorno ubicuo, entre las que destacan EIB (1998) y LONWork(1999), orientadas a la automatización de viviendas, X10(1984), CEBus(1993) y SCP (*Security Cooperation Program*, Programa de Cooperación en Seguridad) de Microsoft, para el manejo de información multimedia HAVi(1998), HomePNA(1998) y especificaciones que permiten soportar redes inalámbricas como CSMA/CA bajo IEEE802.11, SWAP y el hoy en día muy popular Bluetooth.

Una parte fundamental del entorno ubicuo es la sensibilidad al contexto del usuario [10], ya que proporciona información relevante, para lograr ofrecer un soporte adecuado a las necesidades del mismo, de tal manera que el entorno se modifique de acuerdo a este conocimiento contextual, lo que implica contar con las siguientes características:

- **Reconocer el estado del usuario**. Con el fin de lograr la toma de decisiones adecuadas, el sistema debe ser capaz de reconocer el estado del usuario, en este sentido surgen la localización tanto espacial como temporal.

  Por ejemplo se puede suponer, que un profesor se dirige a una clase por uno de los pasillos de su universidad. Por otro lado, el sistema de que dispone en su cubículo recibe un correo electrónico, y a través de la velocidad con que se mueve el profesor, puede suponer si va tarde o no a su clase, de ese modo puede determinar si el profesor tiene o no tiene tiempo para atender ese correo, y en función de lo anterior, tomar la decisión de si es o no adecuado que el correo se descargue en la agenda portátil personal del docente.

- **Identificar al usuario**. Para poder cumplir con este punto se podría por ejemplo hacer uso de señales de identidad *(tags)*, las cuales serían portadas por el propio usuario, o quizás un mecanismo más sofisticado como puede ser el uso de algún sensor el cual pueda reconocer alguna característica biométrica.

- **Inferir sus necesidades**. Una vez que se ha logrado determinar el estado del usuario, es factible deducir cuáles serán sus necesidades, basándose en sus hábitos de comportamiento, así como de situaciones parecidas que le han ocurrido a él o a otros usuarios en la misma circunstancia o similar.

- **Actuar proactivamente**. Es un hecho indiscutible que un sistema de cómputo ubico debe tener la capacidad de poder llevar a cabo operaciones sobre el mundo físico, de tal forma que modifiquen el estado del usuario, y en consecuencia, afectar sus necesidades. Por otro lado, es comprensible que no todos los usuarios están dispuestos a que un sistema tome decisiones de forma transparente por ellos. Es por ello, que esta característica especial tendrá que estar estandarizada y ajustada por el mismo usuario.

En cuanto a las herramientas de software necesarias para el establecimiento de sistemas de computación ubicua es necesario que sus interfaces sean escalables, capaces de integrar una gran variedad de dispositivos de distintos tamaños y tipos de interacción, de rápido y fácil desarrollo.

2.4 Aplicaciones.

A través de Internet numerosas aplicaciones se enriquecen y amplían sus servicios como: *e-commerce, e-banking, e-education, e-goverment, e-business*. El aumento de la capacidad de las agendas digitales personales (PDA) y de los dispositivos móviles permitirá que se conviertan en asistentes personales, controlados por voz, los cuales no sólo tendrán el uso de una agenda o teléfono móvil respectivamente, sino que se convertirán gracias a numerosos sensores (biométricos, de localización, etc.) en vigilantes de nuestra salud, guías turísticos, consejeros financieros, localizadores, por citar algunos ejemplos.

Los espacios inteligentes [11], son el otro gran campo de aplicación de la computación ubicua: viviendas, salas de reuniones, consultas médicas, centros de educación y entrenamiento, centros de mando militar, gabinetes de crisis, todo tipo de vehículos, etc. Dispondrán de acceso a todo tipo de información, facilitarán el trabajo distribuido y colaborativo con otros individuos en la misma sala o de forma remota, identificarán a los usuarios y sus acciones, realizarán sumarios de las actividades y deliberaciones llevadas a cabo, por mencionar algunos.

Aunque actualmente nos podemos percatar de que Weiser [5] vislumbró acertadamente el futuro del cómputo, es un hecho que en el momento en que publicó su propuesta (década de los 90's), el paradigma del cómputo ubicuo no era posible de materializar, por lo cual, Weiser se enfocó en definir un marco conceptual y funcional para sustentar su propuesta de computación ubicua, mientras los avances tecnológicos permitieran llevar a la práctica la creación de escenarios de cómputo ubicuos, que cumplieran con las características propias de este tipo de entornos.

La actual explosión en cuanto a elementos de cómputo se refiere, parece remarcar las directrices que apuntó Weiser y el surgimiento de los nuevos híbridos postPC, con su carácter inalámbrico y presencia múltiple, lo cual anuncia que el advenimiento de la computación ubicua e invisible ya ha llegando, incluso antes de lo previsto.

Los primeros intentos de Weiser por crear escenarios ubicuos se dieron en la década de los años 90's. Sin embargo, los resultados obtenidos no cumplieron con las metas planteadas teóricamente, esto se debió en gran medida a la falta de madurez de la tecnología que existía en ese entonces. Los continuos avances tecnológicos permitieron que para el año 2005 el paradigma de cómputo ubicuo se volviera una tendencia viable, por lo que se estima que para el año 2020, se cuente con la infraestructura tecnológica necesaria para que su operación se desarrolle plenamente. Las conclusiones de Mark nos llevan a un futuro totalmente convergente, donde los dispositivos informáticos se encontrarán en todas partes y la explosión informática será de efectos globales en todos los ámbitos y niveles de la sociedad.

2.5 Arquitecturas de cómputo ubicuo.

Recientemente, diversas instituciones, centros de investigación e industrias han tomado conciencia del potencial que envuelve el cómputo ubicuo, por lo que se han realizado investigaciones, que como resultado han desarrollado sistemas donde aplican dicho enfoque. A continuación se analizarán aquellos proyectos más sobresalientes.

2.5.1 An adaptive context-aware transaction model for mobile and ubiquitous computing.

Se trata de un proyecto desarrollado por el Departamento de Ciencias de Computación e Ingeniería de la Universidad Shanghai Jiao Tong de China [12], el cual propone una arquitectura para el procesamiento de transacciones característica de los ambientes MUC (Móviles y Computo Ubicuo), cuya principal meta es subsanar los problemas que conllevan los modelos tradicionales de transacción, cliente móvil - servidor proxy, dichos problemas pueden ser resumidos esencialmente en dos puntos:

- Las estaciones base (proxy) son el requisito previo para que los dispositivos móviles (cliente) puedan hacer una transacción con los servicios del ambiente de cómputo ubicuo, teniendo como restricción el alcance de rango de la estación base.

- Algunos de estos modelos clásicos no consideran para las transacciones la gestión de la información de contexto.

Además de solucionar los problemas clásicos de otros modelos, dicha propuesta pretende dar una solución a los problemas propios de este tipo de ambientes (MUC), tales como son:

- Arquitectura para llevar a cabo el procesamiento de las transacciones.
- Modelo transaccional basado en información contextual.
- Modelo adaptativo para los constantes cambios en este tipo de entornos.

Dicha arquitectura pretende dotar a los usuarios móviles, de servicios ubicuos en línea sin importar donde se encuentren, eso quiere decir que en las transacciones hechas en el entorno MUC, no deben existir restricciones acerca del rango de alcance que envuelve a las estación base. Para concretar dicha visión, se propone que deben existir varias mallas de redes inalámbricas usando protocolos Wi-Fi, las cuales, se deben auto organizar de tal manera que cada dispositivo debe mantener múltiples conexiones con sus nodos vecinos.

En consecuencia, se propone la siguiente arquitectura, donde los dispositivos móviles que soportan los mismos protocolos de redes inalámbricas, de forma automática descubren a vecinos y se interconectan para formar una malla de red inalámbrica. Véase Figura 2.3.

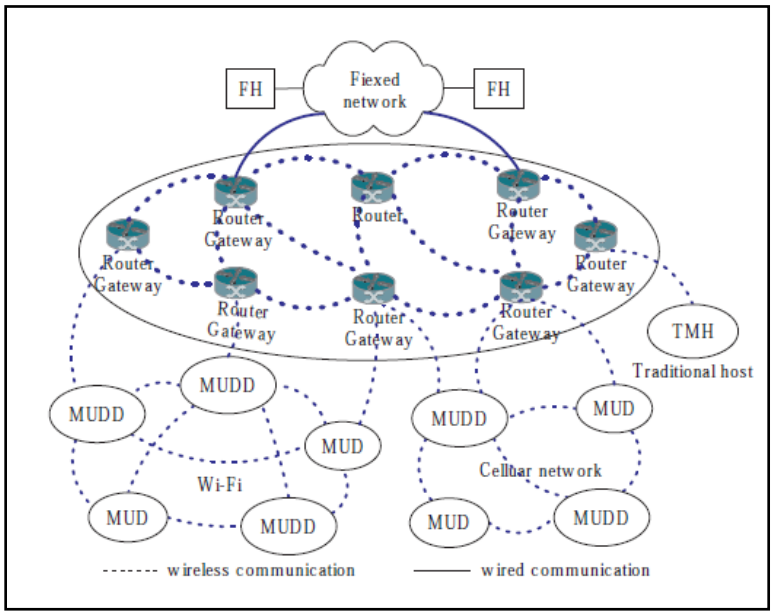


Figura 2.3 Arquitectura Propuesta para el ambiente MUC.

Esencialmente, la arquitectura de red está compuesta de tres capas: la red fija en la capa superior como columna vertebral, una malla inalámbrica de enrutamiento en la capa

media y una malla de subredes inalámbricas en la capa inferior. Cada nodo inalámbrico puede enviar datos a otro de ellos de tal manera que dos de los nodos se pueden comunicar haciendo uso de múltiples saltos. En la Figura 2.3, la línea discontinua y las líneas sólidas representan las comunicaciones inalámbricas y por cable respectivamente, mientras que las líneas más gruesas representan enlaces, los cuales tienen mayor ancho de banda. FH representa un host fijo conectado a un red fija, MUD se refiere a un dispositivo móvil ubicuo sin base de datos, mientras que MUDD a los que si tienen base de datos; TMH representa un nodo móvil sin funcionalidad en la malla. MUD (Celulares, PDA's, entre otros) y MUDD (PCs, Portátiles, entre otros.), son interconectados a través de enlaces inalámbricos para establecer mallas de red inalámbricas automáticamente y dinámicamente. Los dispositivos MUD pueden inicializar transacciones en el ambiente MUC como cliente mientras que los dispositivos MUDD, no solo pueden hacer una petición de transacción sino que también ejecutan transacciones.

En resumen, una operación de transacción en un ambiente MUC, es la que se inicia por cualquier dispositivo móvil MUD y es ejecutada por nodos móviles MUDD.

2.5.2 Context Awareness in a mobile device: Ontologies versus unsupervised / supervised learning.

Se trata de un estudio llevado a cabo en el Centro de Investigación de Nokia ubicado en Nueva Zelanda [13], dicha investigación se centra en lo que se denomina conciencia contextual del entorno, es decir, dotar a los dispositivos de cierta sensibilidad, para que a través del uso de información contextual en un ambiente de cómputo ubicuo, se adapten a los continuos cambios surgidos de las diversas necesidades y preferencias de los usuarios, de una manera automática. Esto plantea el problema de personalización, para dar solución a ello, se proponen dos posibles enfoques, mediante el uso ontologías así como del enfoque denominado SMC *(Symbol String Clustering Map)*.

En primera instancia, se habla del enfoque ontológico donde se hace uso de lógica descriptiva para abordar el concepto de razonamiento o inferencia en una ontología, y es descrito mediante dos partes representadas por T-Box y A-Box. Actores del mundo real o del dominio de interés son instanciados en elementos de A-Box. Las definiciones formales de elementos o clases en T-Box son utilizadas como mapas por los elementos de A-Box; por lo tanto, la inferencia de los elementos del mundo real es posible, y generalizada al considerar elementos del mundo real como instancias de clase en la ontología.

El enfoque SMC, trata de un algoritmo que procesa cadenas de caracteres en un tiempo secuencial y además no es supervisado. Cada símbolo de la cadena representa un estado de la fuente de información en un determinado período de tiempo. La cadena de caracteres representa la fusión de los estados de la fuente de información en un instante de tiempo dado. El funcionamiento de SMC es independiente del origen o características de la fuente

de información. Después del entrenamiento con un conjunto de datos de entrada, el resultado es un conjunto de cadenas de caracteres representativo, las cuales definen las agrupaciones en los datos de entrada. Cabe hacer énfasis en que no existe una supervisión previa y que los resultados que se obtienen debido al entrenamiento previo son totalmente dependientes de los datos de entrada que se utilizaron para realizar el entrenamiento, es decir, hay una cierta imprevisibilidad de los resultados. La estructura del SMC consiste en un conjunto de nodos, cada nodo tiene asociado una cadena de caracteres representativa del mismo y a su vez se le asigna un vector de relevancia.

Para cada símbolo de la cadena obtenido como entrada, se relaciona con el nodo que más convenga y se refiere a éste como el nodo ganador, dicho nodo representa el clúster que el carácter de entrada identificará como pertenencia. En resumen, cuando un nodo ganador recibe un símbolo, este lo interpreta como una representación de información contextual.

2.5.3 An architecture concept for ubiquitous computing aware wearable computers.

Desarrollado por el Departamento de Ciencias de la Computación en la Universidad Técnica de Múnich, esta arquitectura ofrece un panorama diferente al tradicional, en lo que se refiere a espacios inteligentes, se cimienta sobre la visión general del cómputo ubicuo, la cual pretende que diversos dispositivos computarizados se integren de manera invisible a la vida de los usuarios, en términos simples lo que el proyecto propone es que por medio de una red flexible compuesta de diversas computadoras de vestir[2] (incorporadas en la vestimenta del usuario); denominadas módulos; se puedan satisfacer las necesidades de dicho usuario de una forma natural. Cada módulo está compuesto por un procesador, memoria, dispositivos de entrada/salida, energía y una conexión a la red, y tiene como misión proveer una funcionalidad específica en la red; es decir, por ejemplo, si el usuario requiere alguna tarea que implique algún tipo de localización, deberá existir un módulo encargado del posicionamiento, el cual debe de estar conformando por un receptor GPS. Por otro lado, los servicios ofertados por un determinado módulo, pueden complementar la funcionalidad de otro módulo, para que de ese modo se pueda satisfacer alguna tarea de un orden más complejo, lo cual se logra a través del uso de un gerente de servicios, que se encarga de administrar los servicios que son ofertados por los módulos que conforman la red.

El concepto que aquí se maneja es en cierta forma innovador ya que permite lograr un alto grado de interoperabilidad entre servicios que se ofertan en la red, de ese modo se logra satisfacer ciertas necesidades para las cuales no existe de forma independiente algún servicio ofertado por un módulo capaz de resolverlo. Este enfoque no se podría aplicar en forma general para todo tipo de ambientes, ya que en algunos casos, en cierta manera, para el usuario sería incómodo el hecho de incorporar en su vestimenta este tipo de computadora de vestir, además, cabe resaltar que no se estaría cumpliendo el principal

[2] Computadoras cuyos componentes los puede llevar una persona incorporados en la vestimenta

objetivo de lograr que los dispositivos computacionales, sean integrados en la vida del humano de forma invisible.

2.5.4 WALLIP.

Esta propuesta ofrece una arquitectura ubicua, la cual permite, que diversas aplicaciones móviles sensibles a información contextual (ubicación y perfiles), contenidas en un dispositivo que portará el usuario, permitan vigilar y notificar cualquier tipo de acontecimiento que sea de interés para el usuario [15].

Cabe resaltar que esta propuesta ha sido desarrollada de forma conjunta por cinco organizaciones: *Euskaltel*, la cual proporciona el soporte en lo que se refiere a las redes de comunicaciones; *Fagor* Electrónica, responsable de las plataformas de localización; Deusto Sistemas, como responsable de la transparencia entre las diferentes redes de WALLIP, GPRS y WiFi; *Mainstrat*, como gestor del proyecto, y la Universidad de Deusto España la cual ha implementado la plataforma y sus aplicaciones.

La arquitectura WALLIP está compuesta de tres servicios principalmente, el servicio de movilidad, el servicio de gestión de perfiles y el servicio de localización, ver Figura 2.4.

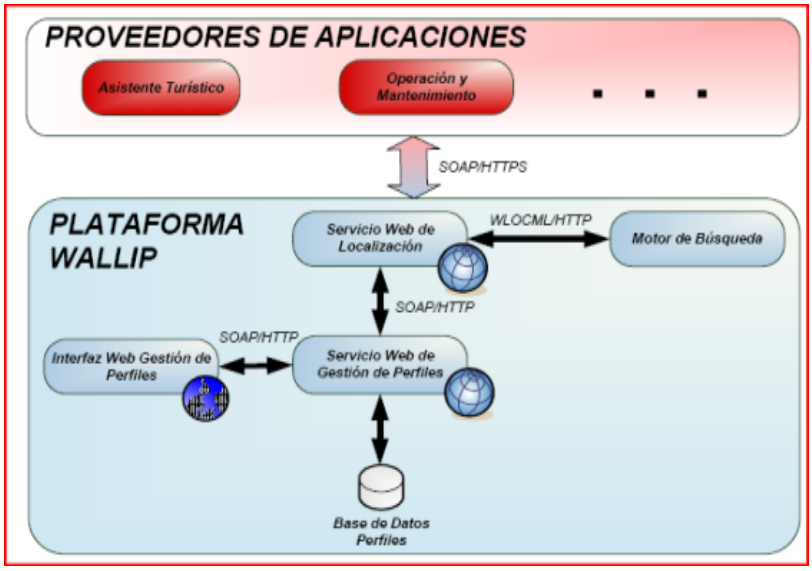


Figura 2.4 Arquitectura propuesta del proyecto WALLIP.

El abanico de posibilidades que ofrece esta arquitectura en cuanto a aplicaciones es extremadamente limitado, ya que esencialmente se centra en la manipulación de información contextual (ubicación y perfiles) de los usuarios, es decir sólo se explota una

pequeña parte de la visión que ofrece la tecnología de cómputo ubicuo, por lo cual, este enfoque desaprovecha diversas potencialidades, que podrían ser implementadas como complemento y de esa forma enriquecer la propuesta inicial. Por otro lado, se puede resaltar el hecho que a diferencia de otras arquitecturas similares, en ésta se hace uso de una forma muy elemental de perfiles, es decir se logra crear un tipo de autentificación de los usuarios, permitiendo de ese modo, discriminar el acceso al uso de las aplicaciones.

2.5.5 Servicios por identificación en el aula ubicua.

Desarrollado en la Universidad de Castilla La Mancha [16], este proyecto está dirigido a instituciones educativas y consiste en la creación de un ambiente ubicuo, el cual permite la identificación de alumnos y profesores, a través de la incorporación de pequeños dispositivos embebidos alrededor de una cierta área o aula. En esta propuesta además de los servicios de identificación se ofertan servicios entre los que sobresale el servicio de visualización (presentaciones, ejercicios, entre otros.), control de trabajos para casa, control de faltas.

La identificación se lleva acabo haciendo uso de tecnología RFID [17][18] (identificación por radio frecuencia), que permite identificar personas y objetos de manera simple, ver Figura 2.5.

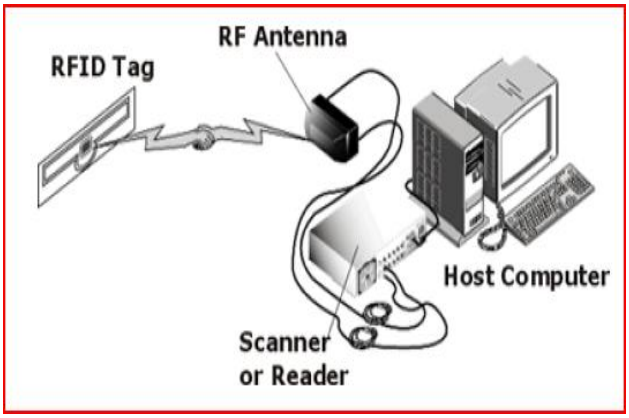


Figura 2.5 Componentes de Tecnología RFID.

El proceso de identificación se realiza colocando en el aula diversos lectores de forma estratégica (por ejemplo en la puerta), de esta manera, al llegar el usuario (que contará con una etiqueta pasiva de radiofrecuencia) será captado por el lector. A través de esta lectura, el sistema podrá obtener datos almacenados en la base de datos de los usuarios, disponiendo así de cada perfil, el cual debe ser previamente inicializado. El sistema también ayuda en el conocimiento de las tareas que se deben realizar, que puede ser adquirido a través de interacción explícita de los usuarios o bien extraída del resto de conocimientos contextuales.

Lo interesante de esta propuesta es que al hacer uso de la tecnología RFID, se puede ofrecer cierta seguridad en cuanto a lo que la privacidad de la información trasmitida se refiere, un punto que en otras propuestas no ha sido abordado. Por defecto, esta tecnología hace uso de un método de encriptación, otro aspecto positivo es que su costo es muy accesible, por ese motivo y además de la facilidad de implementación. Podemos concluir que esta propuesta es una opción muy viable para instituciones educativas que deseen implementar aulas ubicuas, en las cuales, las tareas cotidianas que son parte del modelo educativo, se puedan resolver de una manera trasparente, es decir que logren pasar desapercibidas.

2.5.6 Micro learning on a mobile device.

Se trata de una aplicación para dispositivos móviles (celulares, PDAs, entre otros), desarrollada por el MIT en Cambridge [18], la cual consiste en ayudar a los usuarios a aprender a través de eventos denominados micro aprendizajes. Lo que impulsa a dicho software son problemas que implican la tarea de memorización de información ya que se puede llegara a convertir en un proceso tedioso, en ocasiones ocupa mucho tiempo y cuando es necesario recuperar dicha información puede llegar a ser sumamente difícil. Dicho proyecto tiene su fundamento en un principio que se utiliza en el aprendizaje denominado *"micro learning"*, en el cual una tarea de aprendizaje es dividida en una serie de iteraciones de aprendizaje distribuidas.

En conclusión, al dividir el aprendizaje en intervalos regulares, en lugar de tener que aprender todo a la vez, el usuario dispondrá de un trozo de información más manejable. Las iteraciones de aprendizaje deben de ser entregadas al usuario en los momentos en los cuales son más receptivos, por ejemplo, durante el periodo de puesta en marcha del móvil. Con el propósito de demostrar el funcionamiento de dicha propuesta, se implementó una aplicación que se ejecuta en teléfonos móviles, la cual ayuda al usuario a recordar nombres y rostros. La aplicación es extremadamente fácil de usar. Cuenta con un modo denominado "nombres y caras", donde se presenta al usuario una fotografía de la cara de una persona y presenta una pregunta de opción múltiple acerca de la identidad de la persona. Por otro lado, las iteraciones con el sistema pueden ser iniciadas por el usuario, o de forma automática por el dispositivo. Aunque el sistema tiene la capacidad de iniciar interacciones de aprendizaje donde se presenta al usuario nueva información, esto se muestra en la Figura 2.6. En dicho sistema independientemente de usar rostros y nombres como base para la iteración, se cuenta con una interfaz de entrada para añadir nuevas entradas, haciendo uso de la cámara del dispositivo móvil, ya que basta con tomar una foto de una persona que conozca e inmediatamente introducir los datos sobre el nombre y asociación. Por otro lado el sistema puede comunicarse con un servidor para obtener nuevas entradas actualizadas.

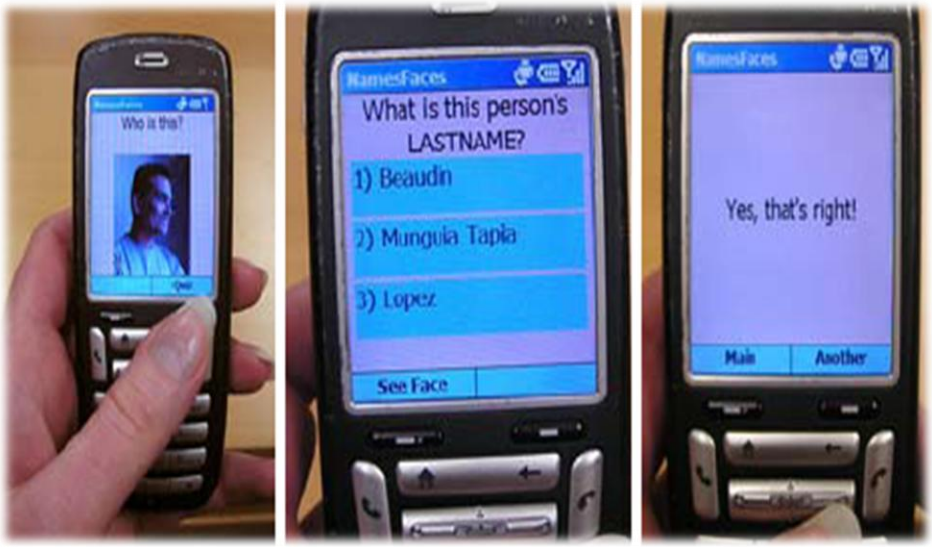


Figura 2.6 Aplicación móvil propuesta por el proyecto Micro learning.

2.5.7 Smart-Its: An embedded platform for smart objects.

Se trata de un proyecto desarrollado en la Universidad de Karslrude en Alemania [19], el cual consiste en un concepto de plataforma de cómputo ubicuo con el objeto de lograr embeber pequeños dispositivos computacionales en objetos de la vida diaria, de tal manera que se pueda lograr potencializar los objetos con capacidad de procesamiento en lo que se refiere a sensibilidad a la información contextual y comunicaciones.

Para lograr el objetivo planteado, la plataforma está compuesta tanto de elementos de hardware como de software, además de contar con interfaces de entrada, salida y redes inalámbricas. Con ello se permite la personalización de los sensores, la percepción, la sensibilidad al contexto, así como de la agregación física.

Dicho hardware presenta esencialmente 3 funcionalidades principales, integrados a su vez en dos módulos, uno para lo que se refiere a las comunicaciones y otro para las interfaces físicas de entrada y salida, cada uno de estos módulos cuenta con su propio procesador (20 MHz Arizona Microchip Processor 1687x), ver Figura 2.7.

Los módulos son interconectados a través de un bus de datos y de potencia denominado I2C. El módulo central es denominado como tarjeta de comunicaciones o tarjeta principal y es el responsable de las comunicaciones entre otros objetos computacionales o con otros servicios.

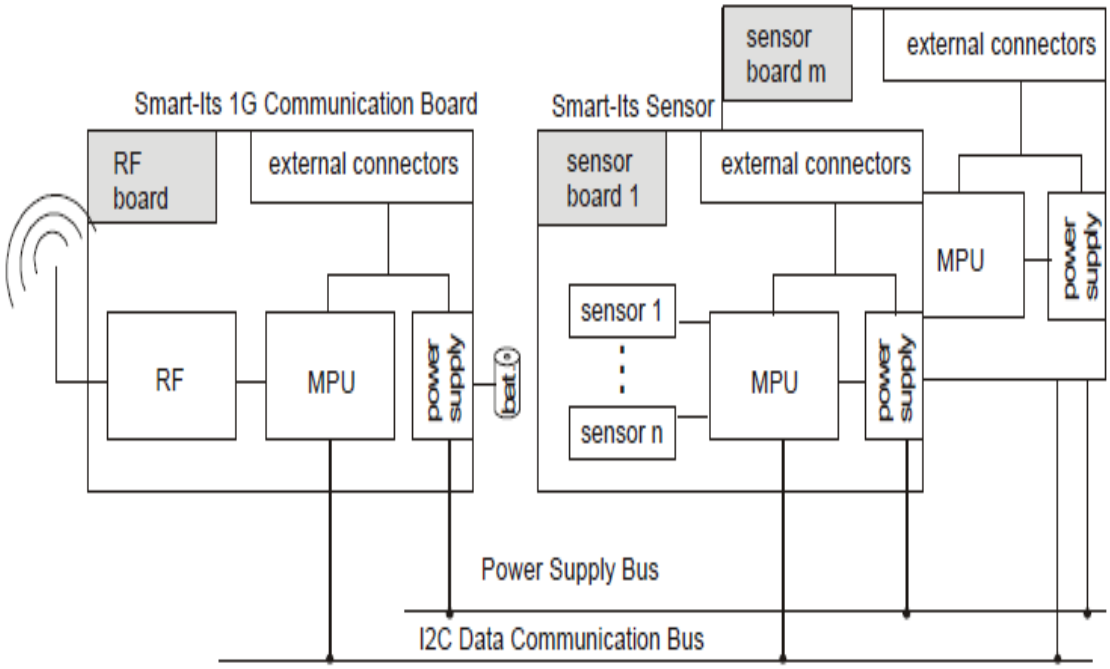


Figura 2.7 Arquitectura propuesta (Smart-Its).

La tarjeta de comunicaciones integra uno o más sensores y además una interfaz adicional es la encargada de recibir parámetros provenientes del entorno ubicuo. Las dimensiones de dicha tarjeta son 50 x 50 x 10 mm., y tiene un peso de 10 gramos.

El otro módulo, denominado como tarjeta de sensores, está compuesto de su propio microprocesador, memoria RAM para permitir el cálculo de la información contextual proveniente de los sensores de observación integrados en la misma y además permite la ejecución de código especifico de aplicación. Los sensores integrados en dicha tarjeta son para audio (micrófono y un amplificador), sensores de luz, temperatura, presión, aceleración, actuadores básicos y varias interfaces para la interconexión de sensores-actuadores adicionales.

En lo que al software se refiere, esencialmente se basa en tres bibliotecas, las cuales incluyen controladores para cada componente de hardware. Además de proveer una abstracción del hardware, se otorga un soporte sobre los sensores de manera individual, también se incluyen librerías que se encargan del control de las interfaces físicas de entrada / salida y subsistemas de comunicación. Finalmente, ofrece funciones para que de manera coordinada se pueda acceder a los recursos.

2.5.8 PUMAS (Peer Ubiquitous Multi-Agent System).

Es un proyecto que fue publicado en la Revista Colombiana de Computación en el año 2005 [20]. Consiste de un *framework* basado en agentes cuyo principal objetivo es proveer a usuarios nómadas de información relevante adaptada a diferentes criterios cuando acceden a Sistemas de Información Web (SIW) a través de sus Dispositivos Móviles (DM). Estos criterios pueden ser sus preferencias, su localización, etc. En cuanto a los SIW, estos pueden ejecutarse en uno o más servidores o, en uno o más DM.

Los agentes de PUMAS están organizados para llevar a cabo la búsqueda de la información que el usuario requiere, proveniente de uno o más SIW, y que debe ser inteligente y adaptativa. Inteligente porque debe estar basada tanto en el conocimiento propio, adquirido e inferido de los agentes, como en su capacidad de razonamiento. Y adaptativa porque toma en cuenta las características del usuario y las de su DM.

Cuando un usuario busca información, a través de su DM, sus consultas se propagan a través de los agentes de PUMAS hacia el *Router Agent*. Estos agentes de PUMAS, añaden a las consultas algunas características del usuario y de su DM para ser consideradas al adaptar la información. Por otro lado, el *Router Agent* ejecuta el proceso de enrutamiento de consultas, que consiste en la búsqueda de fuentes de información adecuadas que pueden responder a las consultas del usuario teniendo en cuenta sus preferencias, las características de su DM, su localización, etc.

La información acerca del usuario y su DM es almacenada en archivos XML específicos y el conocimiento manejado por los agentes se almacena en Bases de Conocimiento (BC). La arquitectura de este sistema se compone de cuatro Sistemas Multi-Agente: el SMA de Conexión, el SMA de Comunicación, el SMA de Información y el SMA de Adaptación.

El SMA de Conexión es la plataforma central de PUMAS y provee los mecanismos que facilitan la conexión de diferentes tipos de DM al sistema. El SMA de Comunicación garantiza una comunicación transparente entre los DM y el sistema y aplica el Filtro de Despliegue que consiste en mostrar al usuario, a través de su DM, la información de acuerdo a las restricciones técnicas de su DM. Para la aplicación de este filtro, cuenta con la ayuda de los agentes del SMA de Adaptación.

El SMA de Información recibe las consultas del usuario y hace una redirección hacia el Sistema de Información (SI) “más adecuado”, aplica el Filtro de Contenido de acuerdo al perfil de usuario en el sistema y devuelve los resultados “filtrados” al SMA de Comunicación. Para la aplicación del filtro, cuenta con la ayuda de los agentes del SMA de Adaptación.

Los agentes del SMA de Adaptación se comunican con los agentes de los otros tres SMA con el fin de intercambiar información acerca del usuario, de la conexión y de la comunicación, de las características del DM, etc. Los servicios y tareas que desempeñan consisten, esencialmente, en el manejo de los archivos XML. Los agentes del SMA de Adaptación manejan conocimiento que permite filtrar la información a los usuarios. Existe también conocimiento que es inferido a partir del análisis de la historia del usuario en el sistema, incluyendo sus últimas conexiones, consultas, preferencias, etc. Ver Figura 2.8.

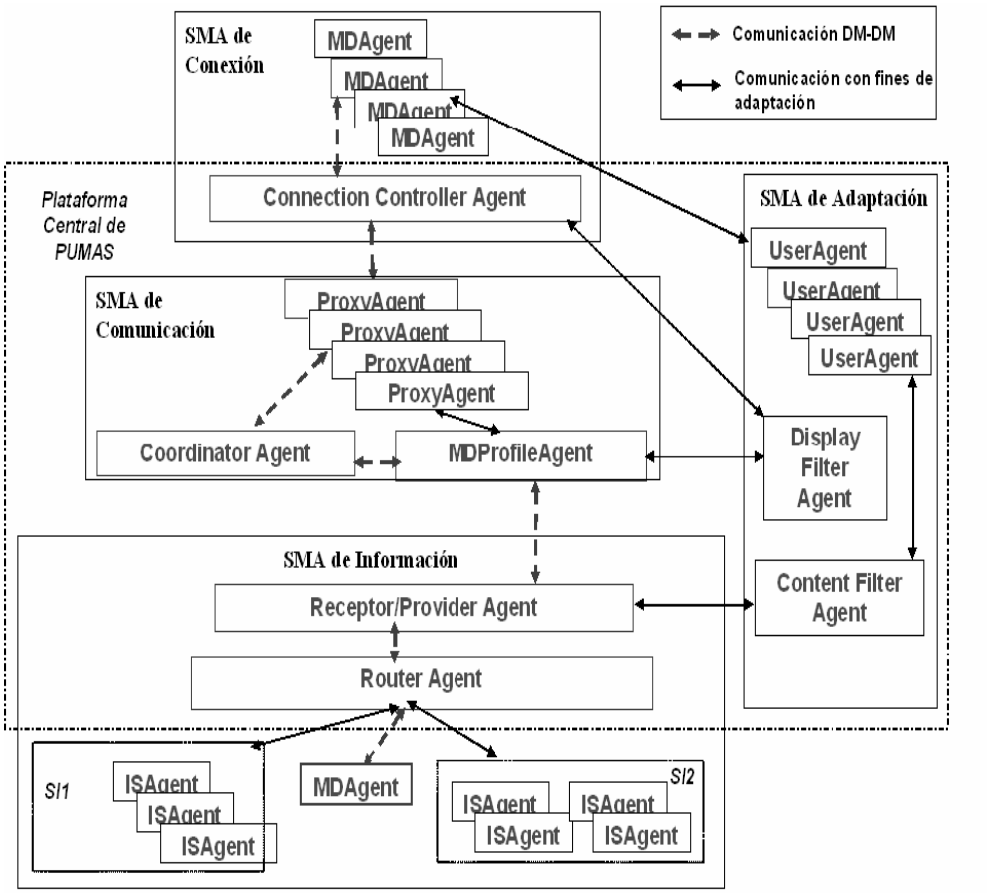


Figura 2.8 Arquitectura propuesta por el proyecto PUMAS.

2.5.9 Arquitectura para facturación y pago por proximidad de servicios ubicuos.

Es un proyecto desarrollado en la Universidad del Cauca, Popayán, Colombia, el cual propone una arquitectura para un sistema de pago y facturación de servicios aplicando técnicas de cómputo ubicuo [22].

La arquitectura de pago y facturación de servicios, ver Figura 2.9, está compuesta de los siguientes componentes: un dispositivo móvil, el cual permite al usuario realizar pagos haciendo uso de un POS (Puntos de venta) que es capaz de recibir pagos que previamente

han sido habilitados, un Administrador de Cuenta (AC) que es el núcleo de la arquitectura el cual se encarga de recibir todas las peticiones de transacciones.

Por otro lado, existe un administrador de usuario que se encarga de almacenar toda la información relacionada con las cuentas de los diferentes usuarios que son generadas por el AC, un proveedor de mensajes (PM) encargado de enviar mensajes de información al dispositivo móvil del usuario y finalmente, un servidor de correo que envía la factura digital a la cuenta de correo del usuario.

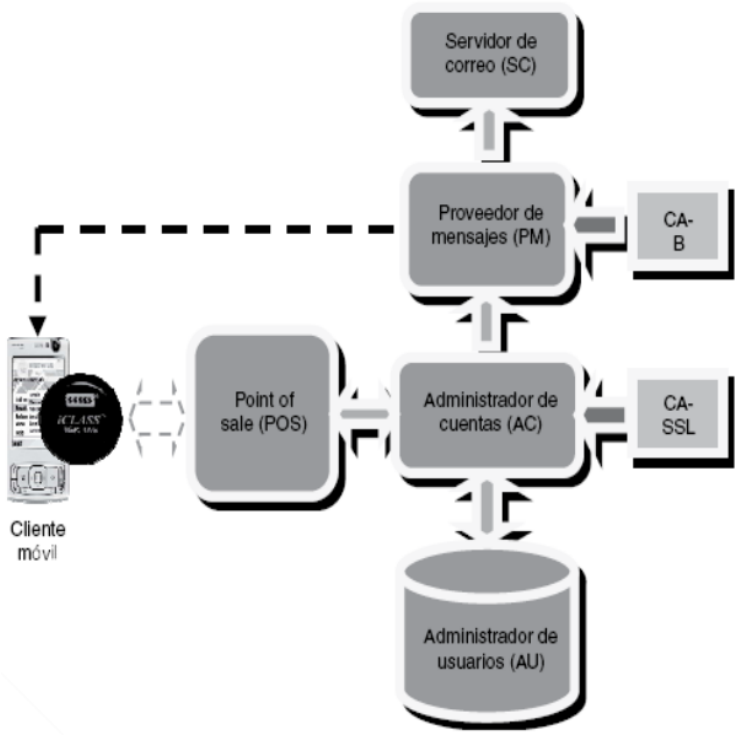


Figura 2.9. Arquitectura propuesta por el proyecto de pago y facturación.

