Centro de Investigación y de Estudios
Avanzados
del Instituto Politécnico Nacional

Departamento de Ingeniería Eléctrica

Sección de Computación

IRIS: Una infraestructura de software para el desarrollo de escenarios de colaboración digital espontánea sobre dispositivos móviles IEEE 802.11b

Tesis que presenta

Silvana Bravo Hernández

para obtener el Grado de

Maestra en Ciencias

en la Especialidad de

Ingeniería Eléctrica

Director de la Tesis

Dr. José Oscar Olmedo Aguirre

México, D.F. **Septiembre 2004**

II

Agradecimientos

Dedico este trabajo como una pequeña forma de agradecimiento a la persona que tuvo la gran visión de concebir la idea que origino este trabajo, por que siempre esta cuando lo necesito, por que siempre me ha apoyado y me impulsa con su gran espíritu emprendedor. Gracias Alfonso. A mi madre por que me dio las armas para enfrentarme a la vida y me enseño con su ejemplo el gran amor incondicional y sin reservas que una madre puede prodigar a sus hijos. A mi padre por que me enseño a enfrentar la vida con optimismo y sin restricciones. A Yola, Raquel, Nacho, Dani y Hugo por que comparten mis tristezas y mis alegrías y me acompañan siempre en mis triunfos. A mis sobrinos Yolita, Dani e Itiel por que son mi esperanza en el futuro. A Gina, Alfonso y Polo por que son parte de mi familia y me han demostrado su afecto dandome su apoyo siempre.

De manera especial agradezco al Dr. José Oscar Olmedo Aguirre por haber creído, apoyado y enriquecido este proyecto.

De igual forma agradezco al Dr. Guillermo Benito Morales Luna y al Dr Francisco Rodríguez Henríquez por el tiempo invertido en la revisión de este documento y por sus valiosos comentarios para mejorar el mismo.

Finalmente mi agradecimiento al Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional.

Resumen

Las aplicaciones tradicionales de colaboración digital están limitadas a escenarios que cuentan con una infraestructura de red previamente establecida sobre la que se realiza la colaboración.

Con el avance en la tecnología de integración a gran escala, la microelectrónica y las telecomunicaciones fue posible la construcción de una gran variedad de dispositivos móviles proliferando los dispositivos de propósito específicos como las agendas electrónicas. Sin