Las características que se pueden resaltar de esta propuesta son las siguientes:

- Espontaneidad: el sistema contempla manejar el concepto de que los usuarios son altamente dinámicos e impredecibles.

- Eficiencia: con esto se asegura que el sistema de pagos sea ligero y eficiente, para poder generar confianza en los usuarios y proveedores de servicios.
- Seguridad: este punto tiene una gran relevancia, ya que por medio de un mecanismo de seguridad se evitan fraudes, como robo, falsificación, etc. Debido a que este sistema brindará servicios en cualquier lugar y hora, es más vulnerable que un ambiente controlado.

- Privacidad: el sistema cuenta con mecanismos para proteger la privacidad del usuario ya que se puede dar el caso en que un individuo no desee revelar información confidencial pero necesita realizar un pago.

- Flexibilidad: el sistema no asume ninguna configuración específica de red, dispositivo o usuario, es decir, el sistema se adapta a los continuos cambios en el ambiente.

- Funcionalidad: se refiere al grado de comodidad y a la utilidad percibida por los usuarios, en el momento de hacer uso del sistema.

- Escalabilidad: que el sistema soporte la implementación de nuevos servicios.

Como desventaja, es necesario que el dispositivo móvil pueda soportar tecnología RFID, lo cual limitaría el uso de esta arquitectura a los usuarios, ya que la mayoría de los dispositivos móviles que se usan de manera cotidiana no soportan este tipo de tecnología.

2.6 ArCU (Arquitectura de Control Ubicuo).

Es una arquitectura desarrollada por Anabel Pineda Briseño en el Centro de Investigación en Computación [4]. Este proyecto tiene como principal objetivo, establecer una arquitectura de cómputo ubicuo (ArCU) por medio de la cual, los usuarios, haciendo uso de aplicaciones que se ejecutan en dispositivos móviles o estaciones de trabajo, puedan encontrar servicios de forma espontánea, como son servicios de impresión, de proyección, entre otros, y puedan hacer uso de ellos sin la necesidad de tener conocimiento previo de los mismos, de una forma trasparente y además, tomando en cuenta el contexto geográfico.

Un punto sobresaliente de esta arquitectura es el uso de protocolos JXTA como alternativa ideal para dar soporte a la infraestructura P2P necesaria, tanto para el diseño como implementación de la misma. La ventaja principal del uso de esta tecnología (JXTA), es que sus protocolos establecen una red virtual sobre Internet, permitiendo a los dispositivos interactuar directamente y auto organizarse independientemente de su conectividad de red y dominio de topología (firewalls o NATs).

Cabe mencionar que esta propuesta se desarrolló en una versión de JXTA que actualmente es obsoleta, además de que no se consideran mecanismos de protección de la información.

2.7 Conclusiones.

Al realizar un estudio comparativo (ver apéndice E) acerca de las cualidades y deficiencias con que cuentan las diversas arquitecturas que se han analizado en este capítulo, incluyendo la propuesta que se realiza en este trabajo de tesis, tomando como base figuras demerito, aspectos cualitativos así como características técnicas, mismas que forman una parte fundamental en lo que al cómputo ubicuo se refiere, se puede concluir que los desarrollos realizados en los últimos años no cuentan con una plataforma robusta, amigable y homogénea que permita aprovechar la potencialidad que un sistema de cómputo ubicuo puede ofrecer en diversos campos de aplicación. La propuesta del presente trabajo es un esfuerzo más por aprovechar lo mejor del cómputo ubicuo, considerando para ello, cubrir las deficiencias que presentan los desarrollos comerciales.

CAPÍTULO 3
MARCO TEÓRICO

Resumen: En el presente capítulo se presentarán los elementos que de manera general sustentan el desarrollo del presente trabajo, se ofrece un resumen de los conceptos básicos relacionados con la presente investigación, como son: las redes de tipo ad hoc, los sistemas operativos de libre distribución, la criptografía y los sistemas mínimos. De tal manera que se logre proporcionar el contexto necesario para generar un lenguaje común y claro entre el lector y el documento.

3.1 Introducción.

La computación ubicua pretende establecer entornos, en donde existan dispositivos con capacidades de procesamiento y comunicaciones (teléfonos celulares, PDA, sensores, electrodomésticos, entre otros), de tal manera que puedan cooperar y comunicarse inteligentemente, tomando en cuenta aspectos acerca del entorno que les rodea de una forma totalmente trasparente hacia los usuarios. Es por ello que para poder materializar esta visión, una parte fundamental es la que tiene que ver con los sistemas de comunicación inalámbrica [1], que proporcione conexión entre los diferentes dispositivos que intervienen en este tipo de espacios, más específicamente las redes inalámbricas de tipo ad hoc, que debido a sus características tecnológicas se presentan como una opción ideal en este tipo de entornos. A continuación se presentará una definición formal, así como aquellos aspectos más sobresalientes que envuelven a esta tecnología.

3.2 Redes ad hoc.

También conocidas como redes MANETs, las redes inalámbricas ad hoc no requieren de algún tipo de estructura fija, ni administración centralizada para poder existir, este tipo de redes está conformada por múltiples nodos, de tal manera que cada nodo, además de ofrecer características propias de una estación final (host), debe proporcionar servicios de enrutamiento, con el fin de lograr retransmitir paquetes a aquellos nodos que no cuentan con una conexión inalámbrica directa, véase Figura 3.1.

Dichas redes pueden desplegarse de forma completamente autónoma o combinarse con otros puntos de acceso; además, deben poder adaptarse dinámicamente ante los cambios propios de la red, como son: los continuos cambios de topología de los nodos, la potencia de la señal, tráfico de la red y distribución [2].

3.2.1 Aplicaciones de las redes ad hoc.

Entre las principales aplicaciones que tienen las redes ad hoc, podemos enlistar las siguientes:

- **Aplicaciones Militares**, donde una configuración de red descentralizada es una ventaja operativa o incluso una necesidad.

- **Apropiada para aplicaciones de redes domesticas**, donde los dispositivos pueden comunicarse directamente para intercambiar información, tal como audio / video, alarmas, y actualizaciones de configuración.

- **Aplicaciones de mayor alcance,** en este contexto son las redes más o menos autónomas de robots domésticos interconectados que limpian, lavan los platos, cortan el césped, realizan vigilancia de seguridad, y otras labores parecidas.

- **Red de área personal**. Una MANET de corto alcance puede simplificar la conexión entre varios dispositivos móviles (tales como PDAs, computadoras portátiles, teléfonos celulares). Una red ad hoc puede extender el acceso a Internet u otras redes usando mecanismos tales como WLANS, GPRS y UMTS. La red de área personal (PAN) es un campo prometedor del uso de las MANET en el contexto de cómputo ubicuo.

Figura 3.1 Ejemplo de red de tipo Ad Hoc

3.2.2 Principales características.

- **Seguridad.** En una red inalámbrica ad hoc, la confianza es un problema fundamental. Debido a que no es posible confiar en el medio, la única elección que surge para solucionar este problema es usar la criptografía.

- **Enrutamiento de redes ad hoc**. El enrutamiento de paquetes entre cualquier par de nodos llega a convertirse en una tarea compleja, debido principalmente a que los nodos se pueden mover de manera aleatoria dentro de la red. Un camino que se consideraba óptimo en un punto dado del tiempo podría no funcionar en absoluto unos pocos momentos después. Por lo tanto una MANET debe adaptarse al tráfico así como a las condiciones de propagación como son los patrones de movilidad de los nodos móviles de la red.

- **Funciones de movilidad**. En una red ad hoc la totalidad de la red está basada en la idea de que los dispositivos sirven al mismo tiempo tanto de enrutadores como de anfitriones. En una red ad hoc, la movilidad es gestionada directamente por el algoritmo de enrutamiento.

Hasta ahora se han abordado aquellas características sobresalientes, acerca de una red tipo ad hoc, a continuación se mencionarán los estándares más importantes en cuanto a implementación de este tipo de redes, los cuales están emergiendo fuertemente, de tal forma que actualmente se ha generado un gran interés por este tipo de tecnología, en diversos campos de aplicación.

3.3 IEEE 802.11.

El estándar 802.11 también conocido como tecnología WiFi es una familia de estándares desarrollados en 1997 por la IEEE [3] *(Institute of Electrical and Electronic Engineers)*, para la tecnología de redes de área local inalámbricas, el cual esencialmente define el uso de los dos niveles más bajos de la arquitectura OSI (capa física y de enlace de datos).

En este estándar se especifica una interfaz entre el cliente y la estación base o entre dos clientes inalámbricos. Actualmente incluye seis técnicas de transmisión por modulación que utilizan todos los mismos protocolos.

3.3.1 Arquitectura del estándar 802.11.

El estándar 802.11 establece los niveles inferiores del modelo OSI para las conexiones inalámbricas que utilizan ondas electromagnéticas, que son:

La capa física ("PHY"), tiene como misión definir la modulación de las ondas de radio y las características de señalización para la transmisión, haciendo uso principalmente de tres tipos de codificación de información (DSSS, FHSS, Infrarrojo). Véase Figura 3.2.

Por otro lado, la capa de enlace de datos define la interfaz entre el bus del equipo y la capa física, en particular hace uso de un método de acceso parecido al utilizado en el estándar Ethernet, y las reglas para la comunicación entre las estaciones de la red.

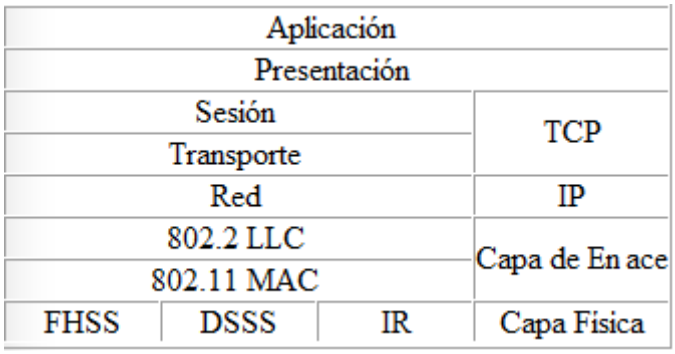


Figura 3.2 Capas del Estándar 802.11.

Esta familia ha desarrollado una serie de estándares [4], [3], además del original (802.11), como lo son: el 802.11h, 802.11i, 802.11e y otros en evolución como 802.11r, 802.11s de los que incluso existen productos comerciales pre-estándar actualmente en el mercado.

Aún así, los que más se conocen y que han sido aprobados; los cuales están en el mercado con gran éxito comercial son los siguientes:

- ***802.11b***. Su tasa de transferencia de datos máxima es de 11 Mbps y ofrece un rango de 100 metros aproximadamente en ambientes cerrados y de más de 200 metros al aire libre. Un punto importante es que no es compatible con el estándar 802.11a. Utiliza modulación con forma de onda CCK *(Complementary Code Keying)* y opera en un rango de frecuencia de una banda de 2.4 GHz [5].

- ***802.11a.*** En teoría ofrece un flujo de datos máximo de 54 Mbps, cinco veces el del 802.11b pero sólo cuenta con un rango de treinta metros aproximadamente. Este estándar está basado en la tecnología llamada OFDM (multiplexación por división de frecuencias ortogonales) [6]. Lo cual quiere decir que la información se transmite en un rango de frecuencia de 5 GHz y utiliza 8 canales no superpuestos.

- ***802.11g***. Este estándar ofrece un máximo de transferencia de datos de 54 Mbps en rangos comparables a los del estándar 802.11b. Además, el estándar 802.11g utiliza el rango de frecuencia de 2.4 GHz con codificación OFDM, gracias al uso de una interfaz de aire SS-DS que es compatible con los dispositivos 802.11b. En resumen este estándar obtiene los mejor de los dos estándares anteriores.

- ***802.11n***. Éste es uno de los estándares en evolución que surge debido a la gran demanda de las WLAN *(Wireless Local Area Network)*. Durante la segunda mitad del año 2003 la IEEE aprueba la creación del IEEE 802.11 TaskGroup N. Este grupo desarrollaría una nueva revisión del estándar 802.11, en el cual la velocidad real de transmisión podría llegar a los 600 Mbps (esto significa que las velocidades teóricas de transmisión podrían ser mayores), debería ser hasta 10 veces más rápida que una red bajo los estándares 802.11a y 802.11g, y cerca de 40 veces más rápida que una red bajo el estándar 802.11b. Además, con el desarrollo de este nuevo estándar, se espera que el alcance de operación de las redes sea mayor con la incorporación de la tecnología MIMO *(Multiple Input-Multiple Output)*, la cual permite la utilización de varios canales a la vez para enviar y recibir datos gracias a la incorporación de varias antenas.

Además de los anteriores estándares, en [5] se mencionan algunos grupos IEEE 802.11 que trabajan en diversas modificaciones del estándar.

3.4 Estándar Bluetooth.

Otro estándar que se muestra prometedor como plataforma de soporte para las redes ad hoc es la tecnología Bluetooth, debido a su habilidad para localizar de forma trasparente diversos dispositivos, así como los servicios ofrecidos por los mismos, de tal forma que esta capacidad se logra traducir en una excelente ventaja para el campo de la computación ubicua.

3.4.1 Definición de Bluetooth.

Fundamentalmente, el Bluetooth vendría a ser el nombre común de la especificación industrial IEEE 802.15.1, que define un estándar global que proporciona una vía de interconexión inalámbrica, por medio de la cual se puede lograr la transmisión de voz y datos entre diversos aparatos que tengan incorporada esta tecnología, como son los celulares, las computadoras de mano (Palm, Pocket PC), las cámaras, las computadoras portátiles, las impresoras y simplemente cualquier dispositivo que cuente con un Bluetooth integrado, usando por supuesto una conexión segura de radio de muy corto alcance [7].

3.4.2 Principales características.

La especificación de Bluetooth indica las siguientes características:

- Define un canal de comunicación de máximo 720 Kb/seg. con rango óptimo de 10 metros (opcionalmente 100 m).

- La frecuencia de radio con la que trabaja está en el rango de 2.4 a 2.48 GHz con un amplio espectro y saltos de frecuencia con posibilidad de transmitir en Full Duplex con un máximo de 1600 saltos / segundo. Los saltos de frecuencia se dan entre un total de 79 frecuencias con intervalos de 1 MHz; esto permite dar seguridad y robustez.

- La potencia de salida para transmitir a una distancia máxima de 10 metros es de 0 dBM (1 mW), mientras que la versión de largo alcance transmite entre -30 y 20 dBM (100 mW).

- Para lograr alcanzar el objetivo de bajo consumo y bajo costo, se ideó una solución que se puede implementar en un solo integrado utilizando circuitos CMOS. De esta manera, se logró crear una solución de 9x9mm y que consume aproximadamente 97% menos energía que un teléfono celular común.

- El protocolo de banda base (canales simples por línea) combina switching de circuitos y paquetes. Para asegurar que los paquetes no lleguen fuera de orden, los slots pueden ser reservados por paquetes síncronos, un salto diferente de señal es usado para cada paquete [8].

- Bluetooth utiliza un esquema de funcionamiento orientado a conexión y basado en una configuración amo-esclavo en el cual, un único amo coordina el acceso al medio de hasta 7 dispositivos esclavos empleando un *polling scheme* para determinar el orden en el cual los esclavos envían y reciben datos.

3.4.3 Arquitectura Bluetooth

Fundamentalmente el mecanismo que conforma la tecnología Bluetooth consta de dos partes: Un dispositivo de radio, encargado de modular y transmitir la señal y un controlador digital que está integrado por un CPU, un procesador de señales digitales (DSP, *Digital Signal Processor*) llamado *Link Controller* y de las interfaces con el dispositivo anfitrión [9]. Bluetooth cuenta con una serie de protocolos entre los que destacan los siguientes:

- **Standby.** Cuando los dispositivos están en modo de reposo ellos escuchan mensajes cada 1.8 segundos sobre 32 saltos de frecuencia.

- **Page / inquiri**. Permite el envío de un paquete denominado page que permite realizar la conexión con otro dispositivo, y si el receptor de este page contesta se comienza con la transferencia de datos.

- **Active**. Permite la transmisión de datos.

- **Hold**. Permite realizar la conexión sin necesidad de transferir datos. La finalidad de esto es conservar el control entre el maestro *(master)* y el esclavo *(slave)*, siempre y cuando así se desee.

- **Sniff**. Esta técnica solo es aplicada a unidades tipo esclavo y permite conservar el control, durante este modo el esclavo no toma un rol activo pero escucha a un nivel reducido.

- **Park**. Este es un modo más reducido que el modo *hold*, durante el cual el esclavo es sincronizado a la *piconet*, lo cual permite que no se requiera una reactivación completa, y no es parte del tráfico.

3.4.4 Versiones de Bluetooth.

A continuación en la Tabla 3.1 se muestra una descripción de cada una de las versiones del estándar de Bluetooth.

Tabla 3.1 Descripción de cada una de las versiones del estándar Bluetooth.

Versiones	Descripción
1.1	Carecía de un mecanismo que permitiera la compatibilidad del Bluetooth con WiFi.
1.2	A diferencia de la 1.1, cuenta con una solución inalámbrica complementaria la cual permite la coexistencia entre Bluetooth y WiFi en el espectro de los 2.4 GHz, es más segura y ofrece mejor calidad de audio.
2.0	Esta versión incorpora la tecnología *Enhanced Data Rate* (EDR), la cual permite el aumento de la velocidad de trasmisión hasta 3Mbps, así como también corrige diversos fallos de la especificación 1.2.
2.1	Las mejoras en esta versión consisten esencialmente en facilitar la conexión entre equipos y un ahorro de energía cinco veces mayor.
3.0	Permite la transferencia de archivos de mayor tamaño gracias a la incorporación de la tecnología 802.11n, así como también proporciona un mayor rendimiento en lo que a la trasferencia de datos se refiere, ya que se puede conseguir una tasa de transferencia de hasta 24 Mbps.

3.4.5 Comentarios finales.

Las numerosas ventajas tecnológicas con las que cuentan los estándares 802.11 y Bluetooth, otorgan un amplio abanico de características, las cuales permiten resolver una serie de requerimientos, en cuanto a comunicaciones se refiere, al momento de implementar aplicaciones de cómputo ubicuo. Por lo anterior, el diseño de la arquitectura propuesta en esta tesis (que será presentado formalmente en capítulos posteriores) está cimentado en estas tecnologías.

3.5 Sistemas P2P.

En esencia, lo que se pretende lograr en un sistema de cómputo ubicuo es que la computadora, que tradicionalmente existe como un artefacto integrado de múltiples tareas, se desintegre en una serie de dispositivos autónomos de red, orientados a tareas particulares con el fin de satisfacer diversas demandas de forma trasparente. Por lo tanto, una tecnología que nos proporciona el marco perfecto de aplicación para el desarrollo de infraestructuras de cómputo ubicuo, es la de los sistemas de igual-a-igual (P2P, *peer to peer*), la cual ofrece una serie de ventajas para los ambientes ubicuos, tales como el balanceo de carga automática y la auto organización. Pero la característica más importante de estos sistemas, es que debido a la naturaleza simétrica de par a par, el sistema puede ajustarse a las propiedades deseables cuando nuevos nodos son añadidos a la red. Esas propiedades incluyen desempeño, disponibilidad y tolerancia a fallas [10].

A diferencia del modelo cliente servidor, los sistemas P2P intentan ser descentralizados, auto organizados con respecto a un contexto externo, y con un enfoque de recursos balanceados. Por lo tanto, cada nodo en el sistema puede proporcionar ambas funciones tanto de cliente como de servidor. Un usuario puede tener acceso fácilmente a una red *peer to peer* desde una computadora de escritorio o desde un dispositivo móvil sin alguna complejidad extra e independientemente de la localización física. De ahí la integración de P2P en aplicaciones de trabajo en grupo donde los participantes puedan interactuar sin dificultad con el sistema y con los demás participantes. Debido al amplio panorama que esta la tecnología (P2P) ofrece, en lo referente a aplicaciones ubicuas, en la siguiente sección se realizará una revisión más detallada de la misma.

3.5.1 Enfoque P2P.

Como se ha mencionado, el enfoque que persiguen las arquitecturas P2P es el de compartir contenidos, los cuales pueden tratarse de recursos, archivos de diverso índole o simplemente servicios los cuales son representados como nodos. Este enfoque se lleva a cabo, sin el requerimiento de un ente central, en otras palabras cada nodo se puede comportar como cliente (contribuyendo con contenidos), servidor, o bien, como enrutador.

Con esto se asegura que si un nodo no estuviera disponible, la arquitectura seguiría funcionando ya que el funcionamiento no radica en un nodo central, tal como se maneja en una arquitectura cliente/servidor, en donde todo el éxito radica fundamentalmente en el correcto funcionamiento del servidor.

3.5.2 Arquitectura.

Principalmente existen 4 tipos de arquitecturas P2P [11], las cuales principalmente se diferencian en el mecanismo de búsqueda de nodos activos y contenidos.

P2P puro.

En este tipo de implementación cuando un nodo requiere información de otro nodo, este último funciona como servidor, sin embargo, si el nodo sólo requiere información de otro nodo éste toma el papel de cliente, y por último si el nodo sólo actúa como un intermediario éste se estará desempeñando como nodo enrutador.

En cuanto a la búsqueda de un nodo en este tipo de arquitectura usa diversos algoritmos para realizarla, por ejemplo, puede usar una lista de todos los nodos así como de los servicios que ofrecen o simplemente puede enviar un mensaje de Multicast o Broadcast a toda la red (Ver Figura 3.3).

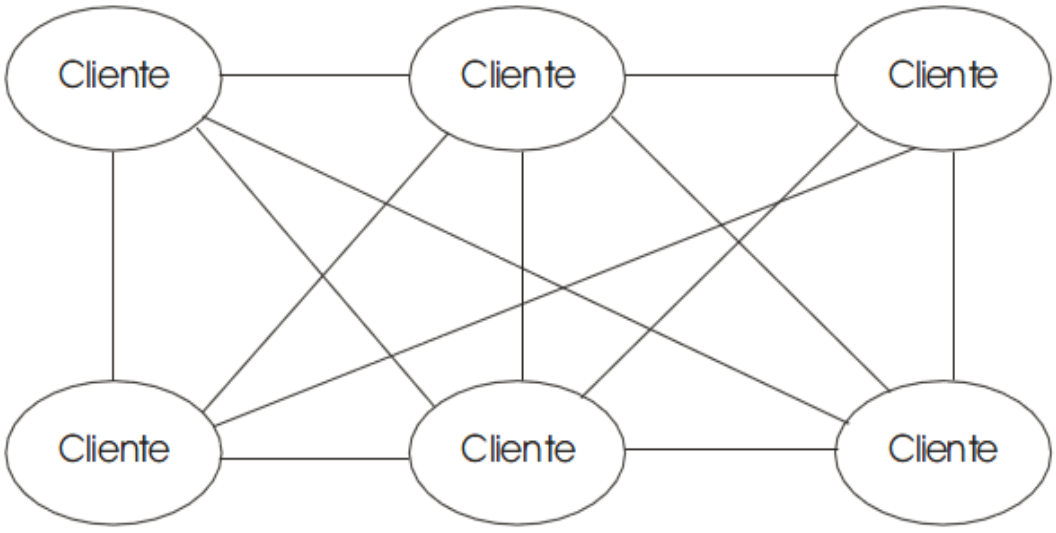


Figura 3.3 Arquitectura P2P del tipo puro o descentralizada.

P2P con servicio de consulta de nodos.

En este tipo de arquitectura un nodo realiza una consulta a un servidor donde puede revisar los nodos que están activos, y luego de consultar esa información el nodo podrá establecer una conexión de manera directa a otro nodo para poder compartir recursos. Cabe hacer hincapié que la aplicación P2P tendrá la tarea de mantenerte informado al servidor acerca de la conexión y desconexión de los nodos para mantener íntegro el servicio (Ver Figura 3.4).

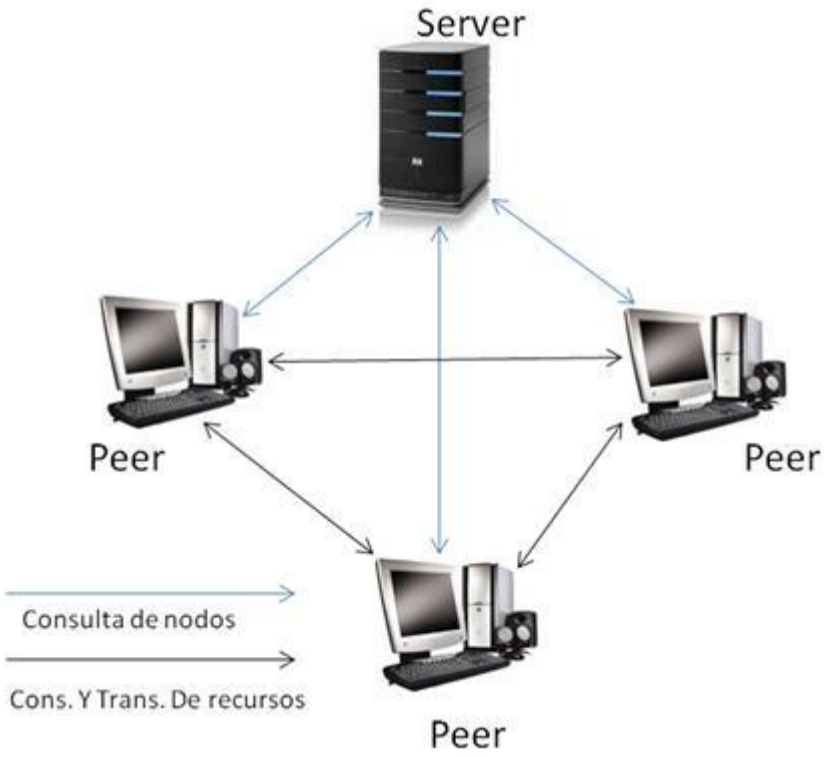


Figura 3.4 Arquitectura P2P con Servicio de consulta de nodos.

P2P con servicio consulta de nodos y recursos.

Este modelo implementa, al igual que P2P con servicio de consulta de nodos, un equipo central que tendrá la misión de almacenar los nodos activos pero con la diferencia de que también existirá información de los contenidos que éstos comparten.

P2P con servicio de consulta de nodos, recursos y fuentes de contenidos

Aquí, un equipo central cumple dos misiones. La primera es, tener almacenados los nodos activos y los contenidos que comparten. La segunda, almacenar contenidos para compartir con los nodos conectados. Ver Figura 3.5.

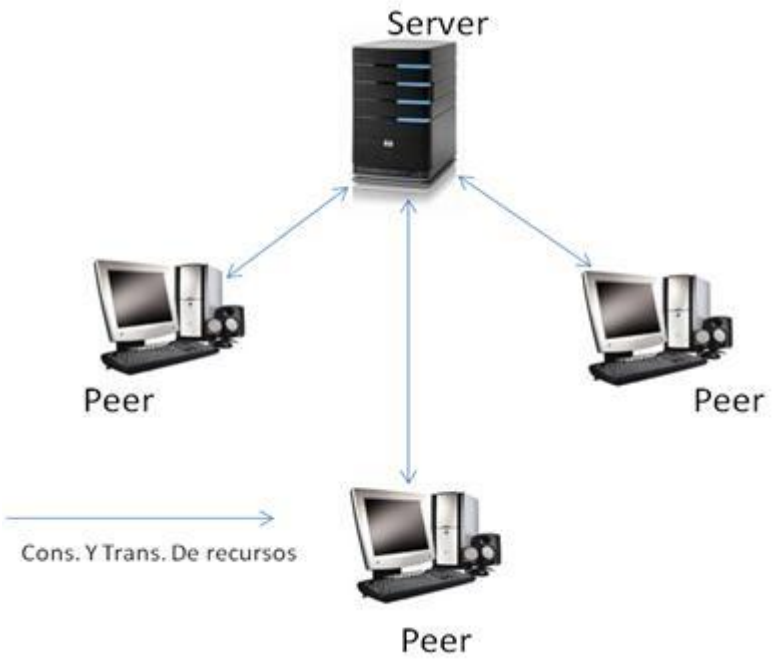


Figura 3.5 Arquitectura P2P, recursos y fuentes de contenidos.

3.5.3 Funcionamiento.

En la arquitectura P2P Pura principalmente existen tres acciones básicas [12]:

- La búsqueda de nodos activos.
- La consulta.
- La transferencia de contenidos.

En forma general la manera de operar es la siguiente; primero para buscar nodos activos, el nodo emite un mensaje en forma Broadcast o Multicast a la red. Éste a su vez debe ser respondido por los nodos activos con otro mensaje de respuesta. Inmediatamente después de que recibe respuesta el nodo anunciará una consulta, la cual deberá ser respondida por aquellos nodos que posean los recursos que han sido solicitados. Por último, el usuario podrá seleccionar los recursos que serán trasferidos.

La arquitectura P2P híbrido también la conforman tres acciones básicas [12]:

- Registro.
- Consulta.

- Transferencia de contenidos.

En primera instancia para llevar a cabo el proceso de registro, la aplicación P2P debe indicar al nodo que esté prestando el servicio de consulta que está activo, así como enviar la información sobre los contenidos que puede compartir, de tal forma que cuando una consulta es recibida por el nodo que presta el servicio de consulta, se procesará. Ésta se lleva a cabo, dependiendo de la arquitectura, mediante la búsqueda en índices o enviando la consulta a los nodos conectados. Por último, la respuesta será enviada al nodo para que el usuario pueda seleccionar los recursos a transferir.

3.5.4 Características de P2P.

Descentralización: De manera global lo que conlleva este término es lo siguiente [13]:

- Permite devolver el poder y autonomía a los nodos en el borde de la red.
- Realiza el manejo de conexiones variables a direcciones provisionales.
- Está dotado de cierta inteligencia distribuida.

Aprovecha el ancho de banda de los usuarios, esto significa que [14]:

- Lo utiliza completamente.
- El ancho de banda total crece con el número de nodos.
- Los clientes no tienen que compartir el ancho de banda limitado de un servidor.

Cuenta con un mecanismo que le permite hacer distinción entre publicar y ser autor.

3.5.5 Áreas de implementación.

Entre las áreas donde se pueden implementar sistemas P2P podemos nombrar las siguientes [15] [16]:

- **Comunidad Web.** Cualquier grupo con intereses en común, pueden usar P2P para intercambiar recursos.

- **e-Business.** P2P puede dotar nuevas posibilidades, como intercambio más eficiente de información, ya sea con proveedores o con clientes.

- **Motores de búsqueda.** Ofrecer información más actualizada.

- **Protección de virus.** Permite una colaboración en la detección y eliminación de virus.

- **Educación a distancia.** Permite una interacción entre los participantes permitiendo el intercambio de sonido y video.

- **Almacenamiento y procesamiento distribuido.** Múltiples clientes ofrecen un espacio de almacenamiento más flexible y fiable.

3.5.6 Plataformas.

Actualmente existen múltiples plataformas para dar soporte a sistemas P2P y de entre ellas sobresalen principalmente:

- **Microsoft .NET**. Esta plataforma proporciona un marco de trabajo *(framework)* que permite el acceso a servicios web, haciendo uso de protocolos tales como XML *(Extensible Markup Language)*, UDDI *(Universal Description Discovery and Integration protocol)* y WSDL *(Web Services Definition Language)*. Con ello se pretende proporcionar el acceso de múltiples dispositivos de forma descentralizada, en todo momento siempre dirigido al mundo de Internet [17].

- **JXTA de Sun MicroSystems**: JXTA provee programación de red abierta de propósito general y plataformas de cómputo para aplicaciones distribuidas ejecutándose en una variedad de dispositivos. Los tres componentes asociados con JXTA son JXTA Core, JXTA Services, y JXTA Applications [18].

3.6 JXTA.

En específico, esta plataforma nos proporciona un conjunto de características y servicios, los cuales se encargan del descubrimiento, autentificación y administración; así como otros más que incluyen identificación, encaminamiento e indexación. En lo que se refiere a sistemas P2P, nos proporciona un marco de aplicación para poder dar solución a las problemáticas impuestas por las características particulares del cómputo ubicuo; a continuación se tratará más a fondo esta plataforma [18]. JXTA estandariza un conjunto común de protocolos para construir redes virtuales P2P. Así mismo, los protocolos definen los requerimientos mínimos para formar y unirse a redes P2P, así como la forma en que se comunicarán los elementos involucrados en la misma red P2P.

3.6.1 En qué consiste JXTA.

Los protocolos JXTA permiten crear una clase de red virtual por encima de la infraestructura de la red física, de tal forma que cada uno de los iguales que conforman la red, puedan intercambiar mensajes con cualquier otro igual, sin preocuparse de algún modo por su localización física en la red, esto dota a los iguales de tipo móvil de capacidad de desplazamiento libre en la red. Los mensajes intercambiados son transparentemente ruteados, atravesando firewalls o NAT, y pueden ser implementados sobre diferentes protocolos, como pueden ser TCP/IP o HTTP, para alcanzar un igual destino. En JXTA se define un estándar para establecer la manera en la cual los iguales se descubren, se auto organizan, descubren recursos en la red y la forma en que se comunican entre ellos.

La abstracción de la red virtual se divide principalmente en 5 abstracciones por medio de las cuales se implementara una red JXTA, las cuales son:

1. Un modelo de direccionamiento lógico de los iguales, también denominados peer que se expande sobre la red.

2. Los grupos de iguales *(peergroups)* que permiten a los nodos dinámicamente auto organizarse en un dominio virtual protegido.

3. Los anuncios para que los iguales publiquen sus recursos.

4. Un mecanismo de enlace para desempeñar todas las operaciones de enlace requerido en los sistemas distribuidos.

5. Las tuberías *(pipes)* las cuales son los canales de comunicación que permite a los iguales comunicarse.

3.6.2 Iguales *(Peers).*

Un igual es cualquier entidad de la red que implementa uno o más de los protocolos de JXTA. Los iguales pueden residir en sensores, teléfonos, PDA, así como también como en servidores y computadoras de escritorio. Cada igual opera independientemente y asíncronamente de todos los otros iguales y es identificado únicamente por un identificador (ID) de igual [18].

Los iguales publican una o más direcciones de red para usarlas con los protocolos JXTA. Cada dirección publicada se anuncia como un punto final *(endpoint)* del igual, que

identifica la dirección de red. Los puntos finales del igual son usados por los otros iguales para establecer directamente conexiones punto-a-punto entre dos iguales.

Las conexiones de red punto-a-punto directas no siempre están disponibles entre iguales. Iguales intermediarios pueden ser utilizados para encaminar (enrutar) mensajes a los iguales que están separados debido a los límites físicos de la red. Los límites de la red pueden ser límites naturales, por ejemplo, entre redes de Ethernet y de Bluetooth o creadas artificialmente debido a la configuración de la red. Las barreras artificiales pueden incluir NAT, cortafuegos y servidores proxy. El uso de iguales intermedios alistados puede y cambiará en un cierto plazo sin impacto en la aplicación de JXTA. Configuran a los iguales típicamente para descubrirse unos a otros espontáneamente en la red para formar las relaciones conocidas como grupos de iguales, que pueden ser transitorios o persistentes en naturaleza.

Los iguales JXTA pueden ser categorizados dentro de tres tipos principales:

- **Minimal-Edge peers**: Iguales que implementan únicamente los servicios requeridos por el núcleo *(core)* de JXTA y pueden contar con otros iguales para que actúen como su proxy para otros servicios para participar completamente en una red JXTA.

- **Full-Edge peers:** Iguales que implementan todos los servicios del núcleo y estándar de JXTA y pueden participar en todos los protocolos JXTA. Estos iguales conforman la mayoría de iguales en una red JXTA y pueden incluir teléfonos, PC, servidores, etc.

- **Super-Peer**: Iguales que implementan y proveen recursos para soportar el despliegue y operación de una red JXTA. Hay tres funciones principales en un super-peer. Un igual simple puede implementar una o más de estas funciones.

    - Relay: Usada para almacenar y reenviar mensajes entre iguales que no tienen conectividad directa debido a la presencia de un cortafuegos o NAT. Solamente los iguales que no están en capacidad de recibir conexiones de otros iguales requieren un relay.

    - Rendezvous: mantiene los índices globales de publicidad y ayuda a iguales extremos *(edge)* y proxys con el anuncio de búsquedas. También se ocupa de la difusión de mensajes.

    - Proxy: Utilizado por los minimal-edge peers para tener acceso a todas las funcionalidades de la red JXTA. El peer proxy, traduce y resume las peticiones, responde a las consultas y proporciona soporte para la funcionalidad de los minimal-edge peers.

3.6.3 Identificadores (ID) en JXTA.

En lo que respecta al direccionamiento, JXTA está basado en un modelo de direccionamiento uniforme e independientemente de la localización, es decir, a cualquier recurso en la red (peer, pipe, peergroup, etc.), se le asigna un identificador único denominado JXTA ID. Los JXTA ID son objetos, los cuales permiten una representación de un ID (IPv6, MAC) para coexistir en la misma red JXTA. La referencia usada para esta implementación es UUID *(Universal Unique Identifier)* que son generados aleatoriamente y tienen una longitud de 128 bits, permitiendo a cada peer generar su propio ID. De este modo, un peer en la red solamente puede ser identificado por su peer ID de tal forma que el peer puede ser direccionado independientemente de su dirección física. Así mismo, un peer puede soportar múltiples interfaces de red (Ethernet, Wi-Fi, etc.) y será direccionado como un único peer independientemente de la interfaz usada [19].

3.6.4 Anuncios en JXTA.

Todos los recursos en el proyecto JXTA, son representados por anuncios *(advertisements)*. Los anuncios son un recurso que es representado como documentos XML. En JXTA se estandarizan los anuncios de los recursos principales como son: los peers, los peergroups, los pipes, los servicios, etc. Aunque los desarrolladores pueden crear sus propios anuncios, los peers publican e intercambian anuncios para descubrir y encontrar recursos en la red. Todos los anuncios son publicados con un tiempo de vida que especifica la duración de los anuncios sobre la red [18].

3.6.5 Mensajes en JXTA.

Los mensajes son un mecanismo básico de intercambio de información entre los *peers*, es decir, los *peers* interactúan enviando y recibiendo mensajes. Los mensajes pueden estar en formato XML o binario, ambos formatos pueden ser usados dependiendo de la situación.

3.6.6 PeerGroups.

Un peergroup representa un conjunto dinámico de peers que tienen un conjunto común de intereses y además están integrados por un conjunto común de políticas (membresía, intercambio de contenido, etc.). Cada uno de los peergroups es identificado únicamente por su ID. Los peergroups pueden ser creados dinámicamente por los usuarios, por los desarrolladores o por los administradores de red para separar el ámbito de interacción entre los peers.

3.6.7 Tuberías.

Las tuberías *(pipes)* son canales virtuales de comunicación usadas para enviar y recibir mensajes entre los servicios y las aplicaciones. Las tuberías proporcionan una abstracción virtual sobre los peers para proporcionar un tipo de entradas y salidas que no por el hecho de ser virtuales están físicamente limitadas a la localización del peer. Las tuberías pueden conectarse a uno o más peer receptoras.

Los pipes son publicados y descubiertos usando un anuncio de tubería y son identificados únicamente por su pipeID. El proceso de enlazado de un pipe consiste en buscar y conectar dos o más tuberías. Cuando un mensaje es enviado a una tubería, el mensaje es transmitido por la tubería de salida local a la entrada de una tubería que está escuchando. Por lo regular las tuberías ofrecen dos modos de comunicación:

– Una tubería de conexión punto a punto, que son dos tuberías conectadas una a la otra con un canal unidireccional y asíncrono. No existe alguna operación de retransmisión o reconocimiento de que llegó el mensaje. El mensaje puede contener un anuncio de tubería que puede ser usado para abrir una nueva tubería para responder a la tubería que le envió el mensaje.

– Una tubería de propagación la cual conecta una salida a múltiples tuberías de entrada [18].

3.6.8 Sockets.

Los sockets de JXTA son conexiones bidireccionales confiables usadas por las aplicaciones para comunicarse de manera segura.

3.6.9 JXME.

Fundamentalmente consiste en el desarrollo del proyecto JXTA para J2ME, cuya meta es proporcionar compatibilidad de los servicios de JXTA para dispositivos móviles, de tal manera que haciendo uso de los protocolos implementados por JXME cualquier dispositivo MIDP *(Mobile Information Device Profile)* podría participar en una red P2P [18].

3.6.10 Principales objetivos.

Dentro de los principales objetivos de JXTA se tienen los siguientes:

- Poder ser compatible con cualquier componente que conforme la red JXTA.
- Permitir que su implementación sea fácil y transparente.
- Que no se requiera una gran cantidad de recursos para operar debido a las limitaciones impuestas por los dispositivos móviles.

Antes de comenzar a introducirnos más a fondo en el mundo de JXME es necesario hablar un poco de J2ME *(Java 2 Micro Edition)*, la cual es una especificación de un subconjunto de la plataforma Java orientada a proveer una colección certificada de API *(Application Programming Interface)* de desarrollo de software para dispositivos con recursos restringidos, en ella se incluyen clases de API: CLDC (Configuración de Dispositivos de Conexión Limitada) y la MIDP (Perfil para Dispositivos de Información Móvil), esta última nos proporciona una serie de herramientas necesarias para desarrollar aplicaciones en entornos restringidos como podrían ser los dispositivos celulares.

La configuración CLDC opera en dispositivos con microprocesadores RISC/CISC ya sea con 16 o 32 bits y al menos de 160 KB de memoria total. El perfil MIDP es un conjunto de API de Java que está orientada a dispositivos móviles.

Para que el dispositivo móvil logre ser compatible con una red JXTA, es necesario subsanar sus restricciones, principalmente son las siguientes:

- El uso de librerías es limitado.
- Limitaciones para el análisis de documentos XML.
- Sólo soporta comunicación HTTP.

Es por ello que el proyecto JXTA establece JXME para dar soporte a ello.

3.6.11 JXME Relays.

Debido a las limitaciones de los dispositivos móviles, éstos solo pueden fungir como iguales de tipo edge. En otras palabras, no tienen la capacidad para ofrecer servicios a otros miembros que conformen un peergroup. Por ello, cuando se necesita hacer una búsqueda de recursos, ésta debe realizarse por medio de iguales estáticos dentro de la red JXTA denominados iguales relays, que se encargan de realizar todas las tareas de trascendencia de un peer móvil y así permitirle ser parte de la red JXTA como cualquier otro elemento. Al hablar de tareas trascendentales se pueden mencionar las siguientes de forma general:

- Proporcionar a los iguales de tipo J2ME tareas de usuario, de grupo y de descubrimiento de iguales. Así como crear pipes y grupos dentro de la red JXTA.

- Filtrado de Tráfico.

- Ajuste para la optimización de anuncios.

Una vez que se han expuesto de una forma más profunda las características de los sistemas P2P, se puede concluir que dichos sistemas, son una alternativa que ofrece un enorme potencial para poder alcanzar la metas planteadas por el cómputo ubicuo, por lo cual la infraestructura de red necesaria para lograr implementar la arquitectura propuesta por este trabajo de tesis está basada totalmente en este tipo de sistemas.

Por otro lado, hacer uso del enfoque propuesto por los sistemas P2P, puede dar una falsa percepción de que en realidad se está usando un modelo cliente-servidor y por lo tanto se compromete la visión que tienen los sistemas P2P, pero no es así, ya que a diferencia del modelo cliente-servidor, la relación que se crea con el igual relay no requiere ser estática. Lo anterior significa que dos iguales pueden establecer una conexión con relays, totalmente distintos y aún así lograrán descubrirse y podrán comunicarse, lo que posibilita a que un igual móvil pueda moverse dinámicamente de un relay a otro, o ser miembro de múltiples relays.

3.7 Sistemas mínimos y sistemas operativos.

La creación de aplicaciones en computación ubicua encuentra una de sus dificultades en la diversidad de procesadores y de sistemas operativos que se emplean en la actualidad, de tal manera que esta elección puede repercutir directamente en el costo de desarrollo. En este trabajo de tesis, uno de los objetivos es lograr que la implementación de la arquitectura de cómputo ubicuo propuesta se base completamente en un sistema operativo de libre distribución, y requisitos computacionales mínimos, lo que se puede traducir en la materialización de una arquitectura ubicua de bajo costo, que pueda ser utilizada por diversas instituciones y empresas, ya que por lo general los altos costos de estos sistemas pueden ocasionar que no surja el interés en ellos. Es por ello que a continuación, se tratan en profundidad conceptos relacionados con los sistemas mínimos, así como sistemas operativos de libre distribución.

3.7.1 Definición de sistema mínimo.

Un sistema mínimo [20] basado en un microprocesador o microcontrolador [21] es una microcomputadora de propósito específico, equipada con el mínimo de componentes

(memoria RAM, ROM, puertos, sensores, etc.) para realizar diversas funciones. Los propósitos para los que puede diseñarse pueden pertenecer en una infinidad de campos como: instrumentación, control, monitoreo, señalización, serialización, autorización, comunicaciones, procesamiento de señales, etc. (Ver Figura 3.6).

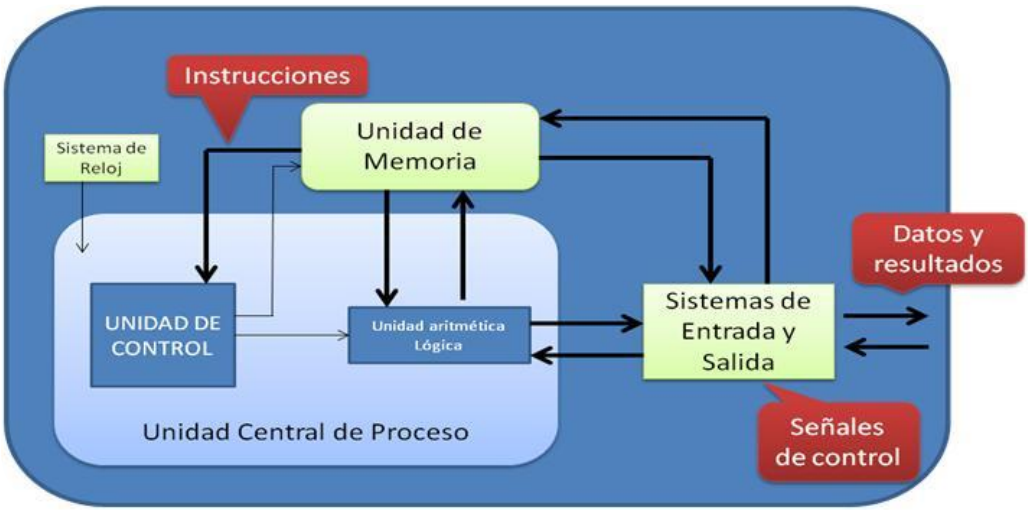


Figura 3.6 Diagrama de bloques básico de un sistema mínimo.

Por lo general, para el desarrollo de sistemas mínimos, se emplean microcontroladores que como se mencionó anteriormente, son dispositivos que contienen: una CPU, basada principalmente en un microprocesador de 4, 8 ó 16 bits, puertos paralelos de entrada y salida, puerto serie, temporizadores, contadores, memoria y en algunos casos, hasta convertidores analógicos digitales, todo dentro de un solo chip. Se emplean en multitud de sistemas presentes en la vida diaria, generalmente empotrados en sistemas donde controlan características o acciones, como lavaplatos, aparatos de TV o coches; entre otros.

Una aplicación típica podría emplear varios integrados para regular pequeñas partes del sistema. Éstos podrían comunicarse entre sí y con un procesador central, potente, compartir la información y coordinar sus acciones, como ocurre en cualquier computadora. Este tipo de sistemas se pueden adquirir de forma comercial para determinadas funciones, el inconveniente que surge es que la mayoría de ellos están basados en microprocesadores antiguos y por lo tanto en ocasiones no se adaptan a las necesidades requeridas; otra forma de subsanar ese problema es diseñarlos de acuerdo a necesidades, es decir partiendo desde cero. Un ejemplo de estos sistemas puede encontrarse en el microcontrolador de red DS80C400 [22] cuya capacidad de trabajo lo hace un buen candidato para diseños donde se habilita Ethernet. Incluye una pila de protocolos TCP/IP en la memoria ROM del procesador.

3.7.2 Sistemas Operativos.

Un sistema operativo [23-25] es un programa que actúa como interfaz entre el usuario de un ordenador y el hardware del mismo, ofreciendo el entorno necesario para que el usuario pueda ejecutar programas. Los sistemas operativos pueden ser clasificados de la siguiente forma:

- **Multiusuario**[26-28]: Permite que dos o más usuarios utilicen sus programas al mismo tiempo. Algunos sistemas operativos permiten a centenares o millares de usuarios trabajar al mismo tiempo.

- **Multitarea** [31]: Permite que varios programas se ejecuten al mismo tiempo.

- **Multitramo** [32]: Permite que diversas partes de un solo programa funcionen al mismo tiempo.

- **Tiempo Real** [33-35]: Responde a las entradas inmediatamente. Los sistemas operativos como DOS y UNIX, no funcionan en tiempo real.

Existen diversos tipos de sistemas operativos [36],[37] algunos pertenecen a la categoría de software de libre distribución y otros pertenecen a software propietario, los considerados importantes en la literatura se muestran a continuación en la Tabla 3.2:

Tabla 3.2 Resumen de características de diferentes sistemas operativos

Sistema	Programación	Usuario único	Usuario múltiple	Tarea única	Multitarea
MS-DOS	16 bits	X		X	No preventivo
Windows 3.1	16/32 bits	X			Cooperativo
Windows 95/98/Me	32 bits	X			Preventivo
Windows 2000	32 bits		X		Preventivo
Windows XP	32/64 bits		X		Preventivo
Unix/Linux	32/64 bits		X		Preventivo
MAC/OS X	32 bits		X		Preventivo
VMS	32 BITS		X		Preventivo

3.7.3 GNU/Linux.

El proyecto [38-40] GNU se inició en 1984 con el objetivo de crear un sistema operativo completo tipo Unix de software libre y que hace uso del núcleo de Linux. Esta combinación da como resultado un sistema operativo estable, con el que se puede trabajar continuamente durante meses, e incluso años, sin sufrir un solo bloqueo. Puede ser instalado en gran variedad de plataformas, incluyendo computadoras de escritorio y portátiles, computadores de bolsillo, teléfonos celulares, dispositivos empotrados, videoconsolas (Xbox, PlayStation 3, PlayStation Portable, Dreamcast, GP2X), y otros (como enrutadores o reproductores de audio digital como el iPod). Existen diversas variantes de este sistema operativo las cuales son denominadas distribuciones que incorporan determinados paquetes de software para satisfacer las necesidades de un grupo específico de usuarios, dando así origen a ediciones para el hogar, empresariales y para servidores [43]. Pueden ser exclusivamente de software libre, o también incorporar aplicaciones o controladores que requieran un pago. Son altamente personalizables y flexibles, a su vez estas distribuciones cuentan con dos sub clasificaciones que a continuación se mencionan:

- **LiveCD.** Se trata de una distribución de Linux que funciona al 100%, sin necesidad de instalarla en la computadora. Utiliza la memoria RAM para instalar y arrancar.

- **Live USB.** Variante en el que una memoria USB contiene el sistema operativo Linux completo, el cual permite iniciar una computadora desde el dispositivo USB.

3.7.4 Principales características.

- **Multitarea**: La palabra multitarea describe la habilidad de ejecutar varios programas al mismo tiempo. LINUX utiliza la llamada multitarea preventiva, la cual asegura que todos los programas que se están utilizando en un momento dado serán ejecutados, siendo el sistema operativo el encargado de ceder tiempo de microprocesador a cada programa.

- **Multiusuario**: Muchos usuarios usando la misma máquina al mismo tiempo.

- **Multiplataforma**: Las plataformas en las que en un principio se puede utilizar Linux son: 386, 486. Pentium, Pentium Pro, Pentium II, Amiga y Atari, también existen versiones para su utilización en otras plataformas, como Alpha, ARM, MIPS, PowerPC, SPARC, etc.

- **Multiprocesador**: Soporte para sistemas con más de un procesador están disponible para Intel y SPARC. Protección de la memoria entre procesos, de manera que uno de ellos no pueda colgar el sistema.

- **Carga de ejecutables por demanda**: Linux sólo lee del disco aquellas partes de un programa que están siendo usadas actualmente.

- **Política de copia en escritura para la compartición de páginas entre ejecutables**: Esto significa que varios procesos pueden usar la misma zona de memoria para ejecutarse. Cuando alguno intenta escribir en esa memoria, la página (4K de memoria) se copia a otro lugar. Esta política de copia en escritura tiene dos beneficios: aumenta la velocidad y reduce el uso de memoria.

- **Memoria virtual usando paginación por demanda (sin intercambio de procesos completos):** Puesto que hay mucha menos memoria física que memoria virtual, el sistema operativo ha de tener especial cuidado de no hacer un mal uso de la memoria física. Una forma de conservar memoria física es cargar sólo las páginas que están siendo utilizadas por un programa. Esta técnica de cargar sólo páginas virtuales en memoria conforme son accedidas es conocida como Paginación por Demanda. Un total de 16 zonas de intercambio de 128Mb de tamaño máximo pueden ser usadas en un momento dado con un límite teórico de 2Gb para intercambio. Este límite se puede aumentar fácilmente con el cambio de unas cuantas líneas en el código fuente.

3.8 Seguridad.

En el Capítulo 2 se analizaron diferentes trabajos y esfuerzos relacionados con implementaciones de entornos que tienen que ver con la integración del paradigma de cómputo ubico, gracias a lo cual se puede concluir que la mayor parte de estas investigaciones, centran su atención esencialmente en la funcionalidad del proyecto propuesto, dejando a un lado un tema de suma importancia, como lo es la seguridad. Al hablar de entornos ubicuos, es claro que el intercambio de información entre diversos dispositivos del entorno se realizará a través de medios inalámbricos, por lo que se requiere de la implementación de algún mecanismo para lograr dotar de seguridad a la información que será intercambiada, con el fin de evitar que usuarios maliciosos (véase Figura 3.7), hagan un uso inadecuado de la misma; para ello se recurrirá al uso de técnicas criptográficas. A continuación se explorarán los diversos aspectos que conforman esta disciplina.

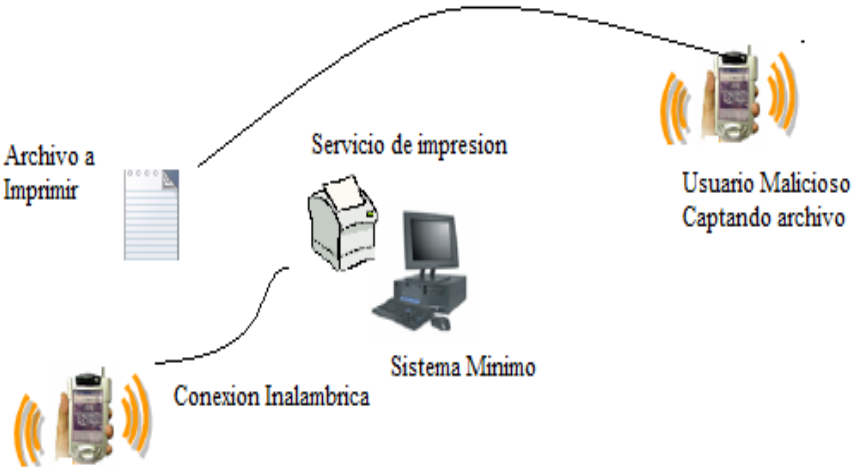


Figura 3.7 Escenario sin la implementación de un mecanismo de seguridad.

3.8.1 Criptografía.

En [42], se define como una técnica, o más bien un conglomerado de técnicas (cifrado y descifrado), que tratan sobre la protección u ocultamiento de la información frente a observadores no autorizados. Entre las disciplinas que engloba caben destacar: la teoría de la información, matemáticas discretas, que estudia las propiedades de los números enteros, y la complejidad algorítmica.

El cifrado es un proceso que mediante la aplicación de un algoritmo criptográfico, que por lo general depende de la existencia de una clave, trasforma información de carácter confidencial en ilegible, la cual es denominada como información cifrada o criptograma. Las dos técnicas más sencillas de cifrado, en la criptografía clásica, son la sustitución (que supone el cambio de significado de los elementos básicos de la información en cuestión), y la trasposición (que se trata de una reordenación de los elementos); la gran mayoría de procesos de cifrado son combinaciones de estas dos operaciones básicas. El descifrado es el proceso inverso que recupera la información a partir del criptograma y la clave.

El protocolo criptográfico especifica los detalles de cómo se utilizan los algoritmos y las claves (y otras operaciones primitivas) para conseguir el efecto deseado. El conjunto de protocolos, algoritmos de cifrado, procesos de gestión de claves y actuaciones de los usuarios, es lo que constituyen en conjunto un sistema criptográfico. Existen dos tipos fundamentales de sistemas criptográficos:

- **Simétricos o de clave privada.** Son aquellos que emplean la misma clave tanto para cifrar como para descifrar. Presentan el inconveniente de que para ser empleados en comunicaciones la clave *(k)* debe estar tanto en el emisor como en el receptor, lo cual nos lleva a cuestionarnos la manera de transmitir la clave de forma segura. Todos los sistemas criptográficos clásicos se pueden considerar simétricos, y los principales algoritmos simétricos actuales son DES, IDEA y RC5 [43] (Ver Figura 3.8).

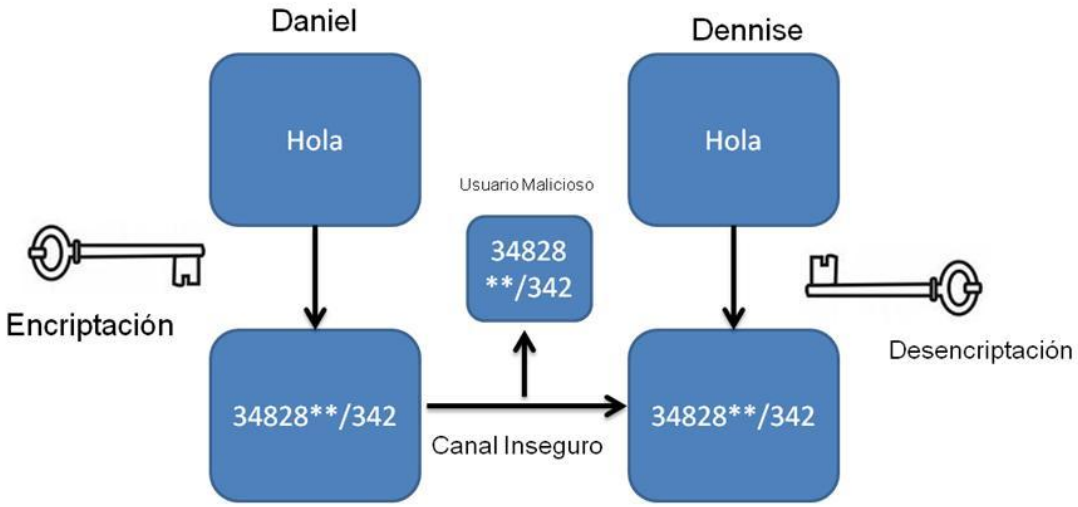


Figura 3.8 Esquema simétrico.

- **Asimétricos o de llave pública** [44], [45]. Emplean una doble clave (kp; kP). kp se conoce como clave privada y kP se conoce como clave pública. Una de ellas sirve para la transformación de cifrado (E) y la otra para la transformación de descifrado (D). En muchos casos son intercambiables, esto es, si empleamos una para cifrar la otra sirve para descifrar y viceversa. Estos algoritmos deben cumplir además que el conocimiento de la clave pública kP no permita calcular la clave privada kp. Ofrecen un abanico superior de posibilidades, pudiendo emplearse para establecer comunicaciones seguras por canales inseguros, puesto que únicamente viaja por el canal la clave pública, que sólo sirve para cifrar, o para llevar a cabo autentificaciones. El primer sistema de clave pública que apareció fue el de Diffie-Hellman [46] en 1976, y fue la base para el desarrollo de los que aparecieron después, entre los que cabe destacar el RSA [47] (el más utilizado en la actualidad), lo anterior se muestra en la Figura 3.9.

Dado que la arquitectura de cómputo ubicuo propuesta por este trabajo de tesis responde a la necesidad de transferir información confidencial, uno de los principales objetivos es la implementación de una API clara, sencilla e intuitiva, destinada al cifrado de la misma. De tal forma que sea lo suficientemente segura y robusta como requiere una aplicación de este tipo, además de que debe poder ser ejecutada en dispositivos con recursos restringidos. Es por ello que para la implementación de algoritmos de cifrado, se hace uso de la librería

criptográfica *Bouncy Castle*, debido a que cuenta con un amplio conjunto de características para satisfacer los requerimientos que se han sido planteados.

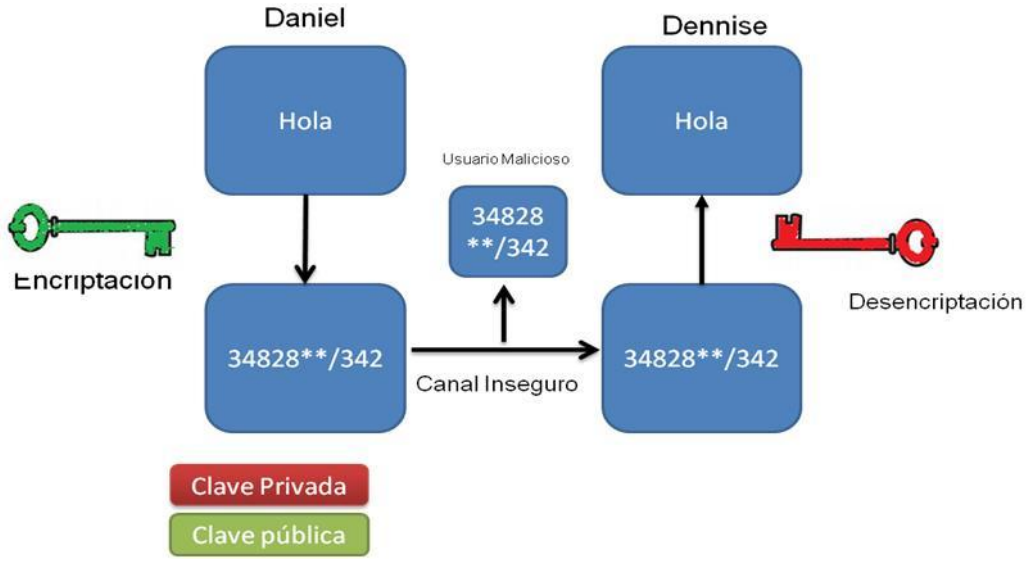


Figura 3.9 Esquema asimétrico

3.8.2 Bouncy Castle.

Bouncy Castle es un proyecto de software libre que pretende desarrollar una serie de librerías criptográficas libres y, entre otros, ofrece un proveedor *(provider)* para el JCE de Java. Además cuenta con versión de API ligera la cual trabaja con J2ME. En la página web del proyecto: http://www.bouncycastle.org/ es posible descargar la versión actual del provider Bouncy Castle para distintas versiones de la máquina virtual de Java.

Ofrece un API que permite:

- Generación de claves (claves secretas y pares de claves pública y privada).
- Cifrado simétrico (DES, 3DES, IDEA, etc).
- Cifrado con curva elíptica (ECIES).
- Cifrado simétrico (RSA, DSA, Diffie-Hellman, ElGamal, etc.).
- Funciones de resumen (MD5 y SHA1) y algoritmos MAC *(Message Authentication Code)*.
- Acuerdo de claves.

3.9 Contexto Geográfico.

En un esquema tradicional de hombre-computadora, la información contextual es una fuente importante de conocimientos que pasa desapercibida. Sin embargo, cuando hablamos de una iteración entre humanos, podemos observar que, la comprensión común de cómo funciona el mundo, y el entendimiento implícito de las situaciones de cada día, hacen que los humanos puedan expresar bien sus ideas y reaccionar apropiadamente a diversas situaciones, lo cual nos lleva a darnos cuenta de la gran riqueza que el conocimiento contextual encierra. Es por ello que en un entorno ubicuo se pretende hacer uso de este potencial, de tal forma que los dispositivos computacionales que lo integran cobren una conciencia contextual, para que de acuerdo a dichos conocimientos, se logre proporcionar una serie de servicios y/o información al usuario. Por otro lado, el mecanismo de búsqueda de servicios, que forma parte de la arquitectura propuesta en este trabajo de tesis, hace uso del conocimiento de la localización física, es decir la ubicación de las personas y objetos que realizarán las tareas, con el fin de proporcionar una lista de servicios disponibles que se adapten más al contexto geográfico del usuario, lo cual se traduce en un mayor aumento de la productividad del usuario al hacer uso de los mismos. Para recabar dicha información, se hace uso de una API de Java denominada JSR-179 Location.

3.9.1 JSR-179 Location.

La especificación JSR-179 Location API for J2ME incluye una serie de clases que permiten acceder desde Java a información relacionada con la posición y el movimiento de un dispositivo móvil. Cabe resaltar que no es necesario que el dispositivo móvil disponga de un GPS. La Location API nos aísla del origen de los datos, que puede ser un GPS o puede ser un servicio de pago proporcionado por alguna compañía proveedora de servició de telefonía celular. Debido a esto, la API nos ofrece una serie de parámetros para filtrar qué tipo de servicios nos interesan de aquellos disponibles.

3.9.2 Fuentes de localización.

En lo referente a localización, existen habitualmente tres fuentes posibles de datos:

- **GPS**: El dispositivo móvil incorpora un GPS, o está conectado a uno (habitualmente a través de Bluetooth).

- **Assisted GPS**: Además de un GPS, el dispositivo móvil tiene acceso a información de localización que evita que el GPS pase hasta varios minutos obteniendo esa información de los satélites antes de dar la primera posición. Para ello es necesario un servicio específico ofrecido por algunas compañías, habitualmente de pago.

- **Cell Id**: El dispositivo móvil no tiene acceso a un GPS, pero sí a un servicio que le proporciona información basada en el identificador de la célula telefónica en la que se encuentra. También suele ser un servicio de pago por parte de las compañías.

La Location API está preparada para que el dispositivo tenga acceso a todo tipo de Proveedores de Localización *(Location Providers)*. Para poder seleccionar entre las opciones actuales y futuras existe una clase que nos permite definir una serie de criterios que debe cumplir el proveedor que nos interese.

La clase Javax.microedition.location.Criteria proporciona funciones que permiten filtrar los proveedores de localización disponibles, entre la cuales sobresalen:

- **La precisión horizontal y vertical**. Se establece mediante los métodos setHorizontalAccuracy y setVerticalAccuracy. La precisión de los datos depende de su origen. Es decir, si indicamos una precisión de 50 metros excluiremos los datos provenientes de localización por celda telefónica, que suelen tener precisiones de cientos de metros, pero aceptaremos los datos del GPS, con precisiones típicas de pocos metros.

- **El permiso para obtener datos a cambio de un precio**. Mediante el método setCostAllowed(false) (el valor por defecto es true) se impide que se genere un costo por obtener datos de localización. Esto excluye habitualmente el A-GPS y la localización por celda telefónica, que necesitan comunicación con la red telefónica para obtener la información, además de un posible sobrecoste por el servicio.

- El nivel de consumo de batería. A través del método setPreferredPowerConsumption se puede indicar el nivel de consumo bajo, medio o alto.

- **Otros criterios**. El tiempo de respuesta, la velocidad o información relativa a la dirección.

CAPÍTULO 4
ESQUEMA PROPUESTO

Resumen: En el presente capítulo se aborda la metodología que fue utilizada para el desarrollo de la arquitectura de cómputo ubicuo. Asimismo de forma explícita se tratarán los aspectos relacionados con: compontes, diseño e implementación del entorno ubicuo y finalmente se mencionará brevemente la aportación del trabajo de tesis.

4.1 Introducción

Actualmente, los avances tecnológicos en dispositivos computarizados, así como en las redes de comunicaciones están potenciando y posibilitando la implantación de gran variedad de sistemas de cómputo ubicuo, logrando explotar parte de las capacidades que un ambiente de esta índole puede proporcionar. Sin embargo, como se ha mencionado en el capítulo 1 sección 1.1, aún hay una serie de problemas que se tienen que resolver para poder materializar un entorno de cómputo ubicuo de forma correcta. Ahora bien, esta propuesta de trabajo propone un entorno ubicuo basado en una infraestructura basada en sistemas operativos de código abierto y requisitos computacionales mínimos, la cual aborda los problemas propios del cómputo ubico, de tal manera que se permita conseguir que un usuario sea capaz, mediante aplicaciones instaladas en sus dispositivos móviles, de lograr de una forma totalmente trasparente y segura (implementado métodos de autentificación y técnicas de criptografía), una búsqueda dinámica de servicios previamente implementados en el entorno, además, tomando en cuenta el entorno geográfico, véase Figura 4.1.

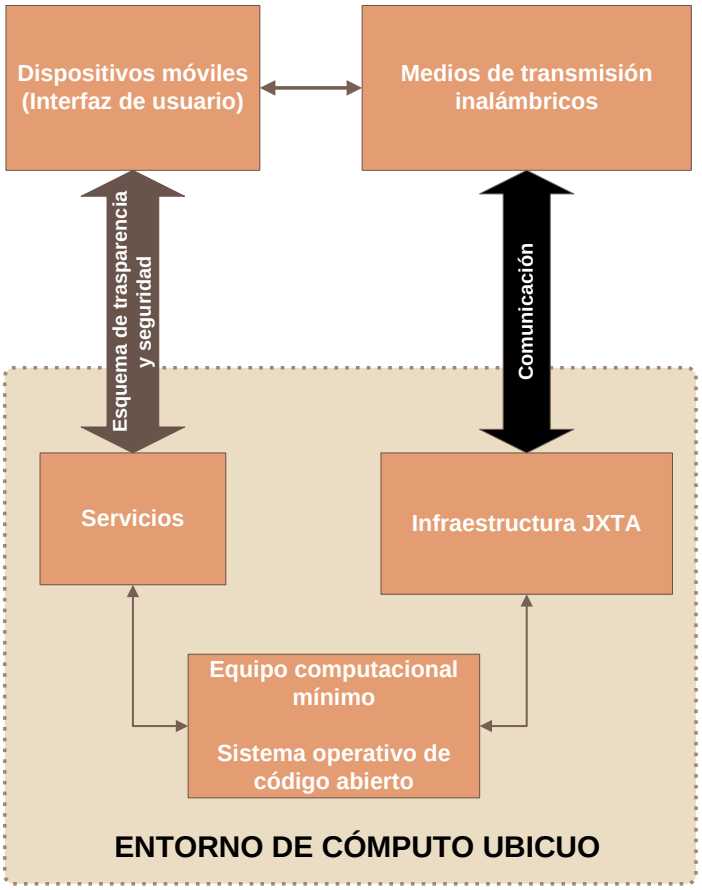


Figura 4.1 Diagrama de bloques del esquema propuesto.

Dichos servicios se refieren más específicamente, a cualquier componente, aplicación o ente que pueda ser controlado a través de un equipo computacional, por ejemplo: un servicio de impresión, servicio de proyección, servicio de autentificación, entre otros ó simplemente una aplicación tal como un procesador de textos.

Del resultado de realizar una reingeniería total a la propuesta de Pineda [4] se logró obtener una infraestructura basada en la plataforma JXTA 2.5 más robusta en cuanto a seguridad y con un desempeño superior en cuanto a la búsqueda de servicios, más rápida debido a que se usan sockets en lugar de pipes y en general más eficiente que la infraestructura original de ArCU, véase Figura 4.2.

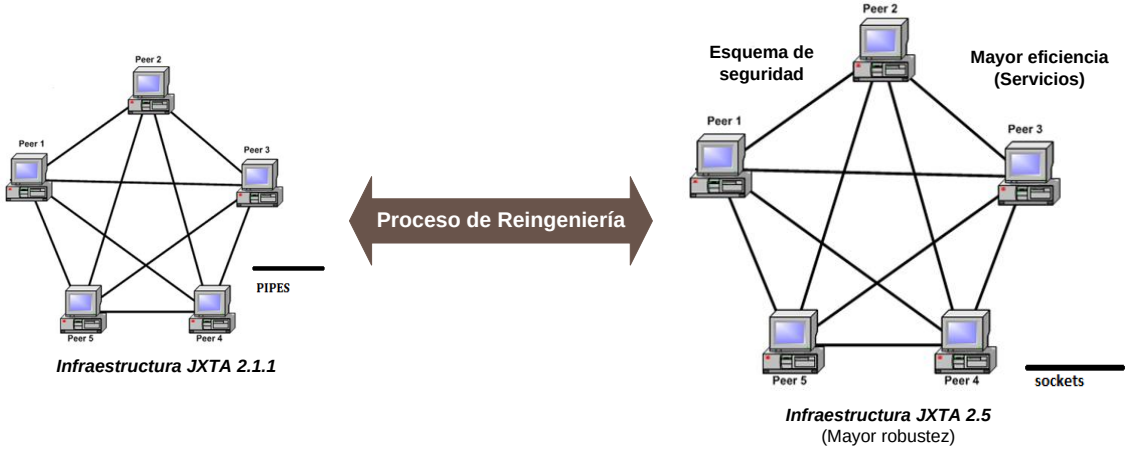


Figura 4.2 Reingeniería de la plataforma JXTA.

Es importante recalcar que el motivo de tener que hacer una reingeniería total y no solo migrar las clases utilizadas, se debió a que dichas clases han pasado a ser obsoletas debido a que la nueva versión de JXTA, eliminó diversos métodos que estaban disponibles en la versión utilizada inicialmente en ArCU. Así mismo, diversos métodos de lenguaje Java que fueron empleados en dicha infraestructura han desaparecido en la versión de Java disponible al momento de realizar la implementación de la presente arquitectura de cómputo ubicuo. En resumen, todos esos aspectos ocasionaron que se tuviera que realizar una nueva programación de diversas clases, además de, con el fin de adaptarla a versiones actuales del software y protocolos necesarios para lograr los objetivos y metas propuestas en el presente trabajo, fortificar diversos aspectos de su versión antecesora.

A continuación se procederá a describir más a detalle cada elemento que compone el diagrama de bloques descrito en la figura 4.1.

4.2 Hardware mínimo & Sistema Operativo (Código abierto).

Con el objetivo de mostrar que el esquema planteado puede ser implementado sin la necesidad de grandes recursos computacionales, así como de una manera simple, se pretende hacer uso de un hardware mínimo y un sistema operativo de código abierto, necesario para poder brindar el soporte adecuado a la infraestructura JXTA, además de permitir la integración de los componentes necesarios para poder lograr la implementación de los servicios ofertados por el entorno ubicuo.

Cabe resaltar que la elección de un hardware mínimo y del sistema operativo antes mencionado, podrían considerarse un tema ambiguo ya que para la implementación y demostración de dicha propuesta fueron considerados una serie de requerimientos (ver apéndice A), así como una serie de servicios específicos, lo cual no significa que para la implementación de un entorno ubicuo en algún área específica, esta alternativa sea la ideal ya que los requerimientos en cuanto a servicios ofertados cambiarán de acuerdo a las necesidades del usuario. De igual manera, el hardware y el sistema operativo tendrán que cambiar para poder cubrir dichos requerimientos, véase Figura 4.3.

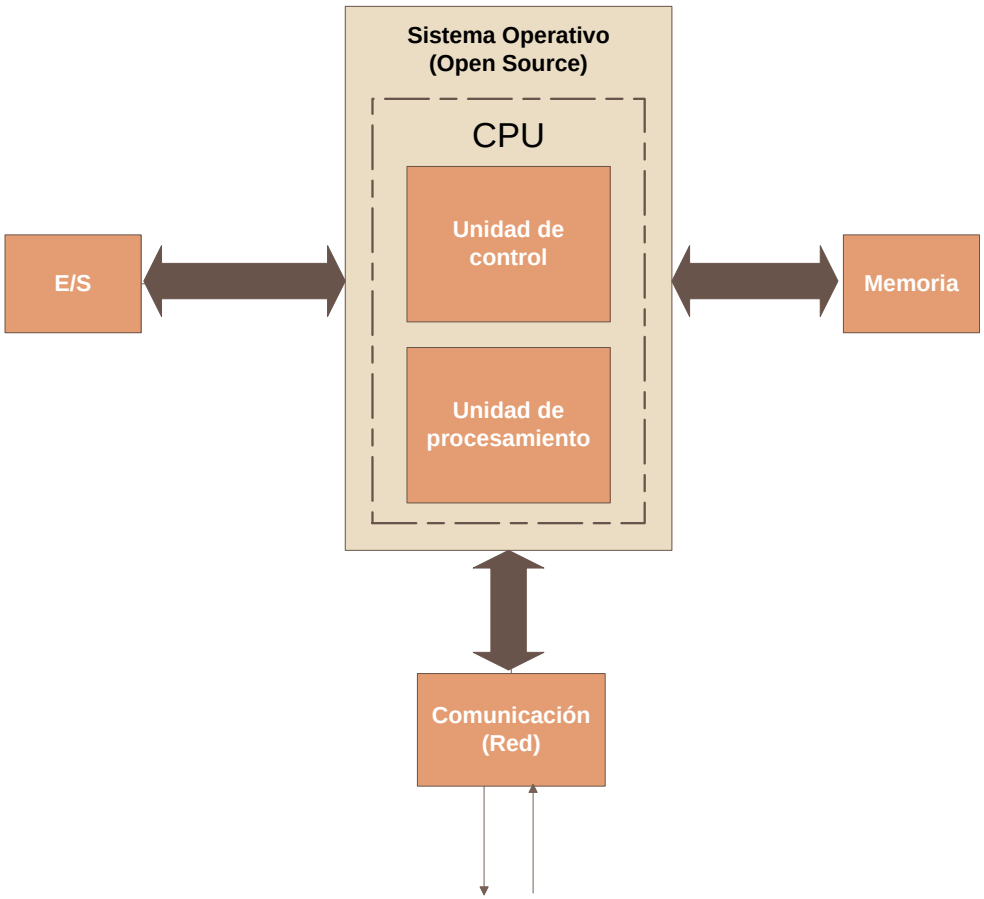


Figura 4.3 Requerimientos Mínimos de Hardware & Sistema operativo.

4.3 Infraestructura JXTA

En la sección 3.5 del capítulo 3 se estableció la gran relevancia que los sistemas de igual a igual (P2P) presentan para la creación de la infraestructura que dará soporte al entorno ubicuo [2], para lo cual se hizo uso de la plataforma JXTA. Dicha plataforma representa cada elemento del entorno como un peer, y principalmente está compuesta por los siguientes elementos (véase Figura 4.4):

- Dispositivos Móviles.
- Servicios.
- Coordinador Central, el corazón de del entorno ubicuo.
- Rendezvous.
- Relay.

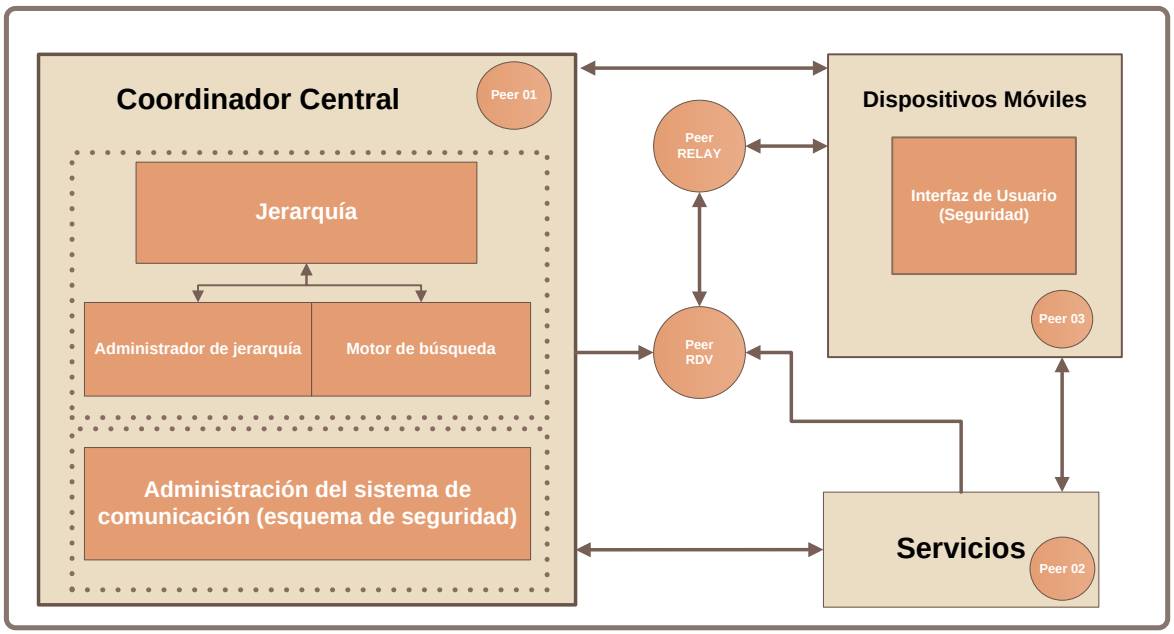


Figura 4.4 Infraestructura JXTA del entorno ubicuo.

4.3.1 Dispositivos móviles.

Los dispositivos móviles son elementos a través de los cuales un usuario podrá acceder a los diversos servicios que son ofertados, para facilitar dicha iteración, se ejecutará una interfaz de usuario en los dispositivos móviles (por ejemplo: teléfono celular, PDA, Smartphone, etc.).

Dichos artefactos incluyen diferentes API de seguridad (criptografía y autentificación), así como una API de localización, la cual permitirá acceder a información relacionada con la ubicación del dispositivo móvil. Por otro lado, a pesar de los constantes avances tecnológicos que han permitido el desarrollo de dispositivos móviles cada vez más potentes, gran parte de los usuarios hacen uso de dispositivos móviles cuyas características son las siguientes: baja capacidad de procesamiento, poco espacio de almacenamiento, acceso discontinuo a la red y movilidad. Dichas propiedades imponen restricciones en la infraestructura, mismas que se verán reflejadas en el comportamiento del sistema.

Para poder hacer frente a dichas restricciones es necesario hacer uso de la infraestructura JXTA diseñada para dispositivos de esta índole denominada JXME. A través de dicha plataforma se consigue que un dispositivo móvil pueda interactuar en la plataforma JXTA, representado como un peer de tipo minimal edge. Para lograr invocar una lista de servicios (los cuales han sido previamente registrados en el peer coordinador central) primeramente se establece una conexión con un peer de tipo relay, que está localizado en la estación de trabajo. Una vez realizado dicho proceso, el dispositivo móvil se podrá comportar como un peer edge sin ningún tipo de limitación en la infraestructura JXTA [1], véase Figura 4.5.

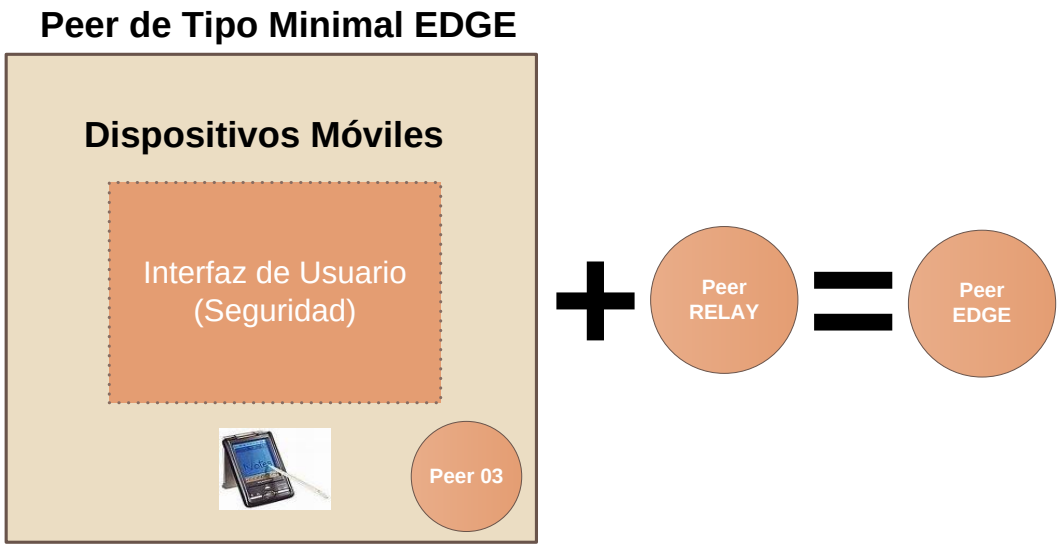


Figura 4.5 Interacción de un peer móvil en infraestructura JXTA.

A continuación, el peer de tipo relay, que ahora representa al dispositivo móvil, solicitará a un peer de tipo rendezvous, el anuncio del peer que se encarga de representar al coordinador central, para lo cual el peer de tipo rendezvous verificará en su tabla hash un índice que corresponda con el del anuncio que le ha sido solicitado. Si dicho anuncio fue localizado, éste reenviará la consulta al peer del coordinador central, cuando éste recibe la solicitud de consulta; ahora él reenvía un anuncio al peer que representa al dispositivo móvil, y cuando éste lo recibe será capaz de reclamar los servicios que han sido registrados previamente en el peer coordinador central. Véase Figura 4.6.

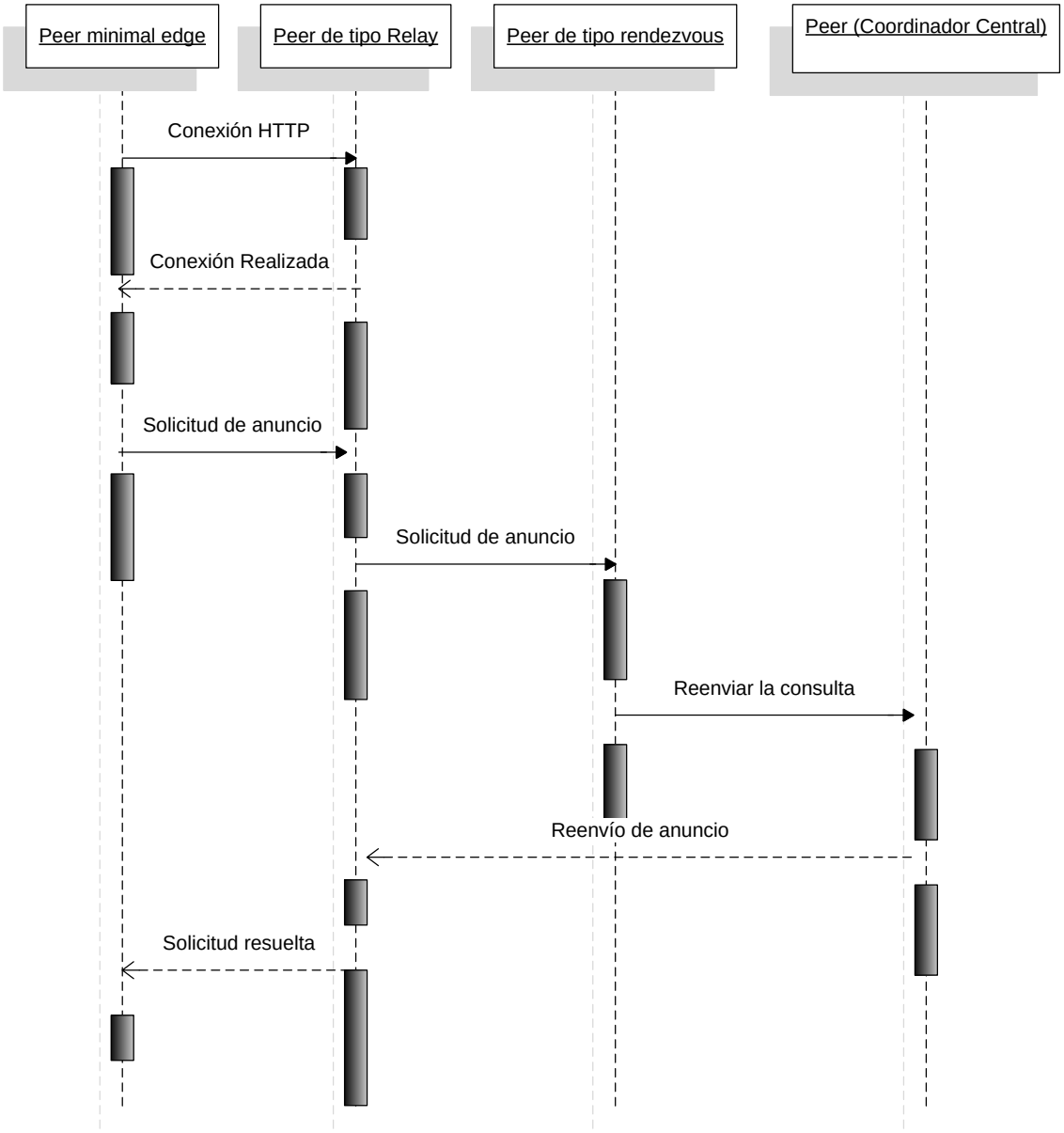


Figura 4.6 Diagrama de secuencia, el cual ilustra el proceso de búsqueda de servicios.

A continuación en la Figura 4.7 se muestra el fragmento de código necesario para la implementación de un anuncio en formato XML, el cual corresponde a un elemento primitivo de comunicación en la plataforma JXTA, denominado pipe, dicho pipe es la base para la implementación de sockets de comunicación.

```
<?xml version="1.0"?>
<!DOCTYPE jxta:PipeAdvertisement>
<jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
<Id>
    urn:jxta:uuid59616261646162614E5047205032503360 3BA68F6A604CFC9B36A276244A643B04
</Id>
<Type>
   JxtaUnicast
</Type>
<Name>
   BasicServiceInferenceAdv
</Name>
</jxta:PipeAdvertisement>
```

Figura 4.7 Ejemplo de anuncio en lenguaje XML, encargado de la creación de un pipe.

Para lograr hacer uso de los servicios, en primer instancia se requiere la localización de los mismos, a través del envío de un mensaje al peer que representa al coordinador central, el cual se encarga de interpretar el mensaje y posteriormente ejecutar una acción de búsqueda de los peers que representan a los servicios previamente registrados. Cuando el peer coordinador central logra consolidar una respuesta de la búsqueda, la traduce en un mensaje, que es enviado hacia el peer que representa el dispositivo móvil, y de esta manera se consigue visualizar en dicho dispositivo los servicios disponibles. Posteriormente, el peer móvil debe localizar el anuncio del peer que representa al servicio que se desea utilizar, por medio de la tabla hash que contiene el peer de tipo rendezvous. Dicho proceso de búsqueda es similar al realizado por el peer que representa al dispositivo móvil para localizar el anuncio del peer que representa el coordinador central, por lo que no es necesario profundizar más en dicho mecanismo de búsqueda.

Cuando el peer móvil recibe el anuncio del peer que representa al servicio que se desea utilizar, debe de enviar un mensaje al peer que representa al servicio, para obtener la interfaz de usuario, éste a su vez interpretará dicho mensaje e inmediatamente enviará un escrito, con la respectiva descripción de la interfaz de usuario necesaria. Es importante aclarar que este escrito, será elaborado completamente en lenguaje XML. Cuando el peer que representa el dispositivo móvil ha recibido dicho escrito, es necesario que utilice el generador de interfaces de usuario con que cuenta, para interpretar la información, y de esa manera se logre desplegar la interfaz necesaria para lograr controlar el servicio ofertado. Véase la Figura 4.8.

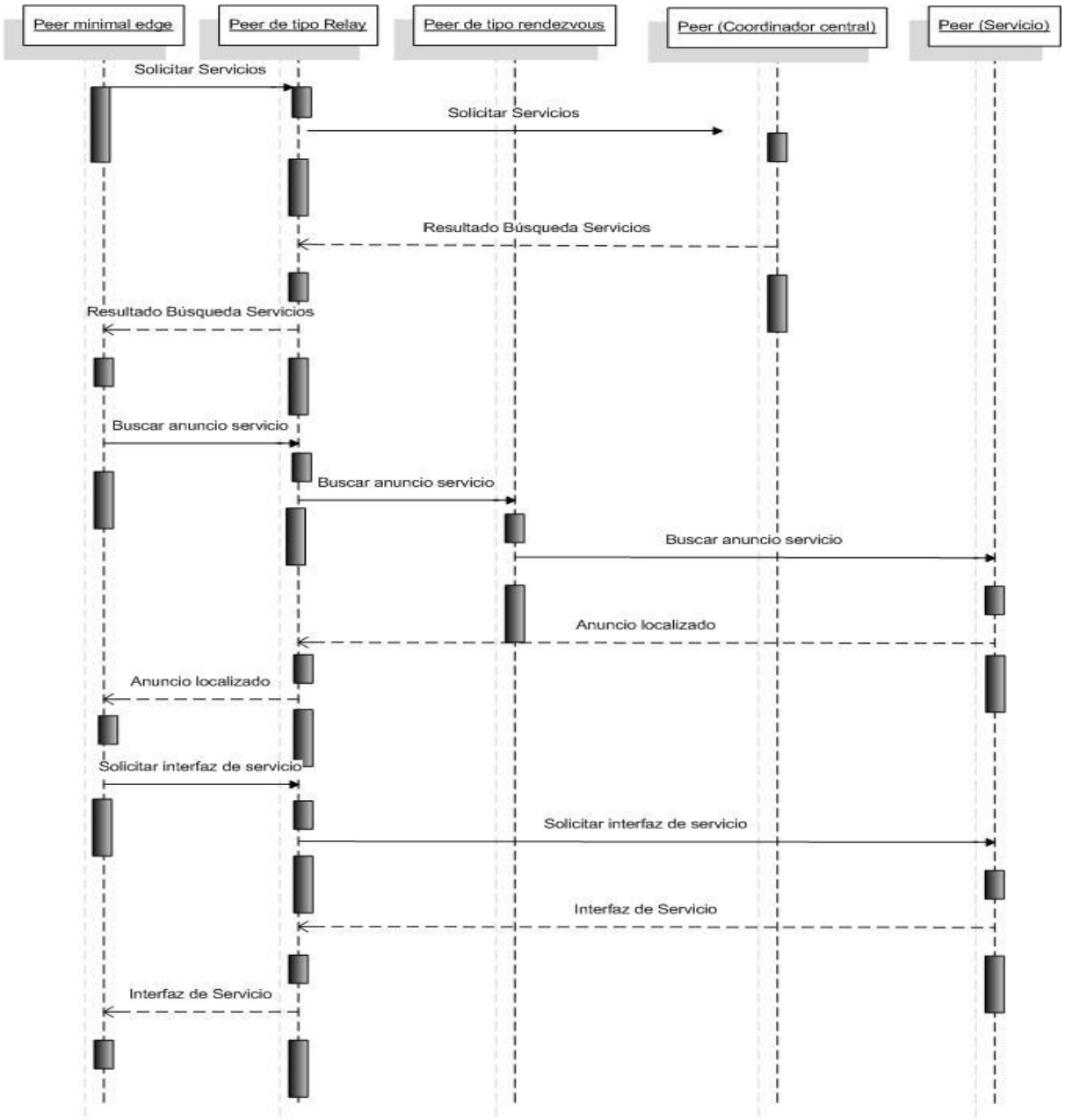


Figura 4.8 Diagrama de secuencia, el cual enuncia la secuencia de pasos necesaria para lograr hacer uso de un servicio.

En la Figura 4.9 se ilustra un archivo de tipo XML, el cual describe la interfaz de usuario necesaria para lograr interactuar con un servicio, en este caso, de impresión.

```
<?xml version="1.0"?>
<uidl>
   <interface>
      <structure>Form</structure>
      <title>Servicio de impresión </title>
      <textbox>Archivo</textbox>
      <textbox>Copias</textbox>
      <textbox>Orientación</textbox>
      <button>Enviar</button>
      </interface>
</uidl>
```

Figura 4.9 Archivo de tipo XML encargado de la generación de una interfaz de usuario.

4.3.2 Servicios.

Un servicio está representado por un peer, que permanece en la estación de trabajo, debido a que para lograr la implementación del mismo, se requiere que éste se haya conectado de alguna manera a la propia estación de trabajo (Véase Figura 4.10). El peer que se encarga de representar el servicio debe publicar un anuncio, haciendo referencia a él mismo en la tabla hash que forma parte del peer de tipo rendezvous. Anteriormente, se mostró el código XML básico de un anuncio, que primordialmente contiene un identificador, el cual es de suma importancia para permitir la localización y solicitud de dicho servicio. Por otro lado, el peer de servicio debe registrar la clase de actividad que ofrece ante el peer que representa al coordinador central, para lo cual es necesario contar con información tal como: el nombre del servicio, el tipo de servicio, el nombre del anuncio y la localización del mismo para contar con un contexto geográfico a la hora de solicitar el servicio.

El conjunto de dicha información es proporcionada por el peer de servicio mediante el uso de un escrito basado en sintaxis XML (Véase Figura 4.11). También se requerirá otro escrito, el cual describe las características del mismo servicio de una manera jerárquica, lo que quiere decir que dicho documento tendrá que estar basado en un lenguaje OWL (Véase Figura 4.12). En resumen, para poder llevar a cabo el registro del peer de servicio ante el peer que representa al coordinador central, el peer que representa al servicio envia al peer que representa al coordinador central, los dos escritos que anteriormente han sido mencionados; todo ello con el objetivo de lograr una interoperabilidad correcta y de esa manera permitir a los peers que representan los dispositivos móviles localizar y hacer uso de los servicios ofertados. Véase Figura 4.13.

Figura 4.10 Peer, el cual representa un servicio de impresión.

```
<?xml version="1.0"?><!DOCTYPE
<info:id><Name>Service</Name>
<NameSvc>Printer</NameSvc>
<FileName>Impresora.owl</FileName>
<LocationSvc>Piso numero 3</LocationSvc>
<ServiceName>ServImpre</ServiceName>
<ServiceId>urn:jxta:uuid-
59616261646162614E50472050325033C05C02F285424626B2D2B07A4549C67804</ServiceId>
<NameADV>ClientServiceSocktetAdv</NameADV>
<NamePipe>ClientServiceSocketAdv</NamePipe>
<Id>urn:jxta:uuid-
59616261646162614E50472050325033A5F5120E55A24E84A15EBE90C2259F5F04</Id>
<Type>JxtaUnicast</Type>
</info:id>
```

Figura 4.11 Ejemplo de Documento XML, el cual contiene información relevante acerca de un servicio.

```
<?xml version='1.0' encoding='ISO-8859-1'?>

<!DOCTYPE uridef[
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  ]>
<!--
This document uses entity types as a shorthand for URIs.
For a version with unexpanded entities, try loading this source
into Internet Explorer.
  -->
<rdf:RDF
  xmlns:rdf=  "&rdf;#"
  xmlns:rdfs= "&rdfs;#"
  xmlns:owl=  "&owl;#"
  xmlns:xsd=  "&xsd;#">

<owl:Ontology rdf:about="">
  <owl:VersionInfo>
    Devices Ontology version 1.0 July 2005
 </owl:VersionInfo>
</owl:Ontology>

<owl:Class rdf:ID="Devices">
  <rdfs:comment>All devices</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="Printing">
     <rdfs:comment> Printing devices</rdfs:comment>
     <rdfs:subClassOf rdf:resource="#Devices"/>
</owl:Class>

<owl:Class rdf:ID="Printer">
     <rdfs:comment>Printers</rdfs:comment>
     <rdfs:subClassOf rdf:resource="#Printing"/>
</owl:Class>

<owl:Class rdf:ID="SIPrinter">
       <rdfs:comment> Laser Jet ubicada en el laboratorio de SI </rdfs:comment>
       <rdfs:subClassOf rdf:resource="#Printer"/>
       <rdfs:subClassOf>
           <owl:Restriction>
</owl:Class>
</rdf:RDF>
```

Figura 4.12 Archivo de tipo XML, donde se ilustra un fragmento de lenguaje OWL.

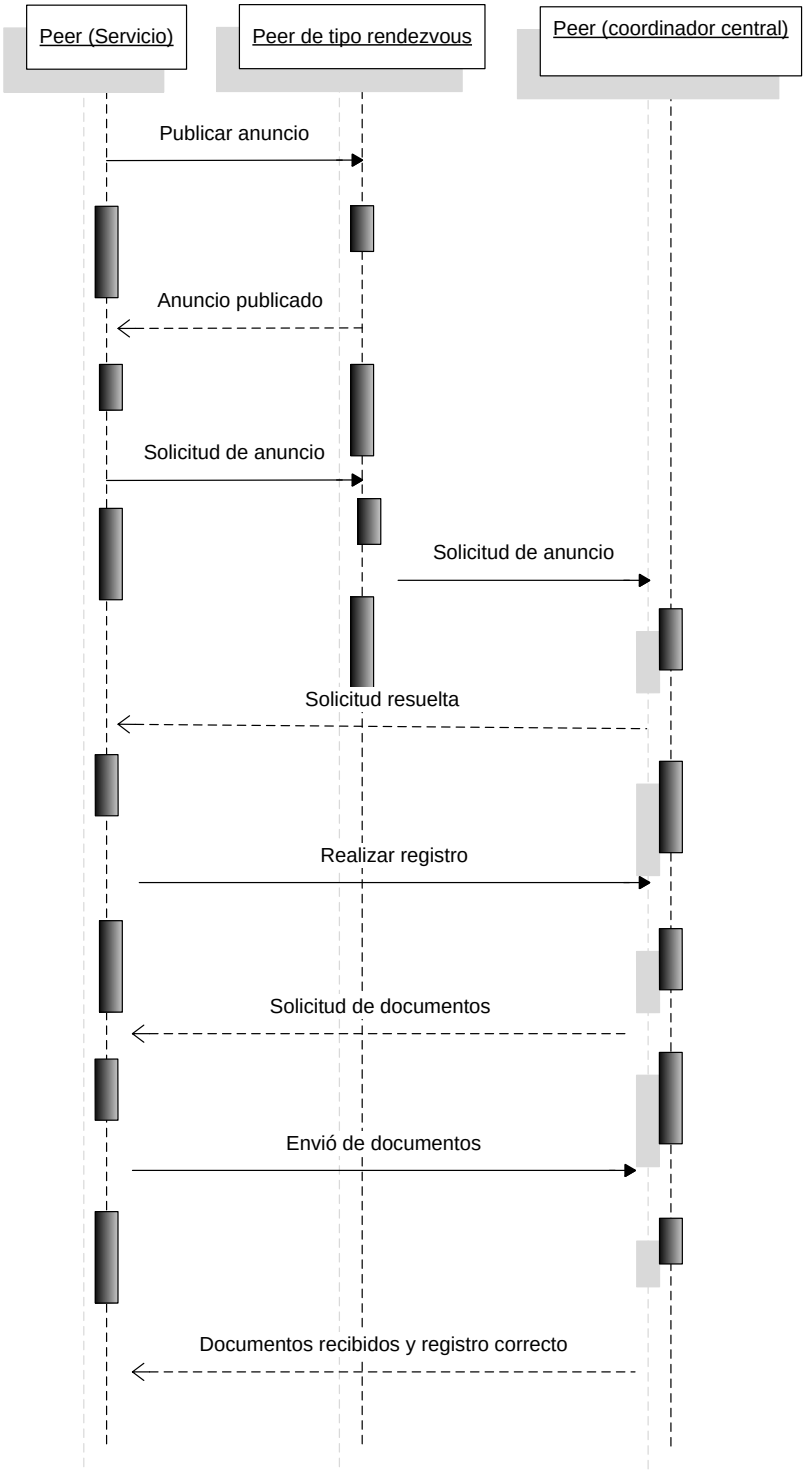


Figura 4.13 Diagrama de secuencia, mediante el cual se ilustra el proceso de registro de un servicio.

4.3.3 Coordinador Central.

El coordinador central (ver Figura 4.14), es el elemento más importante de la infraestructura de cómputo ubicuo, está representado a través de un peer que está comúnmente localizado en una estación de trabajo. Cuando se crea un peer que representa al coordinador central, el primer paso que se debe realizar, es publicar su anuncio en la tabla hash, la cual está contenida por el peer de tipo rendezvous. Por otro lado, es de suma importancia hacer énfasis una vez más el hecho de que para lograr interactuar con un peer coordinador central, con un peer de tipo móvil, o bien, con un peer de tipo de servicio, se debe hacer una consulta al peer rendezvous, para localizar el anuncio del mismo, usando un proceso similar al descrito en las secciones anteriores.

Un peer de tipo coordinador central está compuesto por los siguientes elementos:

- Un administrador del sistema de comunicación.
- Una jerarquía.
- Un administrador de ontología (jerarquías).
- Un motor de búsqueda.

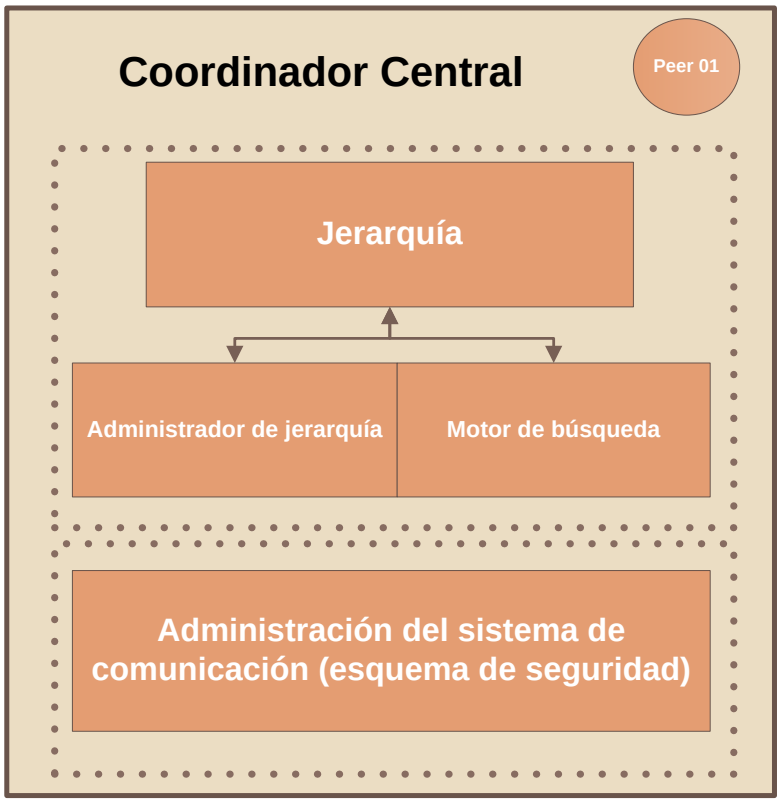


Figura 4.14 Representación de el coordinador central atreves de un peer.

4.3.4 Administrador del Sistema de Comunicación

El Administrador del Sistema de Comunicación se encarga de mantener las comunicaciones entre peers que integran la infraestructura que da soporte al entorno ubicuo propuesto. Haciendo uso de sockets JXTA (Véase Figura 4.15), se consigue un mecanismo a través del cual se puede enviar y recibir cualquier tipo de información. Dicho mecanismo, para lograr proveer un nivel de seguridad sobre el cual se pueda garantizar confidencialidad hacia la información que es manipulada, ha sido fortificado mediante varias API criptográficas (para más detalles ver apéndice B). Cabe resaltar que al hacer uso de sockets se obtiene una amplia gama de características como son:

- Se construyen sobre las tuberías *(pipes)*, mensajeros de punto final, y la biblioteca de fiabilidad.
- Provee canales de comunicación bidireccional, fiable.
- Expone una interfaz basada en un flujo *(stream)*.
- Provee un buffer interno configurable y mensajes fraccionados.
- No limita el tamaño del mensaje a 64k.

A través de los sockets se consigue una forma de comunicación virtual, con lo que es posible establecer una conexión entre peers JXTA que no tienen un enlace físico directo. Como ya se ha mencionado, los sockets se construyen sobre tuberías primitivas (pipes), de tal forma que los puntos finales del pipe se unen dinámicamente a los puntos finales del peer en tiempo de ejecución.

Una de las tareas del administrador de comunicación, consiste en la generación de sockets de entrada y de salida entre dichos peers. También tienen como misión verificar constantemente en el peer de tipo rendezvous, la publicación de los anuncios tanto de los peers que representan a los dispositivos móviles como de los peers que representan los servicios.

Cabe mencionar que si algún peer de tipo móvil o de tipo peer de servicio no está conectado, no se logrará localizar su anuncio ya que todos los anuncios que son publicados cuentan con un tiempo de vida específico. Por lo tanto, el administrador se encarga de cerrar los sockets de comunicación de dichos peers. Por otro lado, el administrador de servicios de comunicación se encarga de dar de alta y/o eliminar cada servicio.

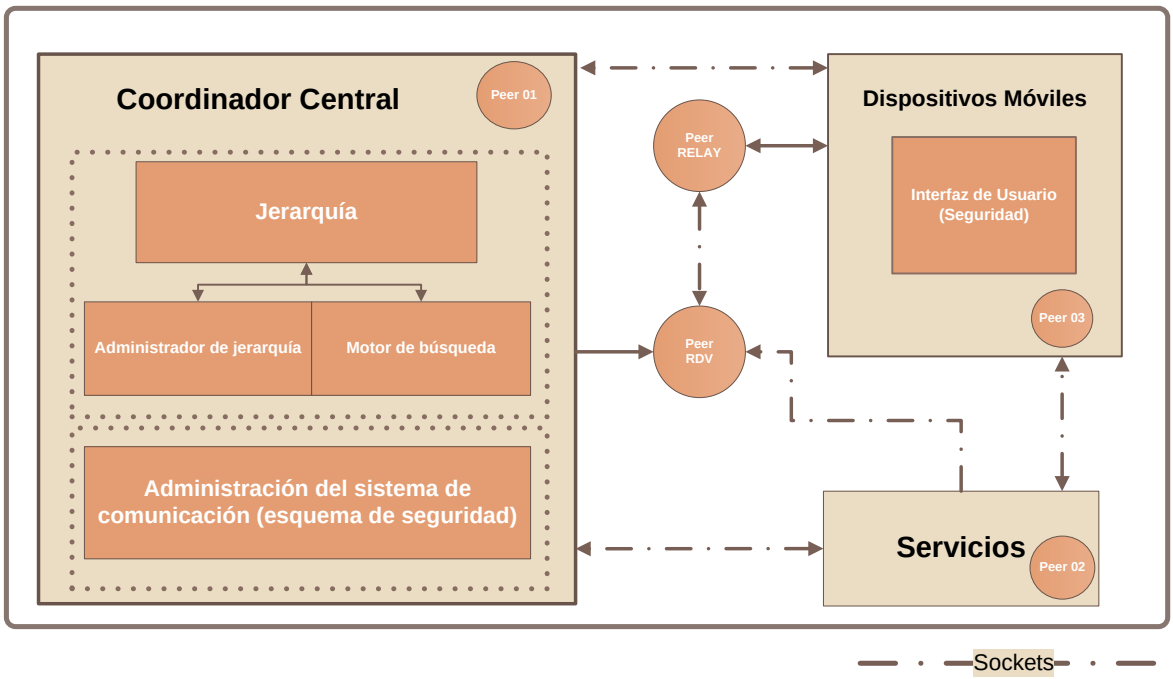


Figura 4.15 Representación de sockets en el esquema propuesto.

4.3.5 Jerarquía de Servicios.

En este trabajo de tesis se hace uso de una jerarquía de clases, la cual es un caso particular de las ontologías, en donde se tiene una sola relación. El objetivo de dicha jerarquía es el de establecer un lenguaje común para el proceso de descripción semántica de cada uno de los servicios, por lo que, cuando un peer de tipo servicio requiera registrase ante el peer que representa al coordinador central, debe proporcionar la información requerida con base en el vocabulario que ha sido propuesto, para que de esa manera se genere parte de la ontología que describirá la clase de servicio que provee. De esa manera el peer coordinador central contendrá jerarquías que incluyen la descripción semántica de los servicios disponibles.

A continuación en la Figura 4.16 se presenta a manera de diagrama de árbol un fragmento del acervo de dispositivos que integran la jerarquía de servicios que ha sido usada para la implementación del entorno ubicuo propuesto en este trabajo. Esencialmente se hace uso de una metodología top-down, lo que se traduce en una jerarquía compuesta inicialmente por el concepto más común y continuando con una subsiguiente especialidad.

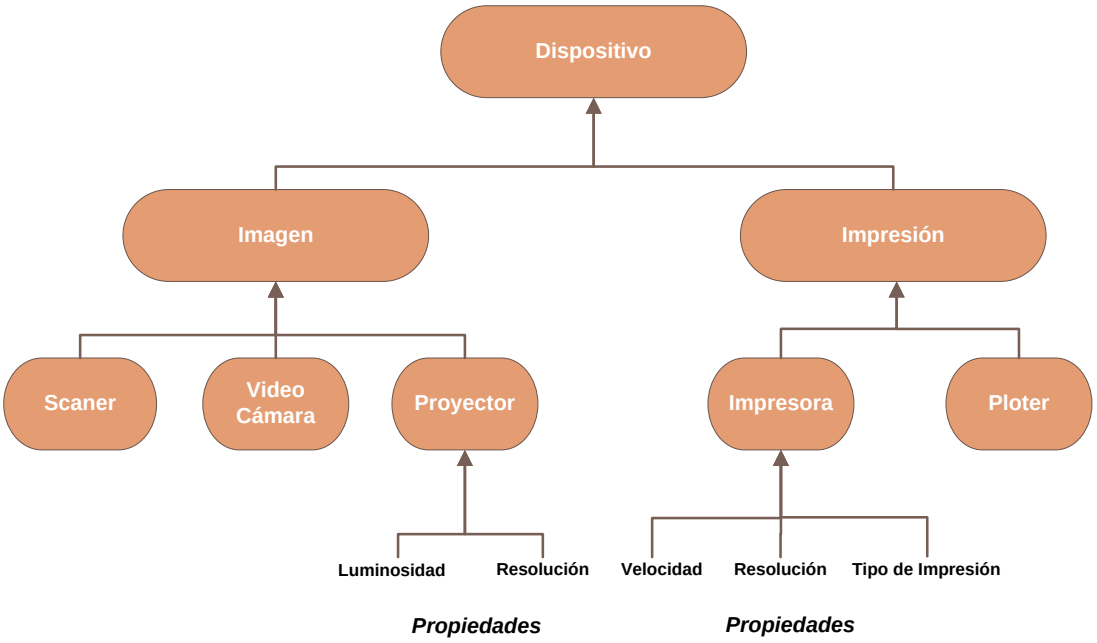


Figura 4.16 Jerarquía de clases usando una metodología top-down.

En este ejemplo, se toma el nodo dispositivo como raíz de la jerarquía, ya que este término puede significar cualquiera de los dispositivos en forma general. En el siguiente nivel, están los nodos imagen e impresión, también se categorizan otros nodos como impresora y plotter como nodos hijos del nodo impresión. Con base en esta jerarquía se construirá el vocabulario que utilizarán los peers de servicio para describir sus servicios, como se mencionó anteriormente.

Como la identificación de los nodos no proporciona suficiente información acerca del dominio y alcance de la ontología (jerarquía), se debe describir también la estructura interna de los nodos. En la Tabla 4.1 y Tabla 4.2 se muestran la descripción de cada una de las propiedades tanto del nodo impresora y como del nodo proyector.

Tabla 4.1 Descripción de propiedades del nodo proyector.

Propiedad	Descripción
Luminosidad	Describe la luminosidad en lúmenes que permiten la reproducción de imágenes de gran definición.
Resolución	Describe la calidad de reproducción

Tabla 4.2 Descripción de propiedades del nodo impresora.

Propiedad	Descripción
Resolución	Describe la calidad de impresión
Velocidad	Describe la cantidad máxima de páginas por minuto
Tipo de impresión	Describe si el formato es a color, blanco y negro o ambos

4.3.6 Administrador de jerarquía.

Como su nombre lo indica, el administrador de jerarquía es el encargado de llevar a cabo la administración (creación, organización y mantenimiento) de la jerarquía de servicios ubicuos. Cuando un peer de servicio invoca al peer coordinador central para realizar el registro de su servicio, el administrador de jerarquía, deberá extraer del documento de tipo OWL, los conceptos relacionados al servicio, y posteriormente hacer uso de los mismos para la generación de la jerarquía. La jerarquía que se genera sólo estará compuesta por términos relacionados a los servicios que han sido registrados. Por otro lado, en el caso de que el peer coordinador central reciba una nueva solicitud de registro de un servicio, el administrador integrará en la jerarquía, los nuevos conceptos relacionados con el servicio. Es importante resaltar el hecho de que el administrador de jerarquía debe mantener una continua comunicación con el administrador de sistema de comunicación, ya que gracias a ello se podrá tener el conocimiento acerca de los servicios que estén inactivos y de esa manera eliminar de la jerarquía, aquellos términos relacionados con dichos servicios. Lo anterior con el objetivo de mantener la jerarquía tan pequeña como sea posible, lo que se traducirá un una reducción de tiempo de las búsquedas semánticas.

4.3.7 Motor de Búsqueda.

El motor de búsqueda es otro elemento del peer coordinador central, el cual se encarga de buscar conceptos de uno o más servicios dependiendo de las coincidencias que encuentre. Este elemento se encarga de procesar las peticiones hechas por los peers móviles y extraer los parámetros de búsqueda que llegan junto con la petición. Con estos parámetros se hacen búsquedas a través de un proceso de razonamiento que desempeña búsquedas por conceptos para luego enviar el resultado al peer móvil que originó la petición.

Es importante señalar que la funcionalidad del peer coordinador central a través de estos elementos asegura la consistencia e integridad de la información en el entorno ubicuo propuesto en este trabajo; esto con el fin de de que los usuarios estén conscientes del estado actual de los servicios.

4.4 Confidencialidad en el intercambio de información.

Uno de los principales objetivos de este trabajo es dotar de seguridad, en lo que se refiere al manejo de información considerada de carácter confidencial. Por lo anterior, se implementó una API, basada en la librería criptográfica Bouncy Castle. Dicha API fue embebida en el coordinador central (véase Figura 4.17) quien, a través de la misma, contará con la posibilidad de poder codificar la información que se envía hacia el dispositivo móvil, así como decodificar la información que recibe de manera codificada. Por otro lado, es necesario incluir una API, con las características necesarias, para poder ser incluida en el dispositivo móvil, de tal manera que se proporcione un mecanismo de codificación/decodificación de información al dispositivo. Dicha API, se empotró dentro de la interfaz de usuario del dispositivo móvil. Véase Figura 4.18

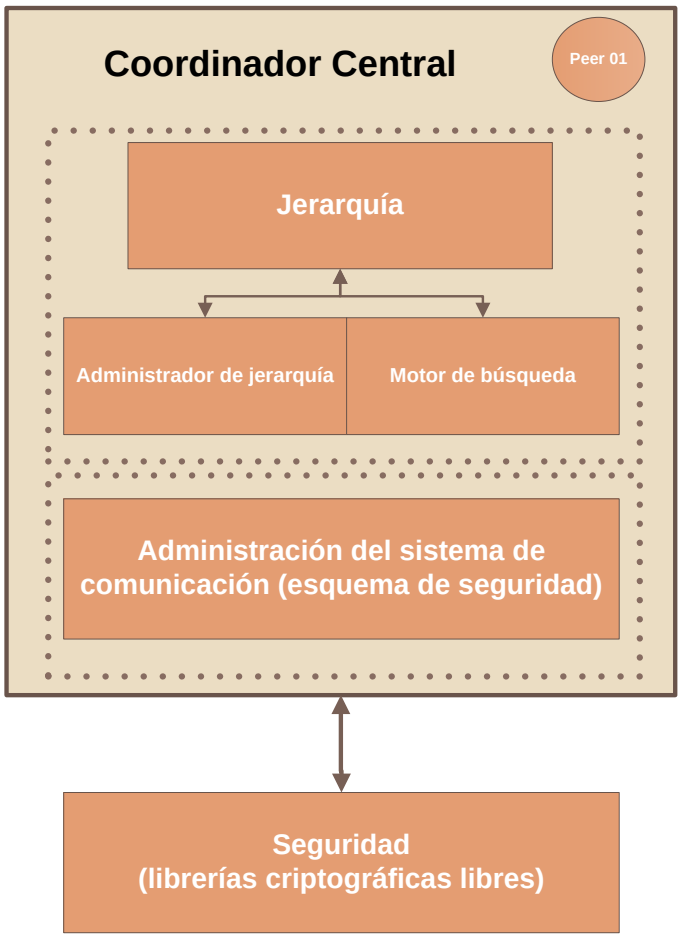


Figura 4.17 Esquema de seguridad implantado.

Figura 4.18 Esquema de seguridad implantado en dispositivo Móvil

4.5 Implementación.

A continuación y con el objetivo de demostrar la potencialidad que se obtiene al hacer uso de la infraestructura planteada por este trabajo de tesis, para su implementación en el entorno ubicuo, se analizarán detalladamente aspectos que envuelven dicha inserción, tales como: la implementación de un peer coordinador central, la integración de servicios y la inserción de una interfaz de usuario en el dispositivo móvil. Dicha interfaz es de gran relevancia ya que debido a la misma, los usuarios podrán interactuar con el entorno ubicuo establecido.

4.5.1 Características del entorno.

Dicha implementación se consigue mediante una serie de clases desarrolladas en el lenguaje de programación Java, se requiere que la estación central cuente con J2SE instalado [3]. La propuesta del presente trabajo, ha sido implementada sobre una versión de Java 1.6, además, se debe configurar la plataforma JXTA 2.5 (para más detalles sobre dicha plataforma ver apéndice D), ya que en gran parte la infraestructura depende directamente de dicha plataforma.

Anteriormente se mencionó que las jerarquías de los servicios son descritas empleando la sintaxis OWL [4], y además, diversos documentos de descripción, los cuales están basados en lenguaje XML. Por lo anterior, es necesario instalar y configurar una analizador XML;

en esta propuesta se utilizó una librería de Apache para analizar y generar documentos XML, la cual se denomina Xerces versión 2.9.1 [5].

Para habilitar los dispositivos móviles, en dicha infraestructura, se hace uso de Java 2 Micro Edition así como CLDC 1.2/MIDP 2.0 *(Connected Limited Device Configuration/ Information Device Profile)*. En consecuencia, fue necesario instalar una plataforma de desarrollo, la cual fue JAVA ME versión 3.0, y además se requiere del uso de una versión JXTA para dispositivos móviles denominada JXME [6]. En cuanto a la parte inalámbrica, esta propuesta está cimentada principalmente sobre una red inalámbrica de tipo Wi-Fi, y una red básica de de tipo Ethernet.

4.5.2 Coordinador Central.

En la sección 4.3.3 se abordó con más profundidad el Coordinador Central, del cual se podría resumir lo siguiente: es el encargado de administrar y proporcionar los servicios disponibles por el entorno ubicuo. A continuación se detallarán segmentos de código, que son de gran relevancia en los principales procesos del coordinador central:

- Proceso de inicialización de la infraestructura.
- Proceso de adicción de servicios.
- Búsqueda semántica y contextual de los servicios.
- Mecanismo de detección de caída y eliminación de la jerarquía de los mismos.

4.5.3 Proceso de inicialización del coordinador central.

En primera instancia, ya que la infraestructura está basada completamente en la plataforma JXTA, el coordinador central debe ser inicializado como un elemento peer. A continuación, en la Figura 4.19 se muestra la parte del código necesaria para llevar acabo dicho proceso:

Cabe resaltar que para llevar cabo la inicialización se utilizó un archivo de tipo XML (véase Figura 4.25), el cual contiene una serie de información previamente seleccionada para lograr el correcto proceso de inicialización como un peer. Específicamente el método encargado de dicha tarea es denominado configure(), y esencialmente se encarga de configurar una serie de parámetros, como son: nombre del peer, TCP/IP, HTTP, peers relay, peers rendezvous, y además como medida de seguridad y autentificación, es necesario un nombre de usuario y password. El método bsi.save() se encarga de almacenar por default la configuración del peer, en el archivo "./.jxta/PlatformConfig", ello con el objetivo de que tome la configuración almacenada, cada que se reinicie el peer que representa al coordinador central. Véase Figura 4.20.

```
public static void main(String args[ ]) throws Exception
{
   try
   {
      CenterCordinator  cc = new CenterCordinator();
      System.out.println("Iniciar Configuración");
      cc.configure();
      System.out.println("Finalizar configuración");
      cc.save(".jxta/PlatformConfig");
      System.out.println("Iniciando conexión");
      cc.JxtaConnection();
      System.out.println("Finalizando proceso de conexión");
      System.out.println("Iniciando hilo de servicios");
      cc.startJxta();
      cc.startServer();
      cc.Status();
   }
   catch (Exception e) { }
}
```

Figura 4.19 Código encargado de la inicialización del coordinador central.

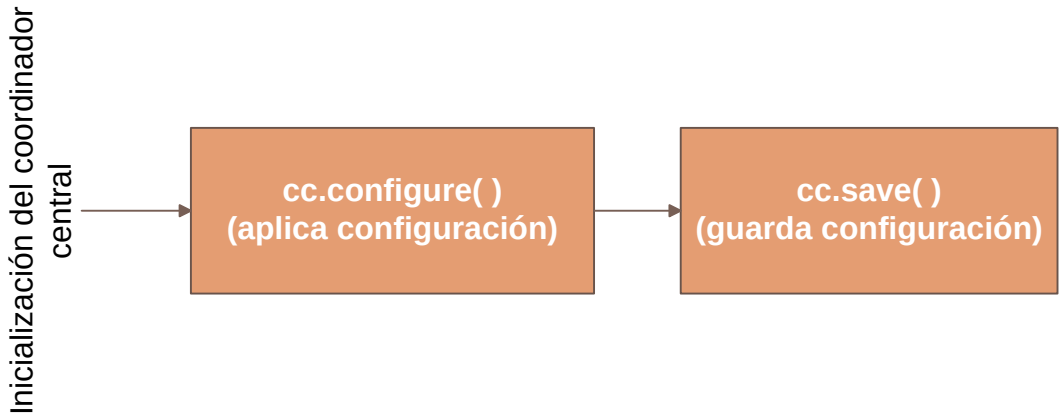


Figura 4.20 Proceso de Configuración de Coordinador Central.

Ahora en la Figura 4.21 se ilustra el fragmento de código correspondiente al método JxtaConnection().

```
public void JxtaConnection( ) throws CommunicationException
{
   try
   {
      //Unión al grupo principal
      setNetPeerGroup(PeerGroupFactory.newNetPeerGroup());
      System.out.println("Se ha unido al grupo Principal");
      GrupoPrincipal = getNetPeerGroup();

      // Unión al grupo actual
      joinGroup(Grupoprincipal);
      System.out.println("Union al grupo actual Exitosa");
   }
…
```

Figura 4.21 Código correspondiente al método JxtaConnection().

Una vez que se logró realizar con éxito la configuración, la tarea del método JxtaConnection() consiste en realizar la unión del peer que representa al coordinador central al NetPeerGroup. Cabe resaltar que el NetPeerGroup es el grupo al que por defecto se une cualquier peer al ingresar a una red de tipo JXTA. Véase Figura 4.22.

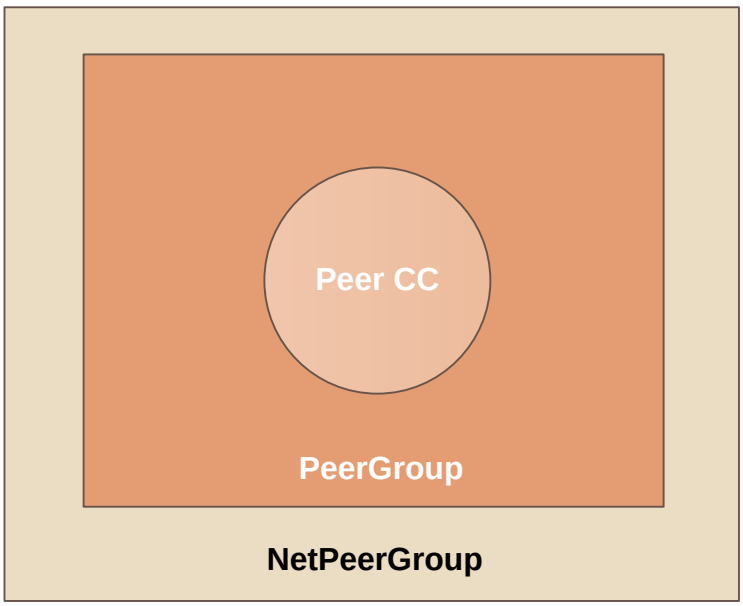


Figura 4. 22 Representación de la unión al PeerGroup actual.

El código que forma parte del método startServer( ) se ilustra en la Figura 4.23.

```
public synchronized void startServer()
{
   System.out.println("\n\nObteniendo anuncio Correspondiente al Coordinador C.");
   try
   {
      paConstructor = new SocketAdvConstructor(Grupoprincipal);
      anuncio = paConstructor.getPipeAdvfromFile(PIPEADVNAME);
      if (adv == null) {
         file = false;
         System.out.println("Creando nuevo anuncio, para el Servicio Básico de
         Inferencia");
         anuncio = paConstructor.generatePipeAdv(ccADV, PipeService.UnicastType);
      }
      System.out.println("\Anuncio socket:\nName:  " + anuncio.getName() + "\nID:
      " + anuncio.getID() + "\nType: "+ anuncio.getType());
      System.out.println("Esperando conexiones........");
   }
   catch (Exception eee)
   {
      System.out.println("Error al publicar socket " + eee.toString());
      return;
   }

   try
   {
      discovery.publish(anuncio);
      discovery.remotePublish(anuncio, DiscoveryService.ADV);
      inputsocket = pipe.createInputPipe(anuncio, this);
      PipeMsgListener myService1Listener = new PipeMsgListener() {};
      inputsocket = new JxtaServerSocket(Grupoprincipal,anuncio);
      PipeMsgEvent event=null;
      inputsocket.pipeMsgEvent(event);
      if(!file)
         paConstructor. writePipeAdv(anuncio, PIPEADVNAME);
   }
   catch (Exception e)
   {
      System.out.println("Error al publicar el anuncio");
      e.printStackTrace();
   }
}
```

Figura 4.23 Código correspondiente al método startServer()

Este método se encarga de la creación del anuncio que representará al coordinador central, además, también hace la publicación del mismo en la infraestructura JXTA, y por último, el método se encargará de construir el socket de entrada respectivo sobre un pipe primitivo, con el objeto de enviar y recibir información.

Finalmente, el método se encarga de indicar que el coordinador central está en condiciones de recibir mensajes para procesar cada uno de ellos, ya sea para realizar un registro de

servicio o para realizar la búsqueda semántica y contextual de algún servicio, ver Figura 4.24.

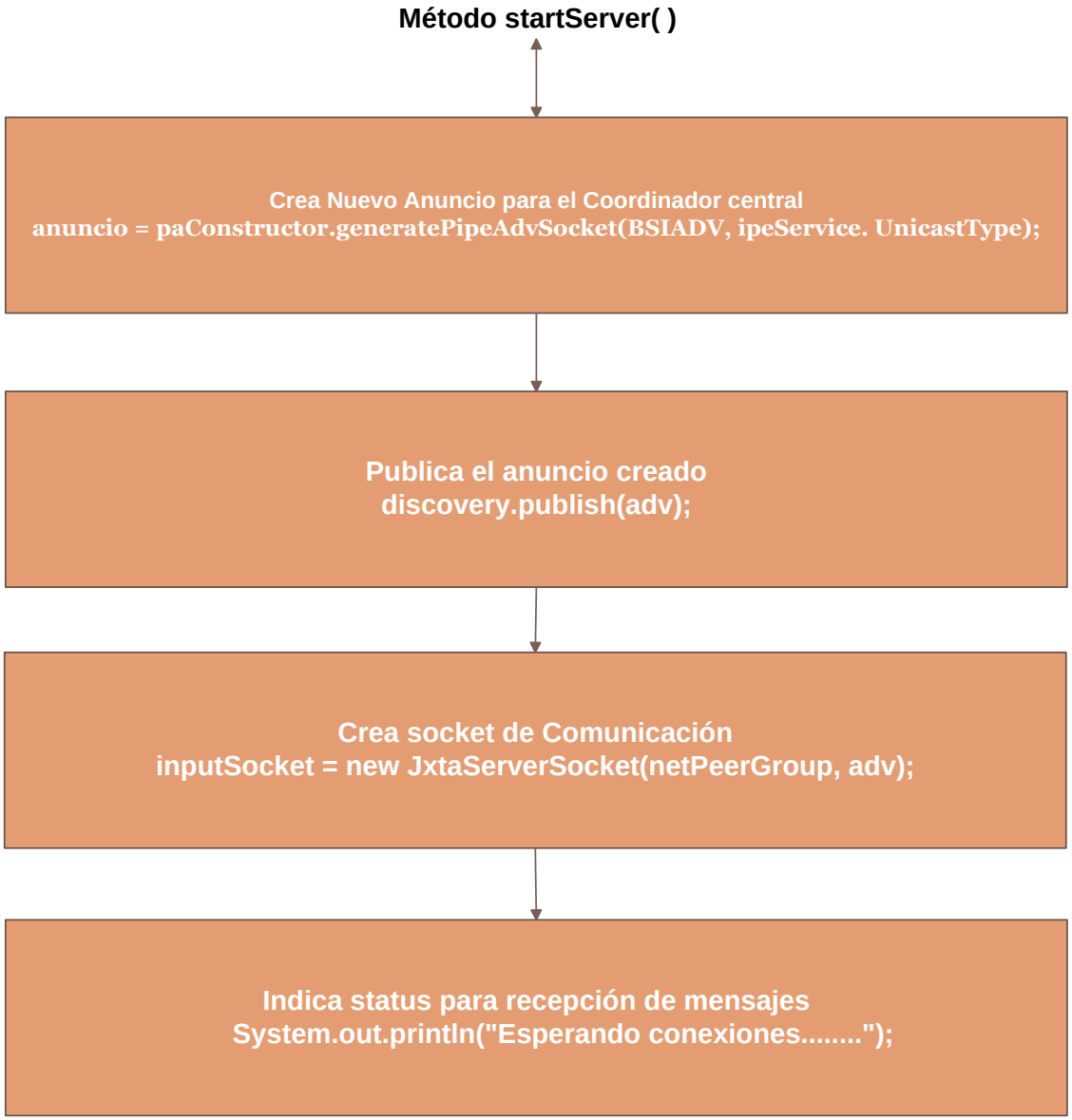


Figura 4.24 Diagrama de bloques correspondiente al método startServer().

```
<?xml version="1.0" encoding="UTF-8"?>
   <configuration>
      <!-- Configuration of JXTA Properties -->
         <JxtaConfiguration>
            <network>
               <tcp isEnabled="true" mode="auto">
                  <ipAddress source="anyAll"/>
                  <port source="inline">9701</port>
                  <multicast address="224.0.1.85" port="1234" size="16384"/>
                  <relay isRelay="false"> </relay>
               </tcp>
               <useRelay>true</useRelay>
               <http isEnabled="true">
                  <ipAddress source="anyAll"/>
                  <port source="inline">9700</port>
                  <server isServer="false"/>
                  <relay isRelay="false"> </relay>
                  <proxy useProxy="false"> <url>myProxy.myDomain:8080</url> </proxy>
               </http>
               <rendezvous isRendezvous="false"> </rendezvous>
            </network>
            <peer credential="JxtaSbiPassword" name="JxtaSbiPeerName"
             principal="JxtaSbiUserName"/>
            <application> <dynamicPort high="9799" isEnabled="true"
             low="9701"/></application>
          </JxtaConfiguration>
   </configuration>
```

Figura 4.25 Archivo XML, el cual contiene información necesaria para iniciar la configuración del coordinador central.

4.5.4 Mecanismo para agregar servicios a la jerarquía.

Através del método SocketsMsgEvent( ), el cual consiste en un proceso que se ejecutará constantemente en segundo plano, el coordinador central es capaz de detectar y atender los mensajes que lleguen. Por otro lado, el coordinador central facilita a los peers, que desean registrar alguna clase de servicio, una solicitud por medio de la cual pueden registrar el servicio que ofertan, el mismo método SocketsMsgEvent( ) (Véase Figura 4.26), se encarga de seleccionar, si el mensaje que llega al coordinador central corresponde a una solicitud de registro de servicio, y de ser así el coordinador central debe atender la solicitud de registro del mismo (Ver Figura 4.27).

```
public void SocketsMsgEvent(PipeMsgEvent event)
{

   try
   {
      Message msg = event.getMessage();
      MessageElement me = msg.getMessageElement(SENDERMESSAGE);
      String clientRequestStream = me.toString();
      System.out.println(">>> El contenido del mensaje es " +
      clientRequestStream );
      String Name = fromXML(clientRequestStream);
      System.out.println(">>> Recibiendo informacion de  "+ Name +"'s" +
      " informacion");
      if ("Cliente Movil".equalsIgnoreCase(Name))
      {
         startSearchServices(clientRequestStream, SvcsContainer,
         IdContainer, LocSvcContainer, NetPeerGroup);
      }
      if ("Service".equalsIgnoreCase(Name))
      {
         String NameSvc = fromXMLSvc(clientRequestStream);
         System.out.println(NameSvc);
         String FileName = fromXMLFileName(clientRequestStream);
         System.out.println(FileName);
         String ServiceName = fromXmlServiceName(clientRequestStream);
         System.out.println( ServiceName);
         String ServiceId = fromXmlServiceId(clientRequestStream);
         System.out.println("Identificador de servicio" + ServiceId);
         String Id = fromXmlId(clientRequestStream);
         System.out.println("identificador de CC: " + Id);
         String LocSvc = fromXmlLocationSvc(clientRequestStream);
         System.out.println("La ubicación del servicio es : " + LocSvc);
         if (SvcsContainer.containsKey(ServiceName) == false)
         {
            SvcsContainer.put(ServiceName, ServiceId);
            IdContainer.put(ServiceName,Id);
            LocSvcContainer.put(ServiceName,LocSvc);
         }
         else
         {
            SvcsContainer.remove(ServiceName);
            IdContainer.remove(ServiceName);
            LocSvcContainer.remove(ServiceName);
            SvcsContainer.put(ServiceName, ServiceId);
            IdContainer.put(ServiceName,Id);
            LocSvcContainer.put(ServiceName,LocSvc);
         }
         PipeAdvertisement clientAdv = GenerateADV(clientRequestStream);
         System.out.println("\nName:  " + clientAdv.getName() + "\nID:"
         + clientAdv.getID() +  "\nType: "+ clientAdv.getType());
         System.out.println("\n\n\n<-----Esperando conexiones--------->\n");
         System.out.println("Recibiendo archivo de : " + clientAdv.getName());
         startFTSThread(clientAdv, NameSvc, FileName, discovery, ServiceName);
         NFList.add(FileName);
         LocSvcs.put(FileName,ServiceName);
         LocAdvs.put(FileName,clientAdv);
      }
   }catch (Exception e) {}
```

Figura 4.26 Código correspondiente al método SocketsMsgEvent().

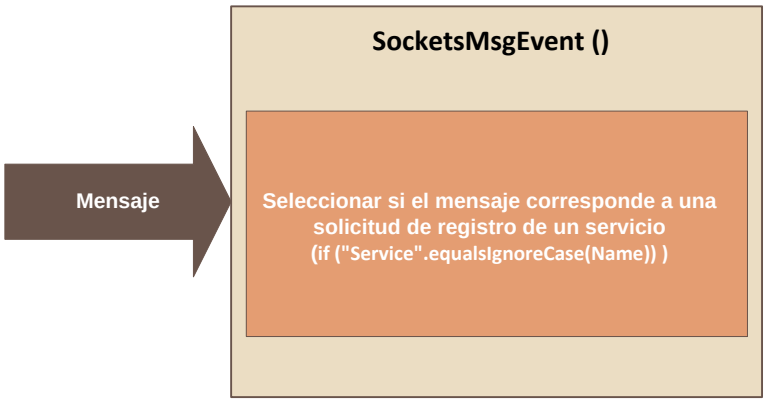


Figura 4.27 Diagrama de bloque correspondiente al método SocketsMsgEvent ().

El mensaje que arriba es representado mediante un escrito XML, del cual se extraen los siguientes parámetros: nombre del servicio, tipo de servicio, información acerca del socket del mismo, entre otros. Este cúmulo de información será de utilidad para lograr iniciar una sesión entre el peer que representa al coordinador central y el peer que representa al servicio, con la finalidad de intercambiar información (Ver Figura 4.29), el fragmento de código que se encarga del inicio de dicha sesión se muestra en la Figura 4.28.

```
System.out.println("\n\n\n<----------------Esperando conexiones--------->\n");
System.out.println("Recibiendo archivo de configuracion de : " +
clientAdv.getName());
starthilodeServicios(clientAdv, NameSvc, FileName, discovery, ServiceName);
```

Figura 4.28 Código encargado de la extracción de la información del archivo XML.

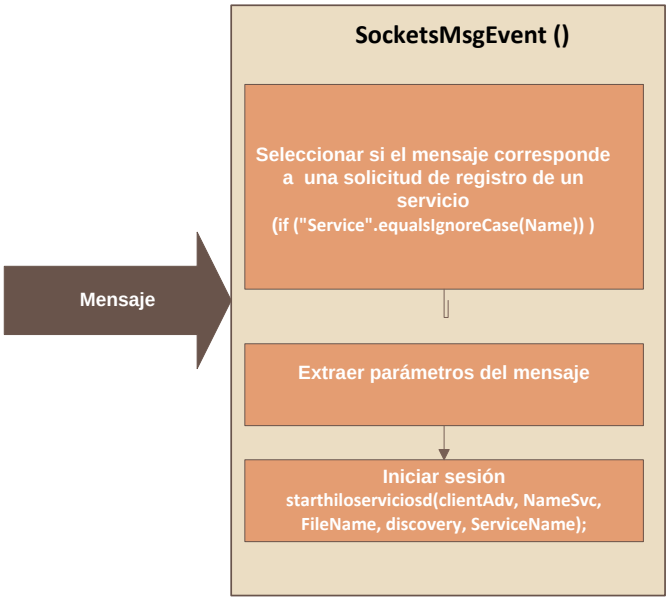


Figura 4.29 Diagrama de bloques correspondiente a la extracción de parámetros de configuración.

Una vez que se inicia la sesión, el coordinador central está en condiciones de aceptar la trasferencia del documento, el cual contiene la descripción jerárquica del servicio. Este documento está descrito en sintaxis propia de la definición de leguaje ontológico OWL.

Una vez que el coordinador central recibe el documento que contiene la descripción de jerarquía del servicio, tiene que extraer la información para cargarla en memoria como un objeto con estructura de tipo árbol, con el fin de seleccionar la jerarquía que hay que seguir y de facilitar la elección de los nodos que se deben crear y agregar a la estructura de árbol que representa la ontología de servicios. Para lograr ese cometido, se debe hacer uso de una API DOM, la cual proporciona una serie de clases y demás métodos, que proporcionan una manera de conseguir cargar los documentos ontológicos en memoria y además permite el acceso y modificación a dichos datos. Véase Figura 4.30.

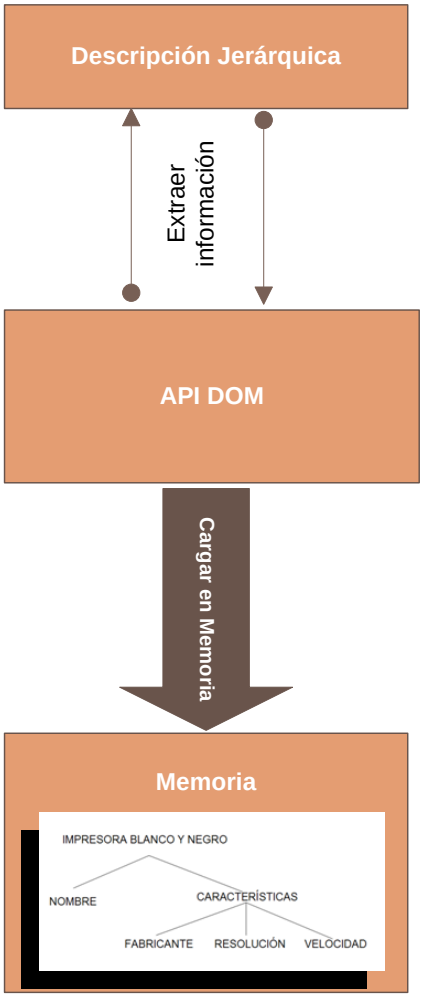


Figura 4.30 Diagrama de bloques correspondiente a la carga en memoria de la jerarquía de clases.

Una vez que se ha cargado la información jerárquica contenida en el documento, en forma de objeto en la memoria, se revisará el contenido de cada uno de los nodos, con el fin de extraer la información necesaria para poder crear y agregar los nodos dentro del árbol que representa la ontología de los servicios en general, el método recorrido, como su nombre lo indica, se encarga de recorrer, por medio de un esquema recursivo nivel a nivel los elementos del árbol que representan al servicio. Véase Figura 4.31.

```
public void recorrido(Node node, int level, Node maindoc, Document main) throws
Exception
{
   listanodos list = node.getChildNodes();
   for (int i=0; i<list.getLength(); i++) {
      Node childNode = list.item(i);
      System.out.println("tmpdoc "+ childNode.getNodeName());
      String name = childNode.getNodeName();
      encontrado = findElementNode(name, maindoc);
      if (encontrado == null)
      {
         nodohijo=childNode;
         String nombrehijo = nodohijo.getNodeName();
         String nombrepadre = nodopadre.getNodeName();
         System.out.println("Nodo hijo " + nombrehijo);
         System.out.println("Nodo Padre "+ nombrepadre);
         boolean resp = addnodo(maindoc, nombrepadre, nodohijo, 0, main);
         recorrido(nodohijo, level+1, maindoc, main);
      }
      else
      {
         nodopadre=encontrado;
         recorrido(childNode, level+1, maindoc, main);
      }
   }
}
```

Figura 4.31 Código encargado de hacer un recorrido por los elementos del árbol jerárquico.

Dicho método además de encargarse de recorrer cada nodo, también debe de verificar si ya existe dicho nodo en la jerarquía de servicios, para lo cual, se auxilia de otro método denominado EncontrarelementoNodo( ), enseguida en la Figura 4.32 se muestra el segmento de código correspondiente a dicho método:

```
public EncontrarelementoNodo(String elementName, Node root)
{
   Node Nodoencontrado = null;

   //Comprueba si la raíz es el elemento deseado.
   String nodeName = root.getNodeName();

   if((nodeName != null) & (nodeName.equals(elementName)))
      return root;

   //Comprueba si la raíz tiene hijos,  si no devuelve null
   if(!(root.hasChildNodes()))
      return null;

   //La raÃíz tiene hijos---seguir buscando.
   Listanodos childNodes = root.getChildNodes();
   int nohijo = childNodes.getLength();
   for(int i = 0; i < noChildren; i++){
      if(Nodoencontrado == null){
         Node child = childNodes.item(i);
         Nodoencontrado = EncontrarelementoNodo(elementName,child);
      } else break;
   }
   return Nodoencontrado;
}
```

Figura 4.32 Código encargado de verificar la existencia de algún nodo.

Por otro lado, si un nodo no existe se deberá crear y agregarlo al nivel jerárquico correspondiente, cabe mencionar que el proceso de adicción de nodos se lleva a cabo por el método agregarnodo( ). Véase Figura 4.33.

4.5.5 Búsqueda semántica y contextual de los servicios.

Un proceso de búsqueda de servicios, es iniciado cuando el coordinador central detecta un mensaje proveniente de un peer móvil, con el objetivo de obtener una lista de los servicios que son ofertados, dicho mensaje es detectado a través del uso del método SocketMsgEvent( ), el cual permite la clasificación de los mensajes que arriban al coordinador central y procede a atender la solicitud. A continuación se muestra el fragmento de código del método SocketMsgEvent( ), el cual detecta si el mensaje que arribo proviene de un dispositivo móvil. Véase Figura 4.34.

```
public boolean agregarnodo (Node main, String nombrepadre, Node nodohijo, int
level, Document main2)throws Exception
{
   listanodos list = main.getChildNodes();
   for (int i=0; i<list.getLength(); i++)
   {
      Node childNode = list.item(i);
      String namep = childNode.getNodeName();
      System.out.println("N "+namep);
      boolean valor = namep.equalsIgnoreCase(nombrepadre);
      if (valor==true)
      {
         NamedNodeMap attrs = nodohijo.getAttributes();
         int numAttrs = attrs.getLength();
         System.out.println(numAttrs);
         if (numAttrs != 0)
         {
            Element elmnode = main2.createElement ( nodohijo.getNodeName());
            nodopadre=nodohijo;
            for (int j=0; j<numAttrs; j++)
            {
               Attr attr = (Attr)attrs.item(j);
               // Obtener atributo name y value.
               String attrName = attr.getNodeName();
               String attrValue = attr.getNodeValue();
               elmnode.setAttribute(attrName, attrValue);
            }
            list.item(i).appendChild(elmnode);
         }
         else
         {
            Element elmnode = main2.createElement (nodohijo.getNodeName());
            list.item(i).appendChild(elmnode);
            nodopadre=nodohijo;
         }
         try
         {
            // Prepara el documento DOM para escritura.
            Source source = new DOMSource(main2);
            // Archivo de salida
            File file = new File("OntologyServices.xml");
            Result result = new StreamResult(file);
            // Write the DOM document to the file
            Transformer xformer =
            TransformerFactory.newInstance().newTransformer();
            xformer.transform(source, result);
         }
         catch (TransformerConfigurationException e) {}
         return true;
      }
      else
      {
         agregarnodo(childNode, nombrepadre, nodohijo, level+1, main2);
      }
   }
   return true;
}
```

Figura 4.33 Código necesario para la adicción de nuevos nodos.

```
public void SocketMsgEvent(PipeMsgEvent event)
{
   try
   {
      Message msg = event.getMessage();
      MessageElement me = msg.getMessageElement(SENDERMESSAGE);
      String clientRequestStream = me.toString();
      System.out.println("El contenido del mensaje es " + clientRequestStream );
      String Name = fromXML(clientRequestStream);
      System.out.println("recibiendo el "+ Name +"'s" + " informacion");
      if ("Clientemovil".equalsIgnoreCase(Name))
      {
         starthiloServicios(clientRequestStream,SvcsContainer,
         IdContainer,LocSvcContainer,NetPeerGroup);
      }
    ...
```

Figura 4.34 Código necesario para detectar peticiones de clientes móviles.

Si dicha solicitud tiene su origen en un peer de tipo móvil, el coordinador central debe atender dicho requerimiento; junto con el mensaje de solicitud semántica de servicios, también se incluye un conjunto de parámetros como son: nombre del servicio, tipo de información del socket de servicio, criterio de búsqueda, ubicación; entre otros, los cuales proporcionan la suficiente información que se requiere para lograr establecer una sesión con el coordinador central, cuyo objetivo primordial es que el mismo, devuelva una respuesta, una vez que obtenga los resultados basados en los criterios de búsqueda que le han solicitado.

La búsqueda se lleva a cabo bajo criterios semánticos, elegidos por el usuario de acuerdo a sus intereses. Esencialmente el coordinador central proporciona dos tipos de criterios de búsqueda que son: búsqueda de todos los servicios existentes y búsqueda personalizada (atributos ó contexto geográfico de los servicios). El código encargado de implementar la búsqueda de todos los servicios existentes se ilustra en la Figura 4.35.

Lo que hace el método es recorrer el árbol que simboliza la jerarquía de servicios en forma recursiva, con el fin de extraer los nombres de los servicios existentes, posteriormente los almacena en una lista que será enviada como respuesta al peer móvil.

```
public List busqueda(Node node, int level, String st) {
   String tipodebusqueda = st;
   if(searchtype.equals("All the services")) {
      listanodos list = node.getChildNodes();
      for (int i=0; i<list.getLength(); i++) {
         Node childNode = list.item(i);
         NamedNodeMap attrs = childNode.getAttributes();
         int numAttrs = attrs.getLength();
         if (numAttrs != 0) {
            slista.agregarnodo(childNode.getNodeName());
         }
         search(childNode, level+1, searchtype);
      }
      return slista;
   }
   return slista;
}
```

Figura 4.35 Código para realizar una búsqueda.

El método encargado de una búsqueda de tipo personalizada de servicios es el denominado perser( ). Véase Figura 4.36.

```
public List perser(Node node, int level, List ListPar) {
   int cont = 0;
   NodeList lista = node.getChildNodes();
   for (int i=0; i<lista.getLength(); i++) {
      Node childNode = lista.item(i);
      NamedNodeMap attrs = childNode.getAttributes();
      int numAttrs = attrs.getLength();
      if (numAttrs > 0) {
         cont = 0;
         for (int j=0; j < ListPar.size(); j++){
            String par = (String)(ListPar.get(j));
            System.out.println(par);
            for (int n = 0 ; n < numAttrs; n++) {
               Attr attr = (Attr)attrs.item(n);
               String attrValue = attr.getNodeValue();
               if (attrValue.equals(par)) {
                  String name = childNode.getNodeName();
                  System.out.println(name);
                  cont = cont + 1;
               } //if
            } //for
         } //for
         if (ListPar.size() == cont)
         perser.agregarnodo(childNode.getNodeName());
      } //if
      getListSvcs(childNode, level+1, ListPar);
   } // for i
   return perser;
}
```

Figura 4.36 Código mediante el cual se ejecutan búsquedas personalizadas

Dicho método hace un recorrido recursivo del árbol para localizar los nombres de los nodos que coincidan con los criterios recibidos, los resultados obtenidos también son almacenados en una lista que de igual manera es enviada al peer móvil en forma de mensaje a través de un socket de comunicación. Véase Figura 4.37.

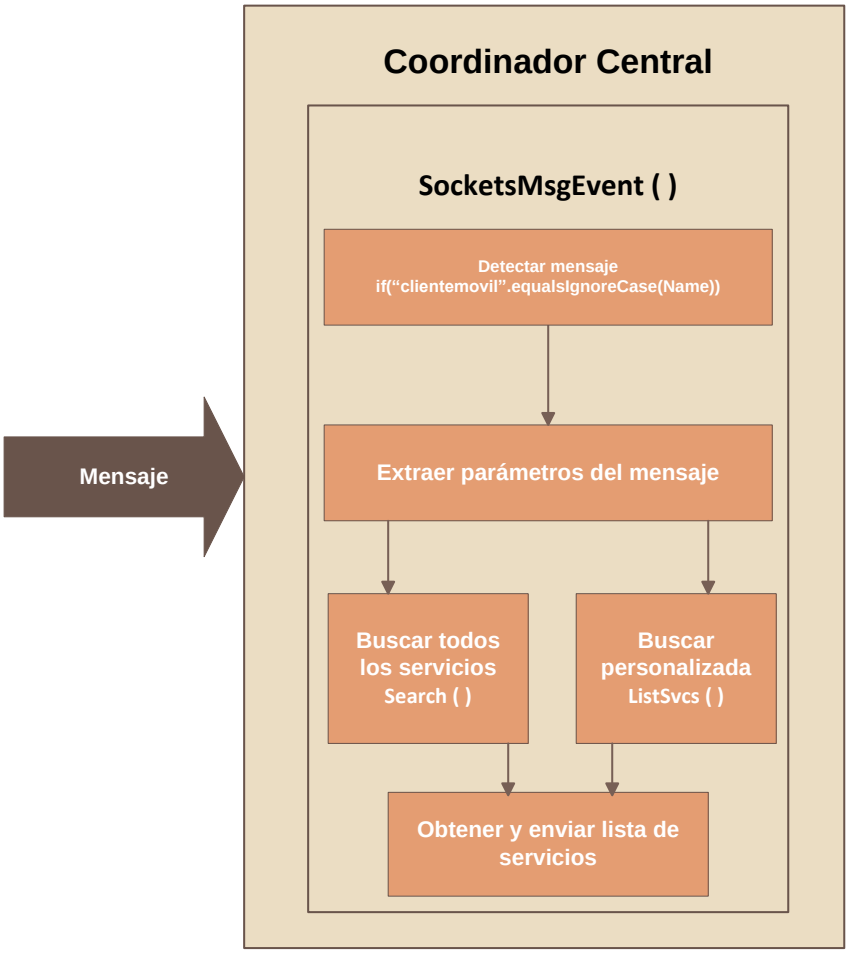


Figura 4.37 Diagrama de bloques correspondiente al proceso de búsqueda semántica de servicios.

4.5.6 Mecanismo de detección de caída y eliminación de los servicios en la jerarquía.

Para lograr mayor desempeño en lo que se refiere a la búsqueda semántica de servicios, es importante mantener la jerarquía de servicios con sólo aquellos servicios que realmente están en línea (funcionando). Debido a ello, es necesario identificar la caída de servicios para proceder a hacer una eliminación de los mismos en la jerarquía. A continuación en la Figura 4.38 se ilustra el código de dicha tarea.

```
public void run() {
   try {
      System.out.println("Empieza subproceso " + NSvc);
      while (flag) {
         expiration = discovery.getAdvLifeTime(sockSvc);
         if (expiration >= 0) {
            flag2 = true;
            flag = false;
         }
      }
      System.out.println("Comenzar nuevo hilo" + NSvc);
      while (flag2) {
         expiration = discovery.getAdvLifeTime(sockSvc);
         if (expiration == -1){
            try {
               System.out.println("El anuncio es "+ NSvc + " expira " +
               expiration);
               removeService();
               Thread.sleep(100);
            }
            catch (Exception e) {
               System.out.println("Error:  " + e.toString());
               e.printStackTrace();
               return;
            }
         }
      }
   }
   catch (Exception e) {
      System.out.println("Error:  " + e.toString());
      e.printStackTrace();
   }
}
```

Figura 4.38 Detección de caída y eliminación de servicios.

Si en la etapa de descubrimiento el coordinador central no encuentra el anuncio de alguno de los servicios que previamente ha sido registrado, se procederá con la eliminación de dicho servicio de la jerarquía, haciendo uso del método removerservicio( ), que se encarga de la tarea de eliminación de servicios inactivos. Véase Figura 4.39.

4.6 Servicios

En el entorno ubicuo propuesto, con fines demostrativos, se implementó el servicio de impresión, lo cual no quiere decir que dicha propuesta esté limitada solo a este servicio. La arquitectura propuesta proporciona un marco mediante el cual en un futuro se pueden adicionar otra clase de servicios sin tener que realizar cambios significativos. Un servicio es representado por un peer JXTA, lo cual se traduce en la necesidad de inicializar el servicio como peer de la red JXTA (de una manera similar al procedimiento llevado a cabo para

lograr inicializar el peer que representa al coordinador central). Una vez realizada la inicialización del peer que representa al servicio, un servicio tiene el siguiente ciclo de operación:

- Registro y publicación del servicio.
- Recepción de solicitud de uso del servicio.
- Recepción de algún archivo y / o parámetros de operación para el servicio.
- Ejecución del servicio.

```
public void removerservicio() throws Exception {
   String mainurl = "file:" + new File("jerarquiaServices.xml").getAbsolutePath();
   try {
      DOMParser mainparser = new DOMParser();
      mainparser.parse(mainurl);
      Document maindoc = mainparser.getDocument();
      ccadmonto = new ccAdmonto();
      parentname = ccadmonto.remove(maindoc,NSvc);
      System.out.println("El parentname es " + parentname);
      while (parentname.equals("Dispositivos") == false)  {
         NSvc=parentname;
         parentname = ccadmonto.remove(maindoc,NSvc);
         ccadmonto.writeXmlToFile("jerarquiaServices.xml",maindoc);
         flag2 = false;
      }
      catch (SAXException saxe) {}
   }
}
```

Figura 4.39 Código por medio del cual se eliminan servicios inactivos.

4.6.1 Registro y publicación del servicio de impresión.

Para llevar a cabo el registro de un servicio, primero el peer que representa al servicio, debe enviar un mensaje al coordinador central, dicho mensaje contendrá el documento basado en sintaxis XML. En la Figura 4.40 se puede apreciar en forma detallada el código responsable de dicha tarea.

```
try {
   Java.io.InputStream SoccketAdvStream;
   sockAdvStream = getInputStream("datos.xml");
   Message message = new Message();
   InputStreamMessageElementisme = new InputStreamMessageElement(SENDERMESSAGE, new
   MimeMediaType("text/xml"), SocketAdvStream, null);
   message.addMessageElement(isme);
   serviceOutputPipe.send(message);
}
catch (Exception e) {
   System.out.println("error al enviar aviso de socket del cliente...");
   e.printStackTrace();
}
```

Figura 4.40 Código encargado del registro y publicación de servicios.

En forma resumida, dicho código se encarga de obtener el documento XML como un flujo de bytes, posteriormente procede a agregarse como un elemento al mensaje y por último, lo envía a través de un socket de salida al coordinador central, con lo que se concluye la primera parte del registro del servicio. La siguiente parte consiste en enviar el documento que contiene la información de la jerarquía. En la Figura 4.41 se puede apreciar el código implementado para dicha tarea.

```
private void enviarFile(String filename) {
   try {
      infile = new FileInputStream(fileName);
      Message message = new Message();
      ByteArrayMessageElement ilclock = new ByteArrayMessageElement(LCLOCK,
      MimeMediaType.AOS, new Integer(ind).toString().getBytes(), null);
      ByteArrayMessageElement icontentMsg = new
      ByteArrayMessageElement(DATA,MimeMediaType.AOS,fileName.getBytes(), null);
      message.addMessageElement(ilclock);
      message.addMessageElement(icontentMsg);
      threadServiceOutputPipe.send(message);
      while((lenght = infile.read(buf))!= -1){
         ind++;
         Message mes = new Message();
         ByteArrayMessageElement lclock = new ByteArrayMessageElement(LCLOCK,
         MimeMediaType.AOS,new Integer(ind).toString().getBytes(), null);
         ByteArrayMessageElement contentMsg;
         if(lenght < BUFSIZE) {
            byte[] finalbuf = new byte[lenght];
            for(int i = 0; i < lenght; finalbuf[i]=buf[i],i++);
            contentMsg = new
            ByteArrayMessageElement(DATA,MimeMediaType.AOS,finalbuf, null);
         }
         else {
            contentMsg = new ByteArrayMessageElement(DATA,MimeMediaType.AOS,
            buf, null);
         }
         mes.addMessageElement(lclock);
         mes.addMessageElement(contentMsg);
         threadServiceOutputPipe.send(mes);
         System.out.println("enviando paquete <"+ ind + "> of length : " +
         buf.length);
      }
   }
}
```

Figura 4.41 Código implementado para el envío de archivos XML.

Esencialmente las actividades que se han realizado, son siempre necesarias para el registro de cualquier servicio que se desee implementar. Véase Figura 4.42.

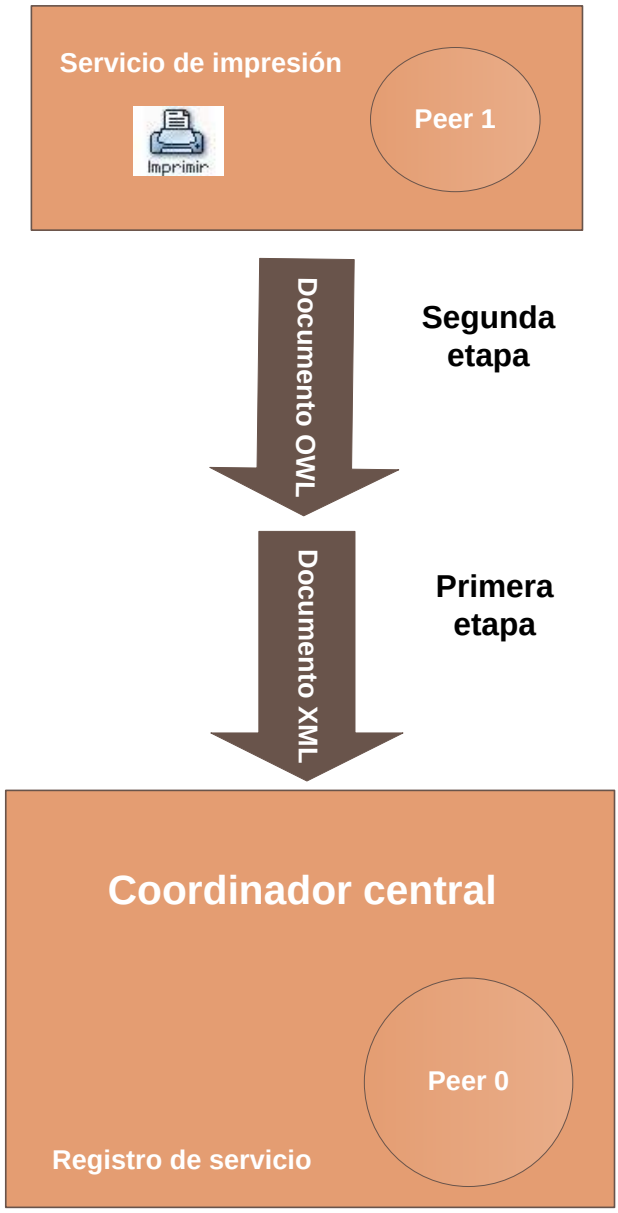


Figura 4.42 Diagrama de bloque indicando pasos a seguir en registro de servicios.

Una vez que el servicio ha sido registrado ante el coordinador central, se procederá a realizar el proceso de publicación del anuncio (Véase Figura 4.43 y Figura 4.44).

```
public void publish() {
   try {
      discovery.publish(inputScocketSvc, DiscoveryService.ADV,ExpTime,LifeTime);
      discovery.publish(inputSocketAdv, DiscoveryService.ADV,ExpTime, LifeTime);
   }
}
```

Figura 4.43 Método responsable de la publicación de un servicio.

Figura 4.44 Diagrama de bloques correspondiente a la publicación de un anuncio.

4.6.2 Recepción de solicitudes de impresión.

En primera instancia se deben detectar los mensajes que arriban al servicio, dicha actividad es llevada a cabo por el método socketMSgEvent ( ). Véase Figura 4.45.

```
public void SocketMsgEvent(SocketMsgEvent event) {
   try
   {
      Message msg = event.getMessage();
      MessageElement serverResponse;
      if((serverResponse=msg.getMessageElement(SENDERMESSAGE)) != null)
      {
         String clientRequestStream =serverResponse.toString();
      }
      socketAdvertisement clientMobAdv = GenerateADV(clientRequestStream);
      System.out.println("\nMOVIL SOCKET ADVERTISEMENT:\nName:  " +
      clientMobAdv.getName() + "\nID:  " +  clientMobAdv.getID() +  "\nType: "+
      clientMobAdv.getType());
      System.out.println("Esperando nuevas conexiones");
      System.out.println("Recibiendo solicitud de trasferencia de: " +
      clientMobAdv.getName());
      starthilo(clientMobAdv,inputdocketsvc);
      return;
   }
   ...
```

Figura 4.45 Código necesario para recibir solicitudes de impresión.

Cuando dicho método detecta el arribo de un mensaje, procede a identificarlo mediante la extracción de la etiqueta con un determinado valor (SENDERMESSAGE), la cual se encuentra inmersa dentro del mensaje que ha arribado. Una vez que se identifica el mensaje como una solicitud de impresión hecha por peer de tipo móvil, se establece el inicio de una sección de comunicación usando un socket de comunicación a través del método iniciarhilo( ). Por último, para finalizar el procedimiento, se debe enviar al peer móvil una notificación de aceptación seguida de un documento de tipo XML, el cual contiene la interfaz de impresión, a través del socket creado en el inicio de sesión.

4.6.3 Recepción de archivos y extracción de parámetros de impresión.

Cuando se logra establecer de forma correcta una sesión entre el peer de tipo servicio y el peer de tipo móvil, ahora el servicio, haciendo uso nuevamente del método socketMsgEvent( ), detecta los mensajes que provengan del peer de tipo móvil a través del socket que ha sido creado previamente durante el inicio de la sesión. Dichos mensajes contienen información del archivo que se desea imprimir, por lo que posteriormente al arribo de los mismos, se debe proceder a extraer de dicho mensaje el contenido del archivo y los parámetros de impresión que han sido indicados por el usuario móvil, en seguida se hace una llamada al método printfile( ), enviando como argumentos del método, los parámetros que fueron obtenidos. En la Figura 4.46 se muestra el segmento de código cuya tarea consiste en la extracción de los parámetros que acompañan al mensaje que ha sido interceptado previamente por el método socketMsgEvent( ). Finalmente, dicho método procederá a realizar la impresión, que se resume en lo descrito en la Figura 4.47

```
{
   printfiles = new printFiles();
   byte[] bufOrientation = Orientation.getBytes(true);
   String orientation = new String(bufOrientation);
   Integer copies = new Integer(new String(Copies.getBytes(true)));
   byte[] bufName = NameMsg.getBytes(true);
   String fileName = new String(bufName);
   if ((fileName.endsWith(".TXT")) || (fileName.endsWith(".txt")))
      typeOfFile = "TXT";
   if ((fileName.endsWith(".PDF")) || (fileName.endsWith(".pdf")))
      typeOfFile = "PDF";
   if ((fileName.endsWith(".RTF")) || (fileName.endsWith(".rtf")))
      typeOfFile = "RTF";
   if ((fileName.endsWith(".PPT")) || (fileName.endsWith(".ppt")))
      typeOfFile = "PPT";
}
```

Figura 4.46 Segmento de código encargado de extraer parámetros de impresión.

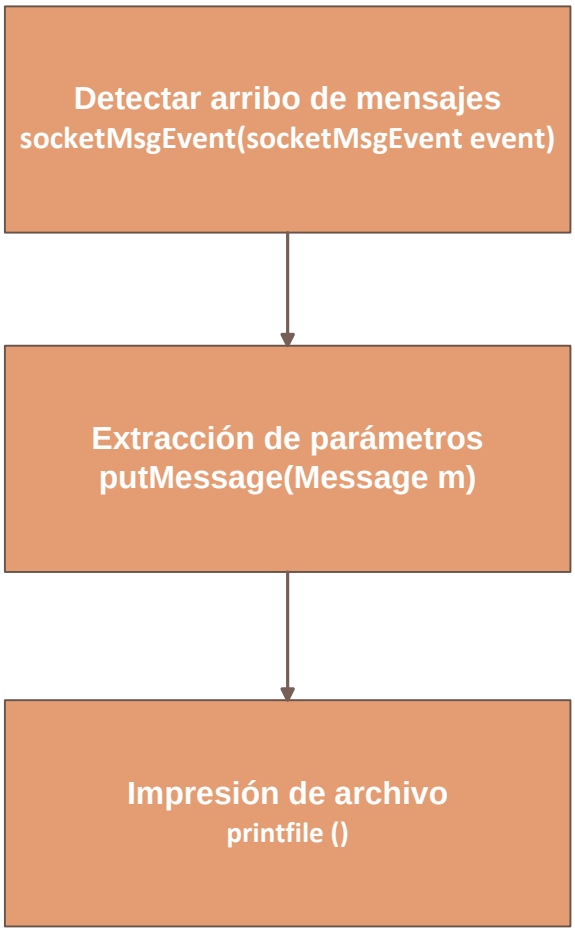


Figura 4.47 Diagrama de bloques indicando la serie de pasos necesarios en el proceso de impresión de un archivo.

4.6.4 Impresión de archivos.

A continuación se muestra el código que hace referencia a la clase printFile, la cual se encarga de realizar la impresión del archivo a través de su único método denominado printfile( ). Véase la siguiente figura.

```
public void printfile(String FileToPrint, String Orientation, String typeOfFile) {
   if (typeOfFile.equalsIgnoreCase("TXT"))
   PATH = "C:\\WINDOWS\\notepad.exe";
   else if (typeOfFile.equalsIgnoreCase("PDF"))
   PATH = " C:\\Program Files\\Adobe\\Reader 9.0\\Reader\\AcroRd32.exe";
   else if (typeOfFile.equalsIgnoreCase("RTF"))
   PATH = "C:\\Program Files\\Windows\\Accessories\\wordpad.exe";
   else if (typeOfFile.equalsIgnoreCase("PPT"))
   PATH = "C:\\Program Files\\Microsoft Office\\Office10\\powerpnt.exe";

   PRINT_COMMAND = "/p";
   SLASH = "/";
   QUOTE = "\"";
   SPACE = " ";
   pFile ="C:\\daniel\\PrinterService\\files\\"+ FileToPrint;
   PrintService service = PrintServiceLookup.lookupDefaultPrintService();

   String lCommand = QUOTE + PATH + QUOTE + SPACE + PRINT_COMMAND + SPACE +
   QUOTE + pFile + QUOTE + SPACE +  QUOTE + service.getName() + QUOTE;
   System.out.println("[print] Command to be executed : " + lCommand);
   Process lPrintProcess = null;
   try {
      lPrintProcess = Runtime.getRuntime().exec(lCommand);
      Thread.sleep(24000);
      lPrintProcess.destroy();
   }
}
```

Figura 4. 48 Código encargado de la tarea de impresión.

Dicho método recibe los parámetros que con anterioridad han sido extraídos, tales como: nombre del archivo, orientación y tipo de archivo. Cabe resaltar que el parámetro tipo de archivo es utilizado para indicar la ruta de acceso al programa ejecutable correspondiente al tipo de archivo. Para hacer la búsqueda del servicio de impresión predefinido (impresora instalada en la estación de trabajo), se hace uso del objeto Java PrintService, y una vez que se cuenta con dicho dato y acompañado de la ruta de acceso al programa ejecutable, se construye un comando de sistema, el cual es ejecutado por el método exec( ), de la clase Rutime, y lanzará un proceso para ejecutar el comando que realizará la impresión del archivo que ha sido enviado por el dispositivo móvil.

4.7 Dispositivos móviles.

En general, los peers JXTA sin importar si son de tipo móvil o no, se comunican entre sí a través de mensajes. Debido a las limitaciones en la comunicación HTTP de los dispositivos J2ME MIDP 2.0 y en general a otro tipo de limitaciones relacionadas con la baja capacidad de memoria para el procesamiento de mensajes XML, dichos artefactos no pueden ser inicializados de igual manera en la red JXTA. Por lo que, para llevar a cabo la

inicialización, los dispositivos móviles deberán hacer uso de una API denominada JXME, la cual permite conectar un dispositivo móvil (J2ME) a redes JXTA. Un peer JXTA basado en J2ME (JXME), sólo puede comunicarse con un peer relay (JXTA), el relay asume la responsabilidad de representar al peer móvil en la red JXTA. Como se dijo anteriormente, la comunicación de los peers J2ME con el relay se realiza a través de peticiones HTTP sucesivas, en un procedimiento conocido como *polling* o sondeo. Mediante este procedimiento, los peers J2ME envían y reciben mensajes JXTA para realizar diferentes operaciones en la red, como las que se relacionan a continuación:

- Creación de grupos de peers y sockets.
- Búsqueda de peers, grupos de peers y sockets.
- Ingreso a grupos de peers.
- Cierre de sockets.

Los mensajes JXME están conformados por unidades llamadas "Elementos" (véase Figura 4.49), encargados de llevar información específica del mensaje, como el tipo y nombre del mensaje, entre otros. Cada elemento, a su vez, se compone de cuatro campos de información, que se describen a continuación:

- **Name**: Campo que describe el nombre del elemento.
- **Data**: Contiene los datos del elemento que son transportados a través de toda la red JXTA.
- **NameSpace**: Describe el espacio de nombres usado por el elemento, los mensajes JXTA usan el namespace "JXTA". Los mensajes en JXME usan un espacio de nombres privado. Si el espacio de nombres es nulo *(null)*, se usa el valor por defecto.
- **MIME Type**: Describe el tipo MIME de los datos que lleva el mensaje. Si se pone un valor de nulo en este campo, automáticamente se asume valor por defecto que es "application/octet-stream".

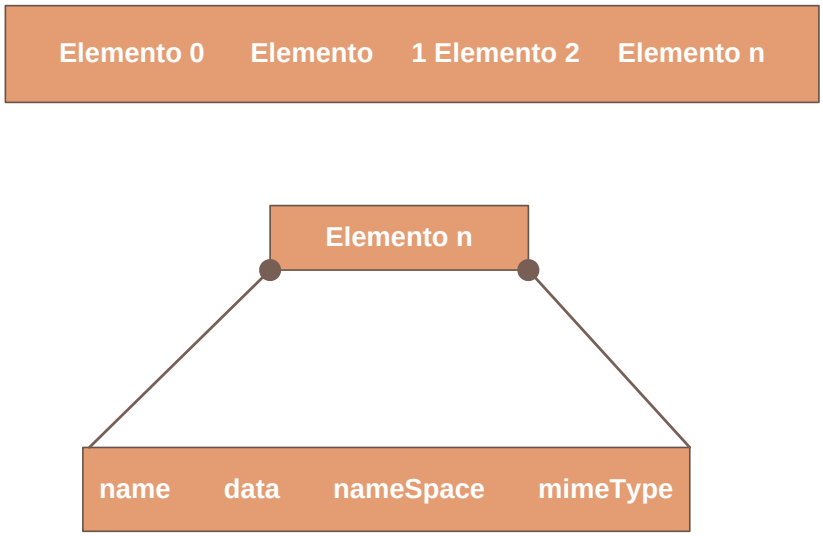


Figura 4.49 Estructura de los mensajes JXME.

En resumen, se puede concluir lo siguiente, un peer basado en J2ME en conjunción con un peer de tipo relay es funcionalmente equivalente a cualquier otro peer de una red JXTA [9]. En la Figura 4.50 se muestra el segmento de código, que mediante los métodos iniateConnect ( ) y connection ( ), lleva a cabo el proceso de conexión hacia un peer de tipo relay. En general el proceso se podría resumir como sigue: una vez que un usuario que disponga de un dispositivo móvil, arriba al entorno ubicuo, y comienza el proceso de conexión (de forma trasparente), primeramente se crea una instancia de PeerNetwork con un nombre previamente especificado, dicha instancia será realizada por el método iniateConnect (), y posteriormente el método connection realizará la conexión con el peer relay, el cual es representado por la URL de la estación de trabajo donde se encuentre dicho peer.

4.7.1 Interfaz instantánea de servicios.

La interfaz instantánea de servicios es la interfaz que deberá ser instalada previamente en el dispositivo móvil, la cual será utilizada por los usuarios para obtener una lista de los servicios que son ofertados en el entorno ubicuo. El código de dicha interfaz se muestra en la Figura 4.51. Este código despliega la interfaz instantánea de servicios en el cliente móvil, ofertando dos opciones de búsqueda de servicios (todos los servicios ó búsqueda personalizada), así como la respectiva opción para seleccionar *(select)* y por supuesto una opción de salir.

```
private void initiateConnect() {
   if (peerNetwork == null) {
      peerNetwork = PeerNetwork.createInstance("ClienteMovil");
   }
   if (connected || connectInitiated) {
      return;
   }
   connectInitiated = true;
}

public boolean connection () throws Exception {
   connectInitiated = false;

   String host = "192.168.1.82";
   int port = 0;
   try {
      port = Integer.parseInt("9705");
      // port = Integer.parseInt("9705");
   }
   catch (NumberFormatException ex) {
      return false;
   }
   String url = "http://" + host + ":" + Integer.toString(port);
   if (DEBUG) {
      System.out.println("Connecting to " + url + "...");
   }
   long startTime, endTime;
   if (QUANTIFY) {
      startTime = System.currentTimeMillis();
   }
   if (QUANTIFY) {
      endTime = System.currentTimeMillis();
      System.out.println("connect took " + Long.toString(endTime-startTime));
   }
   try {
      connected = true;
      state = peerNetwork.connect(url, null);
   }
   ...
```

Figura 4.50 Código correspondiente a la inicialización de un peer móvil en un red JXTA.

```
public Clientemovil() {
   readServices();
   display = Display.getDisplay(this);
   alert = new Alert("Atention!","",null,AlertType.INFO);
   alert.setTimeout(7000);

   cmdExit = new Command("Exit", Command.SCREEN, 1);
   cmdSelect = new Command("Select", Command.SCREEN, 1);
   Ticker WMessage = new Ticker("Bienvenido");
   lsServices = new List("Services", Choice.IMPLICIT);
   lsServices.append("All Services",null);
   lsServices.append("Advanced Search",null);
   lsServices.addCommand(cmdSelect);
   lsServices.addCommand(cmdExit);
   lsServices.setCommandListener(this);
   lsServices.setTicker(WMessage);
}
```

Figura 4.51 Código correspondiente a la interfaz instantánea de servicios.

Si el usuario selecciona la opción “Todos los servicios”, se desplegará una nueva interfaz, la cual contendrá una lista de todos los servicios disponibles, pero en el caso de que el usuario seleccione la opción búsqueda personalizada, se desplegará una interfaz, facilitando la inserción de criterios de búsqueda de los servicios, entre los que sobresalen diversos atributos y/o su ubicación geográfica. Véase Figura 4.52.

```
private void AdvancedForm() {
   if (AdvancedForm == null) {
      AdvancedForm = new Form("Advanced Search");
      SearchPar = new TextField("Attributes:","All",4096,TextField.ANY);
      LocPar = new TextField("Location:","All",4096,TextField.ANY);
      AdvancedForm.append(SearchPar);
      AdvancedForm.append(LocPar);
      cmdSendAdvForm = new Command("Send", Command.SCREEN, 1);
      AdvancedForm.setCommandListener(this) ;
   }
   AdvancedForm.setTitle("Advanced Search");
}
```

Figura 4.52 Código correspondiente a la generación de interfaz de búsqueda personalizada de servicios.

Una vez que se obtiene una lista de los servicios disponibles haciendo uso de cualquiera de los dos métodos de búsqueda previamente mencionados, dicha interfaz provee una opción de salida y otra de selección, con la cual, el usuario puede indicar cuál de los servicios disponibles desea utilizar, al realizar dicha selección se realizará un llamado del servicio y dependiendo del mismo, se desplegará la respectiva interfaz. Véase Figura 4.53.

```
private void readServices() {
   ServicesList = new List("Services List", List.IMPLICIT);
   cmdSelect = new Command("Select", Command.SCREEN, 1);
   cmdBackServicesList = new Command("Exit", Command.BACK, 2);
   ServicesList.addCommand(cmdSelect);
   ServicesList.addCommand(cmdBackServicesList);
   ServicesList.setCommandListener(this);
}
```

Figura 4.53 Código correspondiente a la generación de la lista de servicios disponibles.

4.7.2 Interfaz de servicios.

Cada uno de los servicios que son implementados deben de proveer una interfaz (Véase Figura 4.54) correspondiente al tipo de servicio que ofrece. La interfaz del servicio está

descrita en sintaxis XML (Véase Figura 4.55) y es proporcionada al dispositivo móvil, una vez que el servicio es invocado, para que éste la interprete y la despliegue al usuario.

```
public void printInterface (String Interface) {
   items = parseUsingkXML( Interface );
   if (structure.equals("Form")) {
      ItemForm( option, items, button);
   }
   if (structure.equals("List")) {
      ItemList( items, button, option);
   }
   ...
}

public void ItemForm (String option, String [] items, String button) {
   itemForm = new Form(option);
   for (int i = 0; i < items.length; i ++) {
      itemForm.append(new
      textField(items[i],null,4096,
      TextField.ANY));
   }
   ...
}

private String[] parseUsingkXML( String xml ){
   try {
      ByteArrayInputStream bin =  new
      ByteArrayInputStream( xml.getBytes() );
      InputStreamReader in = new InputStreamReader( bin );
      XmlParser parser = new XmlParser( in );
      Vector    items = new Vector();
      parsekXMLItems( parser, items );

      String[] tmp = new String[ items.size() ];
      items.copyInto( tmp );
      return tmp;
   }
   catch( IOException e ){
      return new String[]{ e.toString() };
   }
   ...
}
```

Figura 4.54 Segmento de código correspondiente a la generación de interfaz de servicios.

```
<?xml version="1.0"?>

  <uidl>
    <interface>
      <structure>Form</structure>
      <title>Servicio de impresión </title>
      <textbox>Archivo</textbox>
      <textbox>Copias</textbox>
      <textbox>Orientación</textbox>
      <button>Enviar</button>
      </interface>
  </uidl>
```

Figura 4.55 Archivo XML, el cual describe la interfaz de un servicio de impresión.

CAPÍTULO 5
PRUEBAS Y RESULTADOS

Resumen. En el capítulo anterior se describió detalladamente cada uno de los componentes que posee la propuesta que en el presente trabajo de tesis se ha planteado, así como la forma en que los mismos convergen para materializar la arquitectura. De igual manera, también se trató lo referente a la implementación, incluyendo todas las características de su forma de trabajo. Con el objetivo de verificar su correcto funcionamiento, el presente capítulo está compuesto por el conjunto de pruebas que se realizaron, mismas que muestran que se cumplió con las expectativas deseadas.

5.1 Pruebas

Primeramente, se debe inicializar el coordinador central, lo cual se consigue mediante el uso de una terminal, ejecutando el siguiente comando:

```
root@usuario: Java –classpath ./ lib/jxta.jar;./lib/bcprov-
jdk14.jar;./lib/Javax.servlet.jar;
./lib/jxta.jar;./lib/jxtashell;./lib/log4j.jar;./lib/org.mortbay.jetty.jar;./lib/xa
lan-2.7.1.jar;./lib/serializer;./lib/xalan.jar; ./lib/xercesImpl.jar;./lib/xml.-
APIs.jar;./lib/xsitc.jar Controlcenter
```

Una vez que se ejecuta dicho comando, el coordinador central inicia de manera correcta y se encuentra listo para recibir conexiones. Véase Figura 5.1

```
Jxta-Config --> set TcpMulticastAddress: 224.0.1.85
Iniciar Configuración
Configuring Jxta BasicServiceInference....
Jxta-Config --> set PeerName: JxtaccPeerName
Jxta-Config --> set HttpPort: 9700
Jxta-Config --> set TcpPort: 9701
Jxta-Config --> set TcpMulticastAddress: 224.0.1.85
Jxta-Config --> set TcpMulticastPort: 1234
Jxta-Config --> set TcpMulticastSize: 16384
Jxta-Config --> set TcpServer: 192.168.1.82:9701
Jxta-Config --> is this Peer a Relay: false
Jxta-Config --> is this Peer Rendezvous: false
Jxta-PlatformConfig-File created!
Finalizar configuración
Jxta-Config --> Configuration saved to: .jxta/PlatformConfig
Iniciando conexión
<INFO  2010-07-16 14:54:25,266 NullConfigurator::<init>:132> JXTA_HOME =
file:/C:/Users/DANIEL/Documents/NetBeansProjects/CC/.jxta/
<INFO  2010-07-16 14:54:25,310 NullConfigurator::adjustLog4JPriority:316> Log4J
[user default] requested, not adjusting logging priority
<INFO  2010-07-16 14:54:28,911 NullConfigurator::adjustLog4JPriority:316> Log4J
[user default] requested, not adjusting logging priority
NetPeerGroup is joint!
Joint the current Group!
Finalizando proceso de conexión
Iniciando hilo de servicios

Obtaining the advertisement for the Coordinador Central...

Pipe Advertisement:
Name:  BasicServiceInferenceAdv
ID:  urn:jxta:uuid-
59616261646162614E50472050325033603BA68F6A604CFC9B36A276244A643B04
Type: JxtaUnicast
Esperando Conexiones........
```

Figura 5.1 Pantalla de inicialización del Coordinador Central.

Después de que el coordinador central inicializa de forma satisfactoria, se agregan los servicios a ofrecer en el entorno ubicuo. En este caso, para fines demostrativos y de prueba se adicionó el servicio de impresión (Véase Figura 5.2 y Figura 5.3) mediante la ejecución de los siguientes comandos en una terminal:

```
root@usuario: Java –classpath ./ lib/jxta.jar;./lib/bcprov-
jdk14.jar;./lib/Javax.servlet.jar;
./lib/jxta.jar;./lib/jxtashell;./lib/log4j.jar;./lib/org.mortbay.jetty.jar;./lib/xa
lan-2.7.1.jar;./lib/serializer;./lib/xalan.jar; ./lib/xercesImpl.jar;./lib/xml.-
APIs.jar;./lib/xsitc.jar ServicioImpresión.
```

Figura 5.2 Proceso de inicialización del servicio de impresión.

Una vez que inicializan el coordinador central, los servicios que serán ofertados así como su correspondiente registro, se montó la infraestructura JXTA para poder hacer las pruebas que permitieron comprobar un correcto funcionamiento de la interfaz instantánea de los servicios (la cual fue instalada previamente en el dispositivo móvil que portará el usuario). Las pruebas que se realizaron fueron las siguientes:

Prueba 1. A través del dispositivo móvil, se realizó una búsqueda general de los servicios, que están disponibles en el entorno ubicuo, de tal manera que los resultados fueran desplegados en el dispositivo mediante la interfaz. El procedimiento para ello se puede resumir como sigue:

En primera instancia se ingresó a la interfaz instantánea de servicios, una vez realizado lo anterior, la interfaz mostró dos opciones de búsqueda: la primera permite hacer una búsqueda de todos los servicios disponibles y la segunda opción permite hacer una búsqueda personalizada. Debido a que el objetivo de esta prueba consistió, en hacer una búsqueda masiva de servicios, se seleccionó la primera opción “Todos los servicios”. A continuación la interfaz instantánea de servicios, mostró una nueva pantalla, donde se presentó el estado del procedimiento, el cual estaba llevando a cabo la conexión, cuando dicha conexión se estableció, la interfaz mostró una nueva pantalla de estado indicando la búsqueda de servicios, cuando se obtuvo una repuesta proveniente del coordinador central, con relación a la búsqueda que se requirió, la interfaz desplegó una nueva pantalla en la cual se enlistaron todos los servicios disponibles en el entorno ubicuo. Véase Figura 5.4

```
run:
Jxta-Config --> set TcpMulticastAddress: 224.0.1.85
It begins configuration......
Jxta-Config --> set debugLevel: error
Configuring Service of Projection....
Logger org.apache.log4j.Logger@152c4d9  now has level DEBUG
Jxta-Config --> set PeerName: JxtaPrintSvcPeerName
Jxta-Config --> set HttpPort: 9702
Jxta-Config --> set TcpPort: 9703
Jxta-Config --> set TcpMulticastAddress: 224.0.1.85
Jxta-Config --> set TcpMulticastPort: 1234
Jxta-Config --> set TcpMulticastSize: 16384
Jxta-Config --> set TcpServer: 192.168.1.82:9703
Jxta-Config --> is this Peer a Relay: false
Jxta-Config --> is this Peer Rendezvous: false
Finished configuration...
Jxta-PlatformConfig-File created!
Finished configuration......
Jxta-Config --> Configuration saved to: .jxta/PlatformConfig
It begins connection..
<INFO  2010-07-16 15:03:03,321 NullConfigurator::<init>:132> JXTA_HOME =
file:/C:/Users/DANIEL/Documents/NetBeansProjects/PrinterService/.jxta/
<INFO  2010-07-16 15:03:03,383 NullConfigurator::adjustLog4JPriority:316> Log4J [user default]
requested, not adjusting logging priority
<INFO  2010-07-16 15:03:04,243 NullConfigurator::adjustLog4JPriority:316> Log4J [user default]
requested, not adjusting logging priority
NetPeerGroup is joint!
Joint the current Group!
Set the services!
Is Rendezvous: false
Is connected to Rendevous: false
Finished connection..

Geting Coordinador Central advertisements
Coordinador Central ADV:
CoordinadorCentralAdv
getOutputPipeBasicServiceInference: Waiting for the output socket event...
Sending a message looking for socket: CoordinadorCentralAdv <--> urn:jxta:uuid-
59616261646162614E50472050325033603BA68F6A604CFC9B36A276244A643B04
Receving a Socket binding response...done!
Creating a new client advertisement for the session...
INPUT PIPE ADVERTISEMENT:Name:  ClientServicedoscketAdv
ID: urn:jxta:uuid-59616261646162614E50472050325033A5F5120E55A24E84A15EBE90C2259F5F04
Type: JxtaUnicast
INPUT socket ADVERTISEMENT SERVICE:
Name:  ServicePipeAdv ID:  urn:jxta:uuid-
59616261646162614E50472050325033C05C02F285424626B2D2B07A4549C67804
Type: JxtaUnicast
THREAD Socket ADVERTISEMENT:
Name:  threadFileTransferenceSocket/pipeAdv
ID:  urn:jxta:uuid-59616261646162614E5047205032503316285A7782004DCB9116BA17E8E2B4BB04
Type: JxtaUnicast
establishSession: Waiting for the output pipe event...
Sending a message looking for socket: threadFileTransferencesocketAdv <--> urn:jxta:uuid-
59616261646162614E5047205032503316285A7782004DCB9116BA17E8E2B4BB04
Receving a socket binding response...done!
Send File..
Sending package <1> of length : 6144
Transference completed!
Archivo enviado..
Waiting connections from Client USE
Connected to the Rendezvous-Peer: urn:jxta:uuid-
59616261646162614A7874615032503323E1B56821534C7EAEAA96B38441A03403
RendezvousEvent: Is Rendezvous? -> false
RendezvousEvent: Is connected to Rendevous? -> true
```

Figura 5.3 Proceso de registro del servicio de impresión.

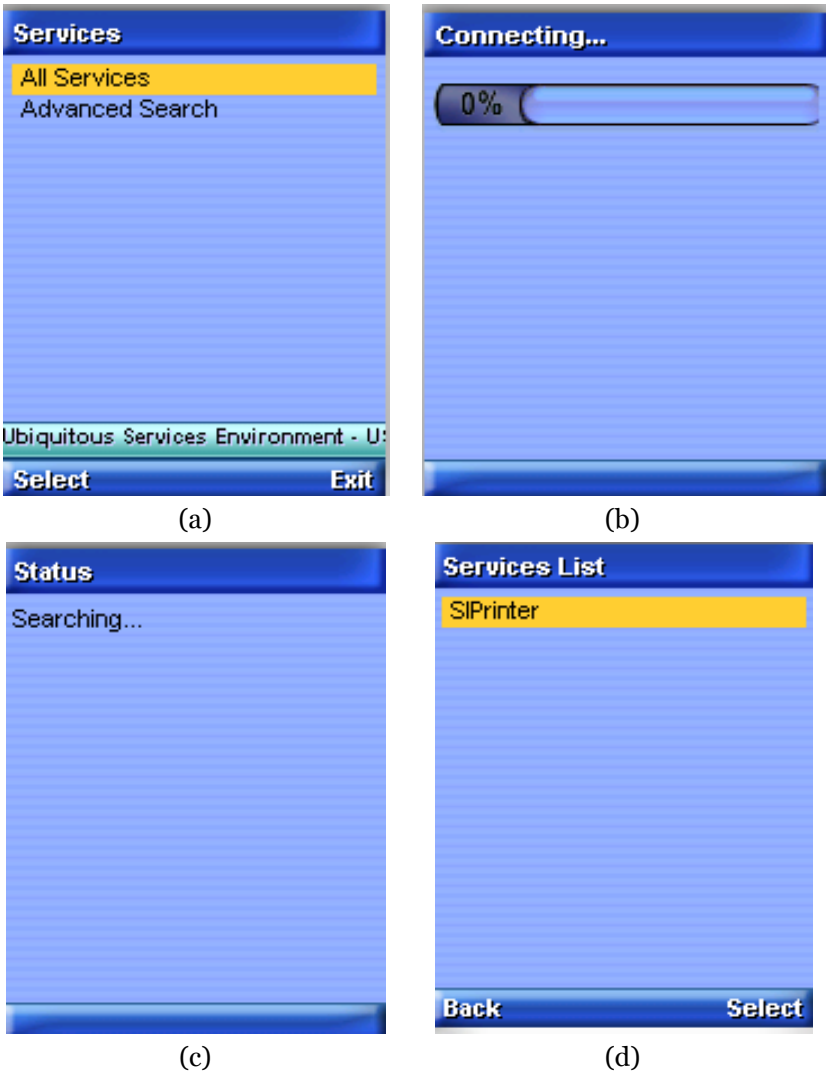


Figura 5.4 (a) Opción de búsqueda. (b) Intento de conexión. (c) Búsqueda de servicios. (d) Servicios encontrados.

Prueba 2. Con el propósito de comprobar que al realizar una búsqueda general sólo se localicen los servicios que realmente se encuentren en línea y no falsas referencias a servicios fuera de línea o inexistentes; previamente se dieron de baja todos los servicios disponibles en el entorno ubicuo, luego se procedió, a través del dispositivo móvil a realizar nuevamente una búsqueda general de los servicios disponibles en el entorno ubicuo, de tal manera que se obtuvo un resultado de cero servicios disponibles, cuyo estado es desplegado en el dispositivo mediante la interfaz. Véase Figura 5.5.

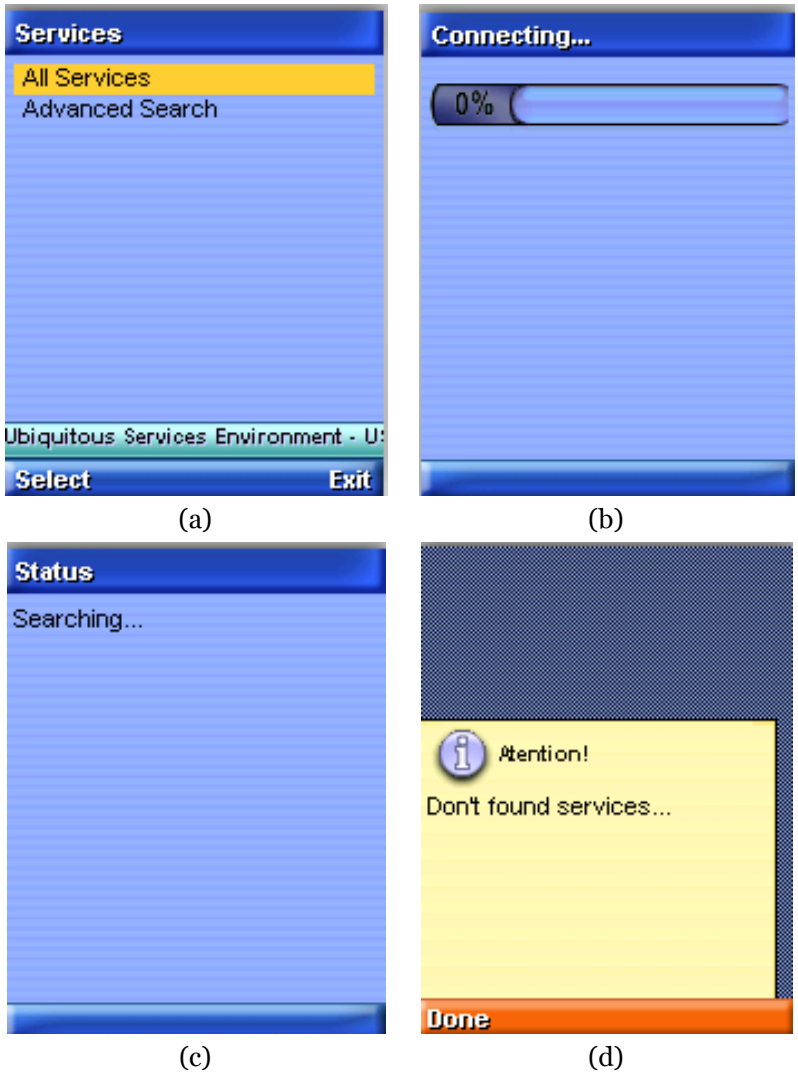


Figura 5.5 (a) Opciones de búsqueda. (b) Intento de conexión. (c) Búsqueda de servicios. (d) Resultado indicando que no hay servicios disponibles.

Primeramente se ingresa a la interfaz instantánea de servicios, una vez realizado lo anterior, la interfaz muestra dos opciones de búsqueda, la primera permite hacer una búsqueda de todos los servicios disponibles y la segunda opción permite hacer una búsqueda personalizada. Debido a que el objetivo consiste en hacer una búsqueda masiva de servicios, se selecciona la primera opción "Todos los servicios".

Posteriormente la interfaz instantánea de servicios mostró una nueva pantalla, donde se presentó el estado del procedimiento, el cual estaba llevando a cabo la conexión, cuando dicha conexión se estableció, la interfaz mostró una nueva pantalla de estado, indicando la búsqueda de servicios, cuando se obtuvo una repuesta proveniente del coordinador central, con relación a la búsqueda que se requirió, la interfaz desplegó una nueva pantalla en donde se mostró un mensaje indicando que no se localizaron servicios disponibles, este

resultado se debe a que, como se comentó anteriormente, se dieron de baja los servicios en el entorno ubicuo con el objetivo de llevar a cabo la presente prueba.

Prueba 3. Otra prueba que se llevo a cabo, fue la búsqueda personalizada, la cual se basa en información de contexto, para ello al invocar la interfaz instantánea de servicios, en esta ocasión se seleccionó la opción número dos (búsqueda personalizada), de tal manera que la búsqueda se realice con base en una serie de parámetros seleccionados. Es importante mencionar que los parámetros admitidos por la interfaz instantánea de servicios son esencialmente: atributos y ubicación, siendo el segundo el que fue usado en esta ocasión para los propósitos de prueba. El procedimiento se puede resumir como sigue:

Se ejecuta la interfaz instantánea de servicios, e inmediatamente se despliegan las opciones de búsqueda, de las cuales se elige "búsqueda personalizada", al seleccionar dicha opción, la interfaz muestra otra pantalla, la cual permite introducir los parámetros de búsqueda. Debido a que la prueba consiste en hacer una búsqueda basada en localización, se indica dicho parámetro de búsqueda, es decir el área donde se requiere identificar los servicios existentes. Después de finalizar la introducción de parámetros, se elige la opción de enviar (send), para iniciar el proceso de conexión, y posteriormente enviar los parámetros de búsqueda, dicho proceso fue indicado por una nueva pantalla donde aparece el estatus del mismo. Una vez que se consiguió establecer la conexión y enviar los parámetros, la interfaz muestra una nueva pantalla, donde se indica que se está llevando a cabo la búsqueda. Cuando la búsqueda recibe respuesta por parte del coordinador central, la interfaz muestra en una nueva pantalla los resultados obtenidos. Véase Figura 5.6.

Continuando con las pruebas de búsqueda personalizada, se ejecutó la interfaz instantánea de servicios, e inmediatamente se desplegaron las opciones de búsqueda, de las cuales se eligió "búsqueda personalizada", al seleccionar dicha opción, la interfaz mostró otra pantalla, la cual permite introducir los parámetros de búsqueda. Sólo que en esta ocasión la prueba consistió en una búsqueda por atributos, por ello se indicó dicho atributo, es decir, alguna característica que pertenezca a cierto servicio (ejemplo: marca del fabricante), por medio del cual se consiga la identificación del mismo, si existe, al terminar de proporcionar los parámetros, se eligió la opción de enviar *(send)* para iniciar el proceso de conexión, y posteriormente enviar los parámetros de búsqueda. Dicho proceso fue indicado por una nueva pantalla donde aparece el estado del mismo, una vez que se consiguió establecer la conexión y enviar los parámetros, la interfaz provee de una nueva pantalla, donde se indicó que se estaba llevando a cabo la búsqueda, cuando dicha búsqueda recibió respuesta por parte del coordinador central, la interfaz mostró en una nueva pantalla los resultados obtenidos. Véase Figura 5.7.

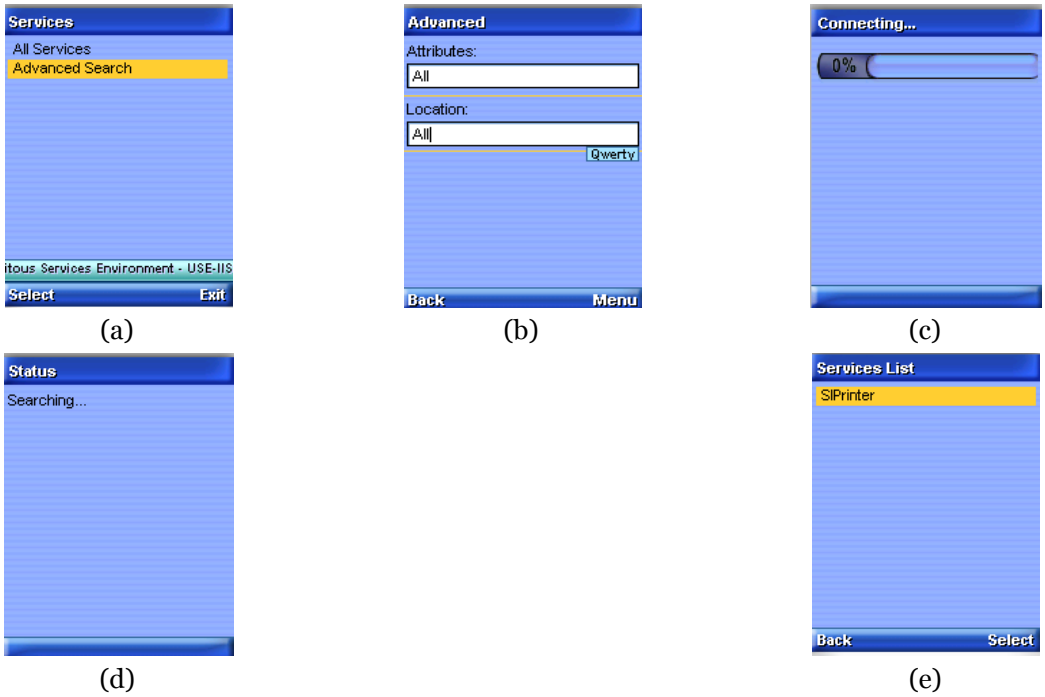


Figura 5.6 (A) Búsqueda Avanzada. (B) Parámetros de Búsqueda. (C) Intento de conexión. (D) Búsqueda de Servicio especificado. (E) Servicio localizado.

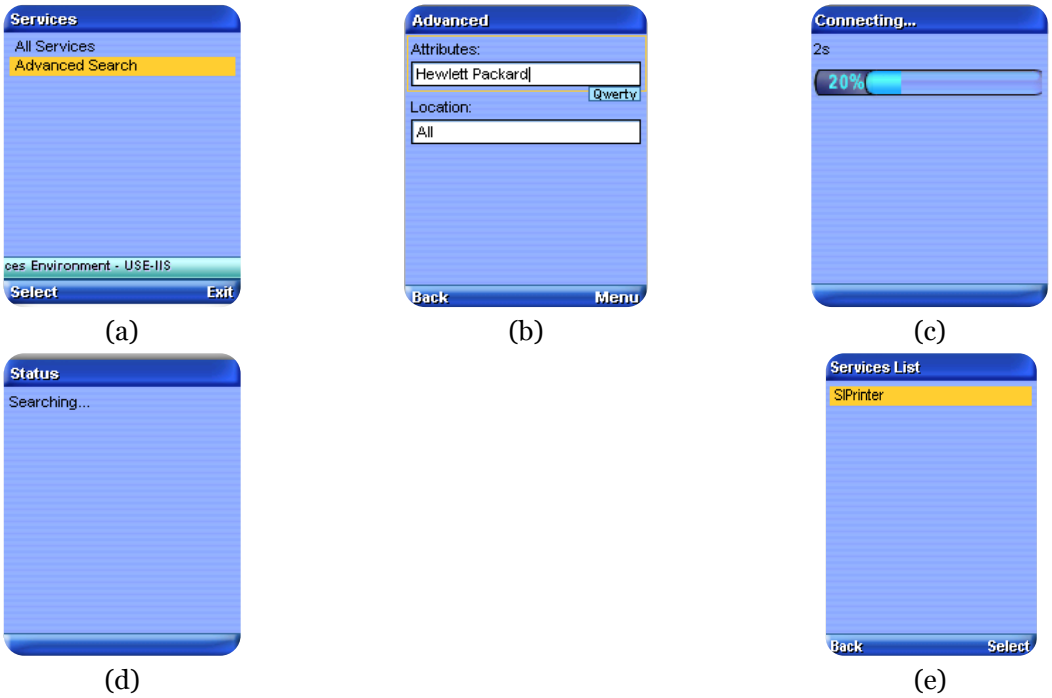


Figura 5.7 (A) Búsqueda personalizada. (B) Parámetro de búsqueda. (C) Intento de conexión. (D) Búsqueda de servicios. (E) Servicios encontrados.

Prueba 4. Una prueba más que se llevó a cabo, en cuanto a la opción de búsqueda avanzada, fue la siguiente: se realizó una nueva búsqueda, sólo que en esta ocasión se especificaron atributos y parámetros de localización para dicha búsqueda, ver Figura 5.8. Se consideró necesario hacer una prueba más, proporcionando parámetros de un servicio inexistente, con el fin de valorar al 100% cualquier posibilidad que tenga que ver con la opción de búsqueda avanzada, los resultados se pueden apreciar en la Figura 5.9.

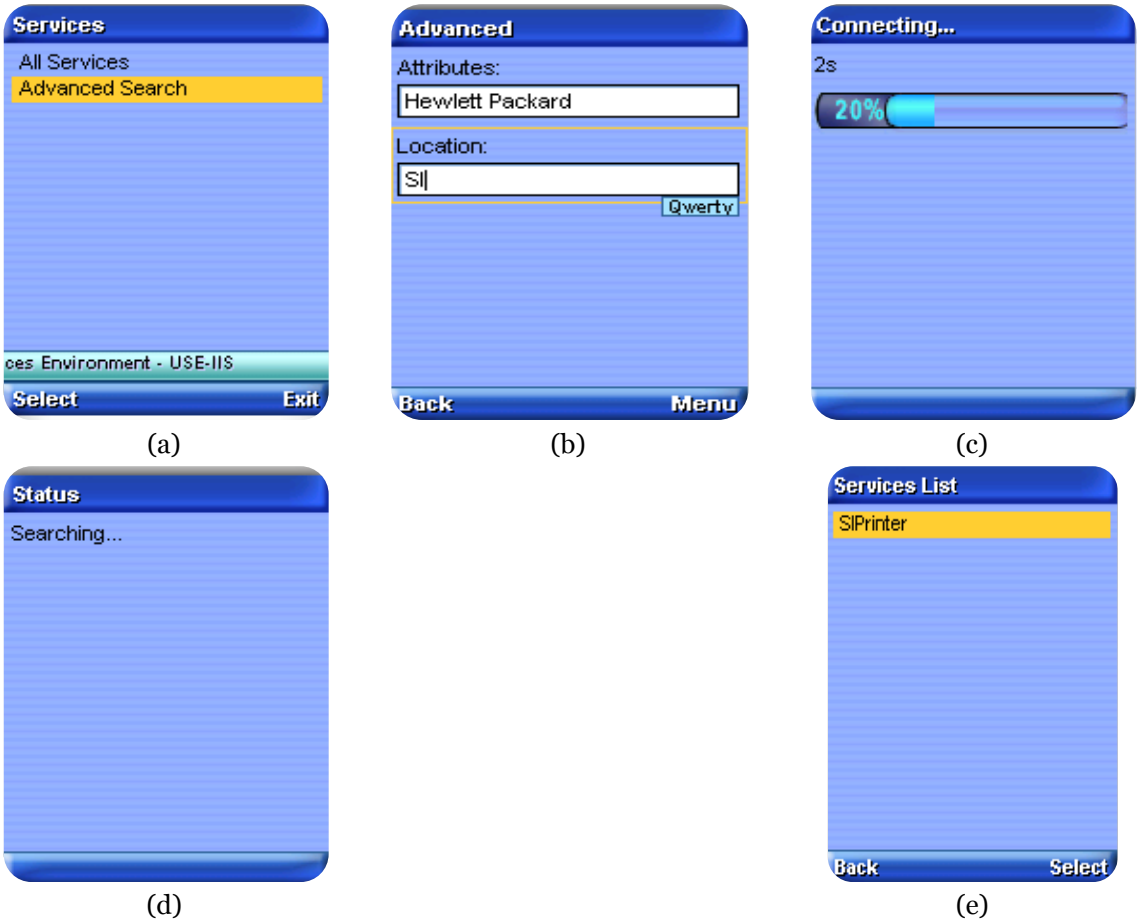


Figura 5.8 (a) Búsqueda personalizada. (b) Parámetro de búsqueda. (c) Intento de conexión. (d) Búsqueda de servicios. (E) Servicios encontrados.

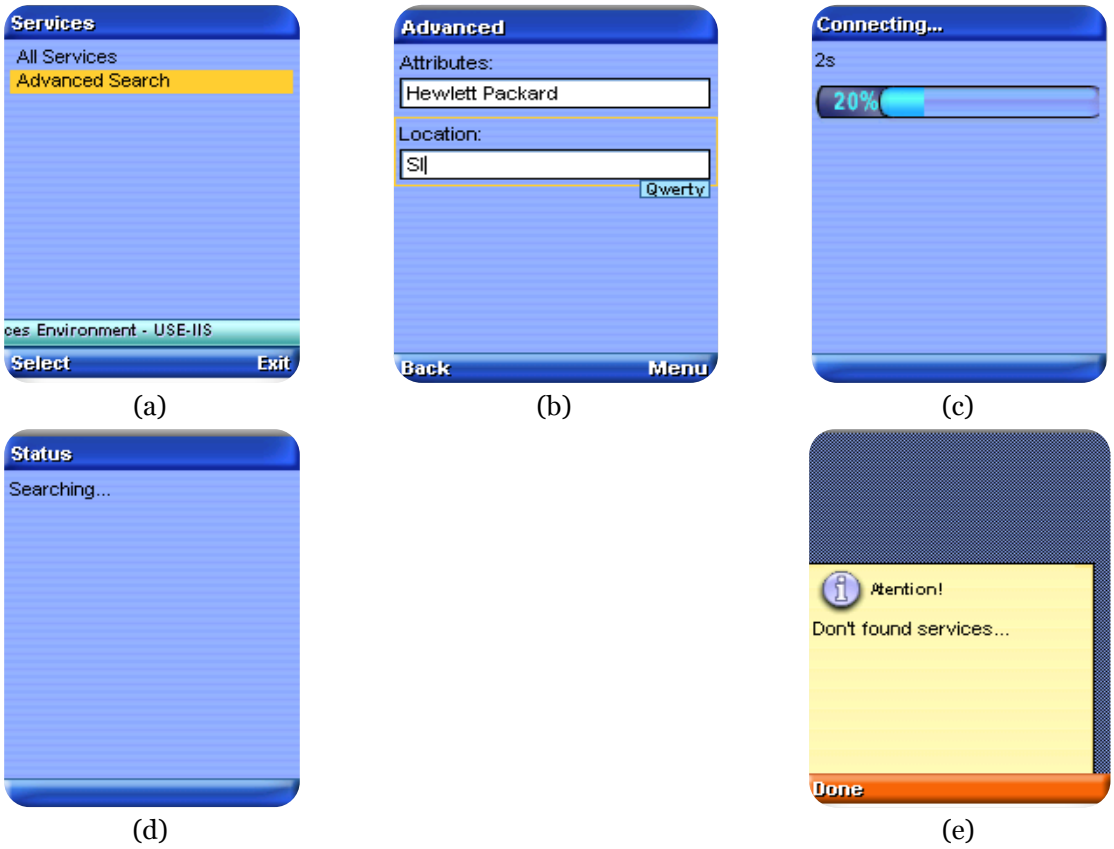


Figura 5.9 (a) Búsqueda personalizada. (b) Selección de parámetros de búsqueda. (c) Intento de conexión. (d) Búsqueda de servicios. (e) No se localizaron servicios.

Prueba 5. Una vez que se concluyó con las pruebas en cuanto a la búsqueda de servicios se refiere, se hicieron pruebas acerca del uso de los servicios localizados por medio del dispositivo móvil; se dieron de alta dos tipos de servicios, uno de impresión y otro de proyección. A continuación se muestra el procedimiento seguido para el uso de cada uno de los servicios antes mencionados.

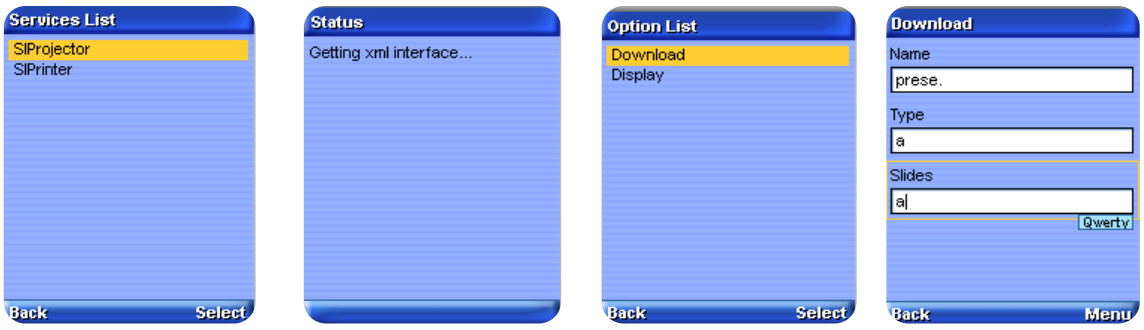
Hablando específicamente del servicio de impresión. Una vez que se obtuvo la lista de servicios disponibles, entre ellos apareció el de impresión, el cual fue seleccionado. Posteriormente, la interfaz instantánea de servicios mostró una pantalla (interfaz), indicando que se solicitó la interfaz del servicio que previamente se ha seleccionado (cabe recordar que dicha interfaz es enviada como un archivo XML por el servicio requerido hacia el dispositivo móvil). Ya que el dispositivo móvil recibió el archivo XML, la interfaz muestra una pantalla, en la cual se indicó que está construyendo la interfaz (Nota: dicha interfaz es construida con base en el archivo XML recibido). Cuando se culminó dicho proceso, entonces la interfaz instantánea de servicios, mostró una nueva pantalla, en la cual se solicitó el nombre del archivo y una serie de parámetros (tipo, orientación; entre otros), los cuales fueron accedidos. Para continuar, se seleccionó enviar, de tal forma que dichos datos fueron enviados al servicio correspondiente, que por su parte recibió dichos

datos y continuó con el proceso de impresión (cabe mencionar que el estado de dicho proceso se visualizó a través de una terminal de comandos). Véase Figura 5.10.

(a) (b) (c)

Figura 5.10 (a) Servicio de impresión. (b) Generación de interfaz de servicio. (C) interfaz de Servicio.

Finalmente, se realizó una prueba más, similar a la anterior con el propósito de comprobar el uso del servicio de proyección. Una vez que se obtuvo la lista de servicios disponibles, en esta ocasión se ha seleccionado proyección, posteriormente, la interfaz instantánea de servicios mostró una pantalla (interfaz), indicando que se solicitó la interfaz del servicio que previamente se ha seleccionado (cabe recordar que dicha interfaz es enviada como un archivo XML por el servicio requerido hacia el dispositivo móvil). Ya que el dispositivo móvil recibió el archivo XML, la interfaz mostró una pantalla, en la cual se indicó que está construyendo la interfaz (Nota: Dicha interfaz es construida con base en el archivo XML recibido).Véase Figura 5.11.



(a) (b) (c) (d)

Figura 5.11 (a) Servicio de proyección. (b) Generación de interfaz de servicio (c) Parámetros de servicio. (d) Parámetros de archivo.

Cuando se terminó el proceso de trasmisión, entonces la interfaz instantánea de servicios, mostró, una nueva pantalla, en la cual se solicitó el nombre, tipo y cantidad de imágenes que conforman la presentación a proyectar, dichos parámetros fueron introducidos (archivo, jpeg, 5), para continuar se seleccionó down, de tal manera que 5 imágenes de tipo jpeg, que pertenecen al archivo, fueron enviados al servicio correspondiente, que por su parte, recibió dichos datos y continuó (cabe mencionar que el estado de dicho proceso se visualizo a través de una terminal de comandos). En el trascurso del proceso de trasmisión de datos, en la terminal móvil apareció una nueva interfaz que decía descargando (downloading). Cuando se culminó el proceso de trasferencia, el dispositivo móvil recibió una notificación y se mostró una pantalla en la cual se indicó que ahora se está solicitando al servicio su interfaz de control. Ya que el dispositivo móvil recibió el archivo XML, la interfaz mostró una pantalla, en la cual se indicó que está construyendo la interfaz, cuando terminó el proceso de construcción, la interfaz instantánea de servicios, mostró, una nueva pantalla, que cuenta con las siguientes opciones: start, next, previous y close. Al seleccionar la opción start, dará inicio a la proyección, por otro lado, para manipular la presentación se hicieron uso de las demás opciones.

5.1.1 Pruebas de Seguridad.

En el capítulo 4, sección 4.4 se habló de la necesidad de desarrollar una API en lenguaje Java para la codificación y decodificación, basadas en librerías de criptografía libres (Bouncy Castle), ver Tabla 5.1. Dicha API, fue desarrollada y embebida dentro de la infraestructura del entorno ubicuo, no obstante, para comprobar el funcionamiento y medir el desempeño de dichas API de seguridad, se hicieron pruebas con cada una de ellas, a través del uso de las herramientas: IDE NetBeans 6.7.1 con Java ME Platform SDK 3.0, a través de las cuales se logró codificar y decodificar una serie de bytes aleatorios, cada uno de ellos con diversas características (tamaño, tipo; entro otros). Cabe resaltar que en lo que se refiere al dispositivo móvil, se realizó una configuración CLDC 1.1 y un perfil MIDP 2.0, ello con la finalidad de conseguir soporte de aritmética flotante en dicho dispositivo, ya que el uso de las librerías criptográficas Bouncy Castle lo requieren. Es importante mencionar que en versiones anteriores de Java para dispositivos móviles (ME) no se proporcionaba soporte de aritmética de punto flotante, lo cual limitaba significativamente el desarrollo de algoritmos criptográficos.

Tabla 5.1 Algoritmos basados en librerias Bouncy castle.

Funcionalidad	Algoritmos
Criptografía simétrica	DES, 3DES, IDEA, AES
Criptografía asimétrica	Diffie-Hellman, RSA, ECC
Códigos hash	MD2, MD4, MD5, RIMPED128/160, Whirlpool, SHA1, SHA256

A continuación en la Figura 5.12 y Figura 5.13 se ilustran algunos fragmentos más significativos de código desarrollado para la implementación de los algoritmos usados para la creación de las API de seguridad. Cabe resaltar que uno de los beneficios más sobresalientes que se consiguen al hacer uso de librerías Bouncy Castle, es que se logra implementar las API de seguridad, tanto en una estación de trabajo como en un dispositivo móvil sin la necesidad de hacer grandes cambios en el código fuente.

```
*/AES 256 (BOUNCY CASTLE)  Author : DANIEL MARTINEZ SALINAS.*/

private BufferedBlockCipher getCipher(final boolean forEncryption) {
   final BlockCipher aesEngine = new AESEngine();
   final BufferedBlockCipher cipher = new
   PaddedBufferedBlockCipher(aesEngine, new PKCS7Padding());
   cipher.init(forEncryption, new KeyParameter(rawKey));
   return cipher;

private byte[] encode(byte[] inputBytes) throws Exception {
   final BufferedBlockCipher cipher = getCipher(true);
   final byte[] outputBytes = new
   byte[cipher.getOutputSize(inputBytes.length)];

   int outputLen = cipher.processBytes(inputBytes, 0, inputBytes.length,
      outputBytes, 0);
   outputLen += cipher.doFinal(outputBytes, outputLen);
   final byte[] finalBytes = new byte[outputLen];
   System.arraycopy(outputBytes, 0, finalBytes, 0, outputLen);
   return finalBytes;
}

public String encryptToBase64(String data) throws Exception {
   final byte[] newData = EncryptionUtils.getBytes(data);
   length%16 = 0
   return Base64.encodeBase64String(encode(newData));
}
```

Figura 5.12 Código para el cifrado AES 256 usado para proporcionar seguridad al entorno ubicuo.

```
package appayudas;

/**
 *
 * @author Daniel Martinez Salinas
 */

import org.bouncycastle.crypto.*;
import org.bouncycastle.crypto.paddings.*;
import org.bouncycastle.crypto.engines.*;
import org.bouncycastle.crypto.modes.*;
import org.bouncycastle.crypto.params.*;

public class EncriptionSource {

   BlockCipher engine = new DESedeEngine();
   public byte[] Encrypt(byte[] key, String plainText) {
      byte[] ptBytes = plainText.getBytes();
      BufferedBlockCipher cipher = new PaddedBufferedBlockCipher(new
      CBCBlockCipher(engine));
      cipher.init(true, new KeyParameter(key));
      byte[] rv = new byte[cipher.getOutputSize(ptBytes.length)];
      int tam = cipher.processBytes(ptBytes, 0, ptBytes.length, rv, 0);
      try {
         cipher.doFinal(rv, tam);
      }
      catch (Exception ce)
      {
         ce.printStackTrace();
      }
      return rv;
   }

   public String Decrypt(byte[] key, byte[] cipherText) {
      BufferedBlockCipher cipher = new PaddedBufferedBlockCipher(new
      CBCBlockCipher(engine));
      cipher.init(false, new KeyParameter(key));
      byte[] rv = new byte[cipher.getOutputSize(cipherText.length)];
      int tam = cipher.processBytes(cipherText,0, cipherText.length, rv, 0);
      try {
         cipher.doFinal(rv, tam);
      } catch (Exception ce) {
         ce.printStackTrace();
      }
      return new String(rv).trim();
   }
}
```

Figura 5.13 Código para encriptar y desencriptar basado en TripleDES.

5.1.2 Análisis de los resultados producidos.

Para llevar a cabo el conjunto de pruebas previamente enunciadas, se ha hecho uso de Java(TM) ME Platform SDK 3.0, con la finalidad de lograr emular el comportamiento de un dispositivo móvil en el entorno ubicuo implementado. De acuerdo con los resultados que se consiguieron, los tiempos de latencia fueron tolerables, estableciendo como referencia un tiempo de sondeo desde el dispositivo móvil al entorno ubicuo implementado. (Ver Tabla 5.2)

Tabla 5.2 Resultados de tráfico en pruebas de iteración entre el dispositivo móvil y el entorno ubicuo.

Análisis del tráfico resultado de la iteración entre dispositivo móvil y peer relay.	
Tiempo máximo de establecimiento de conexión (Segundos.)	50
Retardo máximo entre el envió y retardo máximo de un mensaje (Segundos.)	15
Retardo promedio entre el envió y retardo máximo de un mensaje (Segundos.)	7
Volumen promedio de datos en 6 minutos (Kbyte)	Enviado: 45,806KB Recibido: 28,23KB Total: 74,041KB

Por otro lado, adicionalmente se logró obtener datos adicionales relacionados con el intercambio de información entre los dispositivos móviles y el entorno ubicuo, haciendo uso del programa ip sniffer, el cual incorpora un analizador de tráfico muy útil que permite simular gran parte de las características soportadas por los dispositivos móviles con soporte J2ME. (Ver Figura 5.14.)

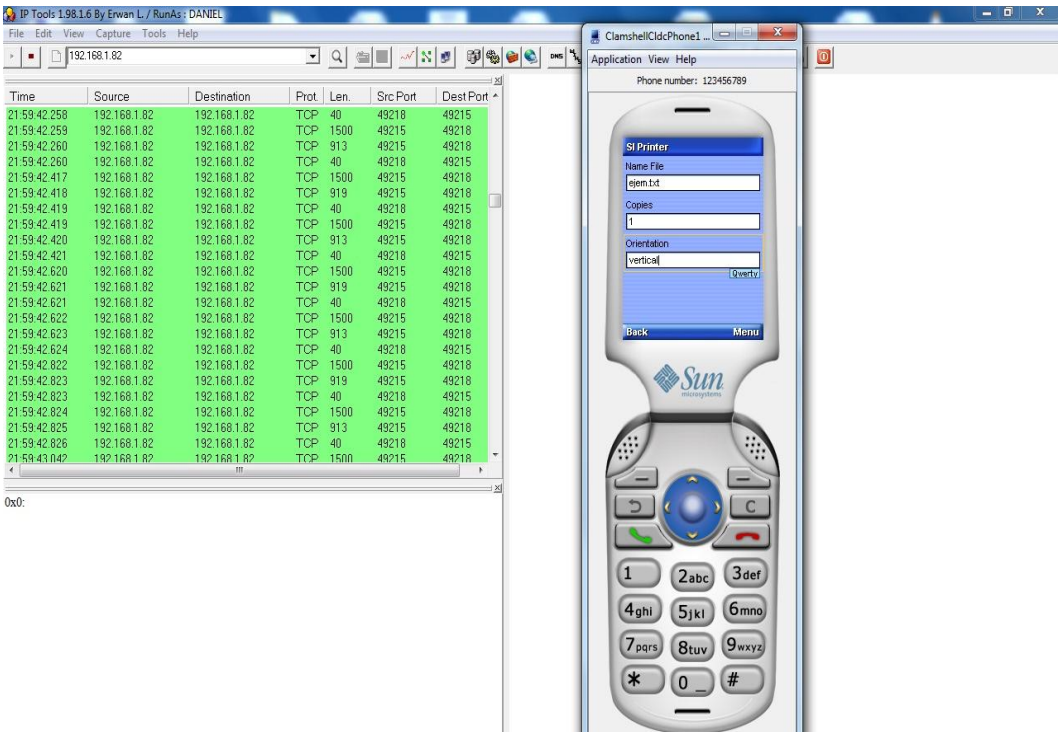


Figura 5.14 Analizador de tráfico ip sniffer.

La Tabla 5.3 resume los resultados del análisis de tráfico en ambos sentidos (dispositivo móvil – entorno ubicuo y entorno ubicuo – dispositivo móvil) generado por una sesión en

cada etapa específica, de acuerdo con los lineamentos de operación de JXME y JXTA: En la fase de establecimiento de conexión se solicita un peerid y un socketid para el móvil y por otra parte se descarga la lista de peer groups disponibles a los cueles el peer que representa al dispositivo móvil puede unirse en el momento de su conexión. El ingreso a un peer group disponible se realiza una vez que la conexión con el peer relay ha sido establecida.

En la fase de búsqueda de servicios se establecieron dos casos para analizar. Se indagó el peer relay en espera de mensajes entrantes (no hay servicios disponibles.), o bien en otro sondeo que se hace, si existen servicios disponibles. En el primer caso se tomó un tiempo de sondeo de referencia de 10 segundos y en el segundo caso se tomó un tiempo de 20 segundos.

Tabla 5.3 Análisis de tráfico.

Fase	Tráfico Uplink (Móvil a relay) [Kbytes]	Tráfico Downlink (relay a Móvil) [Kbytes]
Establecimiento de conexión.	1,526	2,011
Ingreso a peer group	0,823	0,493
Sondeo sin servicios disponibles	0,411	0,134
Sondeo con servicios disponibles	0,974	1,215

Es importante destacar que los relay JXME implementan un mecanismo para minimizar los efectos de una posible congestión. A medida que los mensajes llegan al relay se colocan en una cola de espera y van a evacuar de acuerdo con la prioridad que tienen dichos paquetes. De esta manera, los paquetes que trasmiten información más sensible a los retardos tendrán un nivel de prioridad más alto con respecto a los demás paquetes. Como resultado de las pruebas que se llevaron a cabo con la API de seguridad diseñada, en la Tabla 5.4 se muestra los resultados promedio para criptografía asíncrona. Estos tiempos corresponden al algoritmo RSA aplicado sobre un conjunto de bytes de datos aleatorios.

Tabla 5.4 Tiempo de ejecución de la criptografía asíncrona.

Tamaño/llave	Generación	Encriptación	Desencriptación
128 bits	0.2 segundos.	2 ms.	4 ms.
256 bits	1.0 segundos	5 ms.	20 ms.
512 bits	8.2 segundos	13 ms.	117 ms.
1024 bits	79.2 segundos	44 ms.	873 ms.
2048 bits	879.2 segundos	164 ms.	6752 ms.

En la Tabla 5.5 se presentan los tiempos de ejecución para la criptografía síncrona. En este cuadro, el tiempo de generación de la llave corresponde a llamar al método RandomBytesKeyGenerator( ). Los tiempos de criptografía fueron medidos usando 64

bytes de datos aleatorios. Debido a los tiempos son del orden de los milisegundos y considerando la seguridad como la principal preocupación por resolver, es recomendable usar AES con llaves de 256 bits en todos los intercambios entre dispositivos móviles y arquitectura propuesta.

Tabla 5.5 Tiempo de ejecución de la criptografía síncrona.

Algoritmo	Tamaño/llave	Generación	Encriptación	Desencriptación
DES	64 bits	1.8 ms	2.2 ms	2.8 ms
3DES	128 bits	2.1 ms	3.2 ms	3.4 ms
3DES	192 bits	2.6 ms	3.6 ms	4.0 ms
AES	128 bits	2.1 ms	4.2 ms	5.8 ms
AES	256 bits	2.9 ms	5.4 ms	7.0 ms

Finalmente, en la Tabla 5.6 se resumen los tiempos promedio de ejecución para enviar y recibir mensajes (información) de 64 bytes usando criptografía asíncrona RSA de 512 bits, criptografía síncrona AES de 256 bits. Derivado de los recursos en lo que se refiere al consumo de tiempo requerido para la codificación y decodificación en todos los intercambios de información, se concluye que es posible adaptar a los sistemas ubicuos, la etapa de seguridad con criptografía.

Tabla 5.6 Tiempos de envió y recepción de mensajes (información).

Tipo de servicio	Envío	Recepción
Mensaje codificado	40 ms	138 ms

CAPÍTULO 6
CONCLUSIONES

Resumen. En este capítulo, tomando como base los objetivos planteados, se presenta un resumen de los logros alcanzados en el presente trabajo de tesis. Además se hace una breve descripción acerca de los posibles trabajos a futuro siguiendo con la presente línea de investigación.

6.1 Conclusiones.

Uno de los principales objetivos específicos fue "implementar un entorno de cómputo ubicuo basado en sistemas operativos de código abierto con requisitos computacionales mínimos", tomando como base la infraestructura de cómputo ubicuo previamente desarrollada por [4], cosa que no fue posible, debido a lo siguiente: se presentó el problema de obsolescencia de la versión utilizada en la infraestructura de ArCU (JXTA 2.1.1). Debido a ello, numerosas clases y métodos de la versión 2.1.1, han sido relevados por nuevos y más potentes métodos, por tal motivo, se tuvo que hacer uso de la versión JXTA 2.5. Por lo anterior, fue necesario realizar una nueva programación de la arquitectura de cómputo ubicuo. De los resultados obtenidos con la serie de pruebas que se llevaron a cabo (enunciadas en el capítulo 5), se logró lo siguiente:

- Se comprobó que la inicialización del coordinador central como un peer en la infraestructura JXTA 2.5, fue hecha de manera correcta, fácil y trasparente. Además se pudo comprobar la correcta interacción de dicho peer con los demás componentes de la infraestructura.

- Se logró una correcta recepción, en cuanto a peticiones provenientes de un dispositivo móvil, además se verificó que existiera una comunicación correcta en lo que se refiere a registro y búsqueda de un servicio (peer de tipo servicio).

- En lo referente al tema de la seguridad de la información, se comprobó que la API criptográfica embebidas dentro del núcleo del coordinador central fueron capaces de desencriptar la información encriptada que llega y de igual manera encriptar la información que es enviada de regreso a su destinatario

- También se pudo observar la capacidad del motor de búsqueda del coordinador central, que permite realizar una búsqueda de los servicios previamente registrados tomando como referencia información contextual; en este caso ubicación geográfica, proporcionada por el usuario, de una forma eficaz y trasparente.

En cuanto al servicio que fue implementado (servicio de impresión) para mostrar el correcto funcionamiento de la propuesta de este trabajo de tesis, se pudo comprobar que la inicialización de los mismos, como un peer más de la infraestructura P2P, se realizó de forma correcta, de igual manera se verificó y comprobó la adecuada comunicación con el coordinador central gracias a su correcta publicación del mismo en la infraestructura JXTA. Por otro lado, se verificó que el servicio de impresión fue capaz de recibir peticiones de impresión a través de un dispositivo móvil.

En lo que al dispositivo móvil respecta, se logró, al hacer uso de un peer de tipo relay JXTA, dar soporte al mismo y poder inicializarlo como un peer de la infraestructura JXTA, y con ello conseguir una correcta comunicación con la arquitectura de computo ubicuo. Asimismo, se mostró la búsqueda eficaz de servicios publicados en el entorno ubicuo, haciendo uso de la interfaz instantánea de servicios. Por último, se comprobó la eficiencia de dicha interfaz al permitir hacer uso de los servicios localizados de una manera transparente y segura, auxiliándose de la API de criptografía; que se integran a la arquitectura para dotar de seguridad a la información que se envía o recibe.

Se concluye que es posible hacer uso de "un caso particular de ontologías semánticas (jerarquías de clases)" con el fin de poder definir semánticamente los servicios implementados en dicho entorno, es decir, hacer una conceptualización de los mismos, todo ello con el fin de satisfacer otro de los objetivos que se había planteado, el cual consintió en implementar un motor de búsqueda de servicios ubicuos considerando información de contexto geográfico. Dicho mecanismo fue dotado de un proceso de razonamiento el cual le permite, haciendo uso de la información semántica, hacer la búsqueda pertinente tomando como base los parámetros definidos (contexto geográfico, nombre, tipo; entre otros.). Lo anterior requirió de la definición de un proceso a través del uso de documentos XML y OWL, los cuales describen características propias de de los servicios que han sido implementados en dicho entorno.

Se concluye que para el intercambio de información entre los elementos que componen el entorno es posible la implementación de un mecanismo, el cual permite el uso de sockets, que nos otorga una gran flexibilidad en lo que se refiere al intercambio de información en dicho entorno ubicuo. Para lograr dotar de un mecanismo de autentificación en dicha arquitectura, se ha desarrollado una clase basada en un elemento del protocolo JXTA llamado Java.util.logging, incluida en la versión 2.5, con lo que se consigue satisfacer otro de los objetivos que se han planteado resolver en este trabajo de tesis.

Para dotar de seguridad a dicha arquitectura, en lo que se refiere al intercambio de información de carácter confidencial en el entorno ubicuo haciendo uso de medios inalámbricos, se desarrolló una API de seguridad basada en una librería criptográfica (Bouncy Castle), por lo que se pudo cumplir otro de los problemas importantes que se pretendía resolver en este trabajo de tesis.

Por otra parte, se concluye que un elemento importante y necesario para la comunicación con el entorno por parte de los usuarios, es el dispositivo móvil, el cual cuenta con recursos altamente reducidos en comparación con una estación de trabajo y que es necesario hacer uso de una versión JXTA para dispositivos móviles, la cual es denominada JXME.

Se concluye que es posible, haciendo uso de la versión para móviles Java Micro Edition, conseguir con éxito el desarrollo de la interfaz de usuario por medio de la cual se logra interactuar con el entorno ubicuo, consiguiendo con ello una interoperabilidad espontánea

adecuada para el uso de cualquier dispositivo móvil que soporte Java en dicho entorno ubicuo.

Se cumplió con el objetivo de desarrollar la interfaz de usuario para que los dispositivos móviles contaran con un mecanismo a través del cual se consigue recabar información de tipo contextual (en este caso ubicación geográfica), con el fin de que dicha información apoye de forma directa al motor de búsqueda en la localización de un servicio de forma exitosa.

Se consiguió de forma satisfactoria la implementación de todo el conjunto de elementos que conforman el presente trabajo de tesis, como se mostró en el capítulo 5, en la sección 5.1, logrando apreciar el cumplimiento de cada uno de los objetivos que previamente se plantearon, así como resolver algunos otros problemas que surgieron en el trascurso de la elaboración de esta propuesta.

Finalmente, basándonos en los resultados cuantitativos obtenidos por las pruebas que se mostraron en la sección 5.2, se puede concluir lo siguiente: en comparación con la arquitectura de cómputo ubicuo propuesta por Pineda, esta propuesta ofrece una arquitectura más robusta, en cuanto a seguridad, integridad, eficiencia y desempeño. Un aspecto a destacar de la nueva arquitectura en contraposición con otras del mismo tipo, es el uso de recursos de hardware mínimos y herramientas de uso libre para la integración del entorno de cómputo ubicuo; que a diferencia de otras propuestas no ofrecen una forma de homogeneizar los elementos necesarios para lograr una adecuada interacción de los elementos del entorno ubicuo. Cabe resaltar que con este trabajo de tesis se logra un nivel de interoperabilidad espontánea en lo que se refiere a los dispositivos móviles hacia el entorno ubicuo, de tal manera que no se necesitan dispositivos móviles especializados para dicha propuesta.

6.2 Trabajo a futuro.

De este trabajo de tesis se desprenden diversas líneas de investigación para lograr enriquecer las propuestas que tengan que ver con la creación de entornos ubicuo de una manera fiable, trasparente y segura, todo ello sin realizar una gran inversión:

- La creación de un mecanismo para lograr facilitar la implementación de manera amigable de nuevos servicios, de tal forma que para dicha tarea no se requiera de un experto, el cual se encargue de programar dicho servicio para ser instalado en el ambiente, para que cualquier usuario con un mínimo de conocimientos computacionales, haciendo uso de una interfaz gráfica pudiera implantar servicios que fueran de su agrado.

- Un mecanismo de contexto geográfico exacto e intuitivo, para la localización de servicios. Tomando en cuenta que el dispositivo móvil tuviera integrado un dispositivo GPS, se podría manipular la información generada por él mismo y así hacer una búsqueda más exacta e intuitiva de los servicios. La cuestión es que actualmente no todos los usuarios cuentan con un dispositivo móvil que integre esa funcionalidad. Por otro lado, un enfoque que podría ayudar a mejorar dicho mecanismo de búsqueda sería el uso de un esquema de recursos compartidos, es decir, definir un algoritmo que se encargue de localizar un dispositivo auxiliándose del uso de dispositivos vecinos, como podría ser un GPS, para obtener dicha localización.

- Para dotar de seguridad al entono ubicuo propuesto, se han hecho uso de librerías criptográficas de código abierto, pero no necesariamente se tienen que hacer uso de ellas, se podría desarrollar un algoritmo propio específicamente para este tipo de entornos, el cual permitiera la codificación-decodificación de la información, autentificación, y otras cuestiones de seguridad en cuanto al uso de de servicios se refiere.

REFERENCIAS BIBLIOGRÁFICAS

Referencias Bibliográficas.

[1] Weiser, M. “The Computer for the 21st Century”. Scientific American, Vol. 265, No. 3, pp. 94-104, September 1991.

[2] Satyanarayanan, M. “A Catalyst for Mobile and Ubiquitous Computing”. IEEE Pervasive Computing, Vol. 1, No. 1, January-March 2002.

[3] Satyanarayanan, M. "Pervasive Computing: Vision and Challenges", IEEE Personal Communications, pp. 10-17, August 2001.

[4] Pineda Briseño, Anabel. “Arquitectura de contro ubicuo”, Tesis de Maestría, Centro de Investigación en Computación, Abril del 2006.

[5] Weiser, M. “The future of Ubiquitous Computing on Campus”. Communications of the ACM, Vol. 41, No. 1, pp. 41-42, January 1998.

[6] Kindberg, T. and Fox, A. “System Software for Ubiquitous Computing”, Stanford university, March 2002.

[7] G. Couloris, J. Dollimore, and T. Kindberg. “Distributed Sistems Concepts and Designs.”, Addison-Wesley, July 2008.

[8] Embedded Systems. www.embedded.com.

[9] Weiser, M. “Some Computer Issues in Ubiquitous Computing”. Communications of the ACM, pp. 75-84, May 1993.

[10] Ghen, G., Kotz, D. “Context Aggregation and Dissemination in Ubiquitous Systems”, IEEE Workshop on Mobile Computing Systems and Applications, pp. 105-114, June 2002.

[11] Xavier, A. “Ambient Intelligence at Home: Facts and Future” NOVATICA. “Número especial acerca de la computación ubicua.” October 2006, Vol. 8, No.4, pp. 30-75, October 2006.

[12] Feilong Tang, Minyi Guo, Minglu Li. “AN ADAPTIVE CONTEXT-AWARE”, School of Information Science Korean Bible University, pp. 1-14, January 2008.

[13] John A. “CONTEXT AWARENESS IN A MOBILE DEVICE: ONTOLOGIES VERSUS”, CiteSeerx, pp. 1-8, May 2008.

[14] Ignacio Vázquez. “WALLIP: Una plataforma de servicios con percepción contextual basada en localización”, Ubiquitous Computing and Ambient Intelligence, pp. 1-7, September 2005.

[15] Bravo, J., Hervás, R. “Servicios por identificación en el aula ubicua”, Servicio de Publicaciones - Universidad de Extremadura, pp. 1-8, May 2005.

[16] Frank S. “Security for Ubiquitous Computing”, John Wiley & Sons, March 2002.

[17] Dey, A. “Understanding and Using Context.”, Personal and Ubiquitous Computing, pp. 130-160, December 2001.

[18] Jennifer S. Beaudin Stephen S. Intille. “MicroLearning on a Mobile Device”, Citeseerx, pp. 1-2, August 2009.

[19] Beigl M. “Objects., Smart-Its: An Embedded Platform for Smart” , Citeseerx, pp. 1-4, January 2007.

[20] Kontaxakis M.”PUMAS: Un Framework que adapta la informacion en Ambientes ubicuos”,Proceeding de 20th annual ACM symposium on applied computing, pp. 1003-1008, May 2008.

[21] Milton Ausecha Penagos,Javier Imbús Guzmán “Arquitectura para facturación y pago por proximidad de servicios ubicuos” Communications of the ACM, Vol. 46, No. 2, pp. 165-184, February 2003.

[22] Anderson. G and Steinberg. E. “Hospital readmis-sions in the Medicare population.”, IEEE Pervasive Computing, Vol. 1, No. 1., pp. 270-300, June 2002.

[23] Awtrey, Dan. “The 1-Wire Weather Station”, Dallas Semiconductor, April 1998.

[24] MAXIM. “High Speed Microcontroller User's Guide.”, Technical Report, pp. 60-90, November 2006.

[25] Maxim. Dallas Semiconductor. http://www.maxim-ic.com/.

[26] Andrew. S. “Sistemas operativos distribuidos”, Prentice-Hall, May 2000.

[27] Silberschatz, Abraham. “Operating system concepts”, Addison-Wesley, July 1998.

[28] Comer, Douglas. “Operating system design”, Prentice-Hall, March 2004.

[29] Jamsa, Kris.“DOS 5 : Manual de referencia”, McGraw-Hill, February 1991.

[30] Jamsa, Kris.“DOS : power user's power guide”, McGraw-Hill, June 1998.

[31] Microsoft DOS and command prompt. http://www.computerhope.com/

[32] Ogletre, T.“Windows xp”, SAMS, July 2002.

[33] Sistema operativo windows Vista. http://win-vista.es/windows-vista/.

[34] Daniloff, C. “Mac OS 8”, Paraninfo, January 1998.

[35] Kirk McElhearn. “Mastering Mac OS X v10.4 Tiger”, Sybex, May 2005.

[36] Historia y evolución del Sistema Operativo Mac OS. http://www.maestrosdelweb.com.

[37] Gary J.“Operating systems : a modern perspective”, Addison Wesley, August 2002.

[38] Lorin, Harold. “Operating systems”, Addison-Wesley, January 1981.

[39] Coffin, Stephen.(1990) “UNIX : the complete reference”, McGraw-Hill, September 1990.

[40] Love, Paul. “Beginning Unix”), Wiley, June 2005.

[41] “The Unix System. http://www.unix.org/.

[42] Facundo H. “Linux . Books Worldwide”, MP, February 2000.

[43] Diaz, juan Luis. “Criptografia, Criptologia y Criptoanalisis “, IPN CIC, July 2008.

[44] Bruce, Schneier. “Applie Cryptography”, phil Sutherland, April 1995.

[45] Menchaca, R., Favela, J. “Criptografia de llave pública”, Septiembre del 2003.

[46] Current Public-Key Criptographic System. www.certicom.

[47] Miguel, Gustavo. (1997) “Seguridad Informtaica”, MP, July 1997.

[48] Lars, Klander. (2007) “A prueba de Hackers“, Anaya multimedia, February 2007.

[49] J2ME Wireless Tool Kit”. http://java.sun.com/products/j2mewtoolkit/.

[50] Weadock, Glenn. “Bullet proofing Windows 95”, MacGraw-Hill, February 1997.

[51] Stinson, Craig. “Running Windows”, Microsoft Press, May 1990.

[52] JXTA Programmers Guide Development Project. https://jxta.dev.java.net/.

[53] Profile Mobil Information Device. http://www.sun.com/software/.

[54] Platform, JXTA J2SE. http://download.jxta.org/.

[55] Parser, Xerces Java. http://xml.apache.org/xerces-j/.

[56] Java 2 Platform, Standard Edition. http://java.sun.com/downloads/.

[57] Rudolph, Larry. “Project Oxygen: Pervasive,Human-Centric Computing.”, SpringerLink, Vol. 2068, pp. 1-12, January 2001.

[58] Amandi A. “Monográfico de Inteligencia Artificial y sistema Multiagente.”, Revista iberoamericana de Inteligencia Artificial, Vol. 5, pp. 33-35, May 1998.

[59] Perkins, C. (2002) “IP Mobility Support”, IBM press, August 2002.

[60] Motorola Inc. “Microrocesador 6802 datasheet”, Motorola September 1991, http://www.datasheetcatalog.net/es/datasheets_pdf/M/C/6/8/MC6802.shtml.

[61] MAXIM. “DS80C400 Network Microcontroller.” Data Sheet, July 2005.

[62] Friedewald, M. and Da Costa, O. “Ambient intelligence in everyday life”, Scientific American, Vol. 281, pp. 52-55, May 2006.

[63] Loreto Susperregi Inaki Maurtua, Carlos Tubío. “Una arquitectura multiagente para un Laboratorio de Intelegencia Ambiental”, IEEE Intelligent Systems, Vol.46, pp. 1-10, March-April 2002.

[64] Preece, J. “Human-Computer Interaction”, Addison-Wesley, March 1994.

[65] Hermann, R., Husemann, D., Moser, M., Nidd, M. “DEAPspace – Transient ad hoc networking of pervasive devices”, The International Journal of Computer and Telecommunications Networking, Vol. 46, No. 2, pp. 411-428, June 2001.

[66] Fielding, R. “Hypertext Transfer Protocol /XML”, Springer-Verlag, pp. 60-110, November 1999.

[67] Dertouzos, Michael L. “The human-centric approach”, Communications of the ACM, Vol.1, pp. 90-204, April 2001.

APÉNDICE A

Hardware y Sistema Operativo.

Los parámetros que se tomaron como base para la selección del hardware más adecuado, fueron seleccionados con base en requerimientos técnicas mínimos necesarias, para la incorporación de cada una de las herramientas necesarias para la implementación de la infraestructura JXTA, así como cada uno de los elementos que conforman el entorno. A continuación se muestra una lista con los requisitos mínimos necesarios.

Requisitos mínimos.

- Procesador x86 de 300 MHz.
- 1 GB de RAM.
- 4 GB de espacio en disco (para la instalación completa y el espacio de intercambio).
- Tarjeta gráfica VGA de 640x480 de resolución.
- Unidad de CD-ROM.
- Tarjeta de Red.

Aunque la configuración de hardware mínima antes mencionada es suficiente, para dar soporte y permitir la implementación del entorno ubicuo, se realizaron pruebas con diversas configuraciones de hardware, es decir manipulando parámetros como son: memoria RAM, tamaño de disco duro, velocidad de procesamiento. Dichas pruebas permitieron establecer una configuración mínima recomendada para proporcionar un adecuado balance entre desempeño, estabilidad, respetando el concepto minimalista en cuanto a hardware se refiere, establecido por esta propuesta de trabajo. A continuación se enlistan dichos requerimientos.

Recomendado.

- Procesador x86 de 700 MHz.
- 4 GB de RAM.

- 8 GB de espacio en disco duro.
- Tarjeta gráfica capaz de una resolución de 1024x768.
- Tarjeta de sonido.
- Tarjeta de red.

Para la selección del sistema operativo, el cual como se ha establecido con anterioridad debe ser software libre y código abierto, se probaron diversas distribuciones de Linux mínimas, tomando como parámetro la configuración recomendada anteriormente y además el soporte oportuno a los elementos del entorno ubicuo, en el Apéndice C puede apreciarse una tabla, la cual incluye un resumen de las características que presentaron dichas distribuciones. La distribución que fue seleccionada es la denominada gOS debido a las siguientes características.

- gOS es una distribución Gnu / Linux basada en Ubuntu 8.04 la cual incluye una gran cantidad de aplicaciones y paquetes.
- Consta de una instalación sencilla, una estabilidad aceptada, y rapidez al momento de ejecutar aplicaciones y tareas en el sistema.
- gOS con todo su esplendor visual consume muy poca memoria, menos de 200 megabytes de RAM.
- Debido a que es tan liviano, es posible ejecutarlo incluso en ordenadores muy modestos, pero con todo el diseño y la elegancia que sea posible imaginar.
- gOS proporciona el suficiente soporte para la implementación de cada uno de los elementos que integran el entorno ubicuo.

Instalación de gOS.

Una vez que se ha descargado un archivo de tipo ISO desde el siguiente sitio:
http://www.thinkgos.com/company/index.html,
Se deben seguir los siguientes pasos:

1. Grabar el archivo ISO en un CD / DVD.
   El archivo descargado es un "ISO", que es necesario grabarlo en un CD / DVD. Buscar una opción especial de software de grabación de CD-DVD que permita crear un CD / DVD desde un archivo ISO.

2. Introducir el CD / DVD en el interior de la unidad lectora de CD.
   Tras superar con éxito la grabación de un CD, mantener el CD de gOS 3.1 dentro de la unidad lectora.

3. Reiniciar la computadora.
   Con el CD dentro de gOS 3.1, debe reiniciar el equipo para ver una nueva pantalla de menú gOS.

4. Seleccione "Iniciar o instalar gOS".
   Esperar pacientemente a que gOS 3.1 cargue desde el CD / DVD.

5. Hacer doble clic en el icono "Instalar".

6. Seguir las instrucciones para terminar la instalación del sistema operativo.
   Cuando haya terminado, tendrá que reiniciar el sistema.

APÉNDICE B

Bouncy Castle.

Bouncy Castle es una colección de APIs utilizadas en criptografía. Incluye tanto la API para lenguaje Java como para C#. El paquete Bouncy Castle Crypto es una implementación hecha en Java sobre los principales algoritmos criptográficos, la cual ha sido desarrollada por the Legion of the Bouncy Castle.

El paquete ha sido organizado de tal manera que se incluye un API ligera la cual proporciona un amplio panorama de implementación, como pueden ser en un ambiente de tipo J2ME, JDK 1.6, ó cualquier otro más que cuente con la infraestructura adicional para conformar los algoritmos del framework JCE, todo ello sin la necesidad de hacer ningún cambio. Este conjunto de librerías son distribuidas bajo licencia de software libre y código abierto. Entre las características más importantes de dicho paquete sobresalen las siguientes:

- API de criptografía ligera.
- Un proveedor para la Java Cryptography Extension y la arquitectura Java Cryptography.
- Una librería para leer y escribir objetos codificados ASN.1.
- Generador para certificados versión 1 y 3.X.509.
- Generador de atributos de certificados versión 2.x.509.

Para realizar la descarga así como obtener mayor información acerca de estas librerías criptográficas se puede visitar el siguiente enlace:

http://www.bouncycastle.org/mirrors.html.

APÉNDICE C

A continuación se muestra un cuadro comparativo, entre diversas versiones de Linux tomando como figura demerito las diversas configuraciones (mínimas y recomendadas) en cuanto a requisitos de hardware.

Nombre	Requerimientos		Inst. Live	USB	Disco Duro
	Mínimos	Recomendado			
Ubuntu 8.04	*Procesador 500 MHz* *256 MB de RAM* *4 Gb de Disco duro*	*Procesador 1.5 GHz* *1Gb de Memoria RAM* *8 Gb de disco duro* *Tarjeta de video GeForce 64 a 128 Mb*	SI	NO	SI
Ubuntu Server Edición 8.04.1	*Procesador de 200 MHz* *64 MB de RAM* *800 MB disco duro*	*Procesador de 500 MHz* *128 MB de RAM* *1Gb disco duro*	NO	NO	SI
Zenwalk Linux 5.2	*Procesador 486* *64 MB de RAM* *600 MB de disco duro*	*Procesador tipo Pentium II.* *128 MB de memoria RAM* *2 GB disco duro.*	SI	NO	SI
Vector Linux Light 5.9	*Procesador Pentium 233MHz* *128MB de RAM* *2.5 GB de disco duro*	*Procesador Pentium II a 300 MHz* *256MB de RAM* *6 GB de disco duro* *Tarejta de video soportada por XFree86*	SI	NO	SI
slax-6.0.7	*Procesador 486, Pentium o AMD* *36 MB de RAM*	*Procesador Pentium II 450 MHZ* *36 MB de RAM* *400 disco duro*	SI	SI	SI

Puppy Linux 1.0.2	*Procesador Pentium 166Mhz* *64 MB de RAM* *No requiere disco duro*	*Procesador Pentium 200Mhz* *128 MB de RAM* *No requiere disco duro*	SI	SI	SI
Debian 4.0	*Procesador 500 MHz* *32 MB de RAM* *500 MB de disco duro*	*Procesador 1Ghz* *194 MB de RAM* *3GB de disco duro*	SI	SI	SI
CentOS 5.2	*Procesador Intel Pentium o AMD 400Mhz* *64 MB de RAM* *1024 MB de disco duro*	*Procesador Intel Pentium o AMD 800Mhz* *128 MB de RAM* *2 GB de disco duro*	SI	SI	SI
Fedora 9	*Procesador 233 MHz* *128 Mb de RAM* *175 MB de disco duro*	*Procesador 400 MHz* *256 Mb de RAM* *9 GB de disco duro*	SI	SI	SI
Slackware	*Procesador 166 Mhz* *32 MB de RAM* *1 GB de disco duro*	*Procesador 500 Mhz* *128 MB de RAM* *2 GB de disco duro*	SI	NO	SI
Linux Kanotix	*Procesador Pentium i586* *192 MB de RAM* *No requiere disco duro*	*Procesador Pentium i586* *256 MB de RAM* *No requiere disco duro*	SI	NO	NO
NimbleX 2008	*Procesador 166 MHz* *128 Mb de RAM* *400 MB de disco duro*	*Procesador 450 MHz* *256 Mb de RAM* *1GB de disco duro*	SI	SI	SI
PCLinuxOS	*Procesador 400 MHz* *128 Mb de RAM* *1.5 GB de disco duro*	*Procesador 500 MHz* *256 Mb de RAM* *10 GB de disco duro*	SI	SI	SI

KNOPPIX 5.1	*Procesador I486 MHz* *128 Mb de RAM* *1.5 GB de disco duro*	*Procesador 500 MHz* *256 Mb de RAM* *10 GB de disco duro*	SI	NO	SI

APÉNDICE D

Plataforma JXTA.

En el capítulo 3 sección 3.6 se habló con más profundidad acerca de la tecnología JXTA. En este apéndice solo se mencionarán las características más sobresalientes de la versión 2.5, la cual fue usada para la implementación de la presente propuesta de tesis:

- Implementación de transporte TCP para Java NIO.
- Mejor administración de hilos para reducir consumo de recursos
- Reprogramación del código encargado de la implementación de comunicación de los peers: relay y rendevouz, con el fin de hacer la conexión más rápida y eficiente.
- El mecanismo de seguridad Log4J fue remplazado por Java.util.logging
- Inclusión de una nueva clase NetworkManager, la cual se encarga de todo lo relacionado con la configuración e inicio de JXTA.
- Mejoras en lo que se refiere a rendimiento y confiabilidad para los JXTASockets.
- Se reduce el tráfico de red en lo que se refiere a tuberías de propagación.

APÉNDICE E

Propuestas	***Uso eficiente de espacios(Información Contextual geográfica)***	***Interoperabilidad espontánea***	***Seguridad***	***Conexión***	***Requisitos (Hardware/Software)***
Arquitectura de Cómputo ubicuo (Propuesta de esta tesis)	Cuenta con un mecanismo basado en el contexto geográfico por medio del cual se recomienda el uso de un determinado servicio existente.	Permite componentes (Dispositivos móviles) que puedan interactuar espontáneamente en ambientes cambiantes	Hace el uso de un mecanismo de autentificación de usuarios, así como de algoritmos de encriptación que permitan codificar la información que se intercambia a través de infraestructuras inalámbricas	A través de medios de trasmisión inalámbrica (redes públicas)permite la constante entrada y salida de componentes	La arquitectura de cómputo ubicuo está basada en sistemas operativos de código abierto con requisitos computacionales mínimos.
AN ADAPTIVE CONTEXT-AWARE TRANSACTION MODEL FOR MOBILE AND UBIQUITOUS COMPUTING	Atreves de el uso de mallas de redes inalámbricas Wi-Fi, crea un modelo adaptativo para los constantes cambios en el entorno.	Los dispositivos móviles son interconectados a través de redes inalámbricas haciendo uso de el estándar IEEE 802.11n, lo cual deja fuera dispositivos que no operen en dicho estándar.	No maneja algún tipo de mecanismo de seguridad(encriptación-Autentificación)	Los dispositivos en dicha propuesta, se intercomunican a través de una infraestructura de red inalámbrica creando mallas.	Los requisitos computacionales requeridos para la implementación de esta propuesta están limitados a una plataforma específica.
MicroLearning on a Mobile Device	No cuenta con ningún mecanismo que envuelve la información contextual.	Permite la interoperabilidad con cualquier dispositivo móvil que cuenta con soporte java	No maneja algún tipo de mecanismo de seguridad(encriptación-Autentificación)	Hace uso de redes inalámbricas de tipo ad-hoc	Los requisitos computacionales requeridos para la implementación de esta propuesta son mínimos y exigen el soporte java.
An Architecture Concept for Ubiquitous Computing Aware Wearable Computers	No cuenta con ningún mecanismo que envuelve la información contextual.	En esta propuesta los componentes (módulos) puedan interactuar espontáneamente entre sí.	No maneja algún tipo de mecanismo de seguridad(encriptación-Autentificación)	Los módulos en dicha propuesta hacen uso de un protocolo de comunicación propio diseñado específicamente para ello.	Los requisitos computacionales requeridos para la implementación de esta propuesta están limitados a una plataforma específica.

WALLIP	Cuenta con una plataforma de localización la cual es basada en el uso de redes de comunicaciones celular.	Solo hace uso de redes celulares para lograr la interactividad por lo cual es difícil lograr una interoperabilidad correcta.	Solo maneja un mecanismo muy limitado se definen perfiles de acceso.	Hace uso de redes celulares para conseguir la comunicación con el ente que se encarga de administrar los recursos	Los requisitos computacionales requeridos para la implementación de esta propuesta están limitados a una plataforma específica.
Servicios por identificación en el aula ubicua	A través de pequeños dispositivos embebidos alrededor de una cierta área se consigue la identificación del tipo de usuario sin tomar en cuenta su ubicación en la misma.	No se ofrece una interoperabilidad, debido a que la aplicación de esta propuesta es muy especifica.	Hace uso de de dispositivos RFID, lo cual quiere decir que implícitamente se usa un tipo de encriptación muy limitada en cuanto a seguridad para la trasferencia de información.	Hace uso de protocolos propios de la tecnología de radiofrecuencia para conseguir la comunicación entre componentes de la arquitectura.	Los requisitos computacionales requeridos para la implementación de esta propuesta están limitados a una plataforma específica.
Smart-Its: An Embedded Platform for Smart Objects.	Mediante el uso de sensores, se captura información contextual entre ella localización y estado.	No provee diversos mecanismos de comunicación solo hace uso de tecnología RFID por lo que la interoperabilidad no se logra.	Debido al uso de tecnología RFID se maneja un esquema encriptación de información limitado en cuanto a la información que se trasmite.	Hace uso tecnología RFID para lograr la comunicación entre los componentes.	Los requisitos computacionales requeridos para la implementación de esta propuesta están limitados a una plataforma específica.
PUMAS (Peer Ubiquitous Multi-Agent System).	Atreves de sistemas de información Web inteligentes, se hace la administración de la información contextual (localización, preferencias, entre otros.)	No existe una interoperabilidad espontanea.	No maneja algún tipo de mecanismo de seguridad(encriptación), solo de autentificación	Hace uso de un servidor web para lograr la comunicación entre los componentes.	Los requisitos computacionales requeridos para la implementación de esta propuesta están limitados a una plataforma específica.
C-MONITOR	No cuenta con ningún mecanismo que envuelve la información contextual.	No existe una interoperabilidad espontanea ya que se requiere de una red celular móvil.	No maneja algún tipo de mecanismo de seguridad (encriptación), solo de autentificación.	Para obtener la comunicación con el ente centralizado de esta propuesta se hace uso de una red de celular móvil.	Los requisitos computacionales requeridos para la implementación de esta propuesta están limitados a una plataforma específica.

Arquitectura para facturación y pago por proximidad de servicios ubicuos	No cuenta con ningún mecanismo que envuelve la información contextual.	El sistema contempla maneja el concepto de que los usuarios son altamente dinámicos e impredecibles.	Cuenta con un mecanismo de seguridad para evitar fraudes, como : robo y falsificación	El sistema no asume ninguna configuración específica de red, dispositivo o usuario	Los requisitos computacionales requeridos para la implementación de esta propuesta están limitados a una plataforma específica.
ArCU	No cuenta con un mecanismo basado en el contexto geográfico que permita recomendar el uso de un determinado servicio, acorde a la ubicación del individuo.		No existe un mecanismo que permita controlar y restringir el uso de los servicios, además no hay una forma de garantizar la privacidad de la información	Hace uso de redes publicas	Los requisitos computacionales requeridos para la implementación de esta propuesta están limitados a una plataforma específica.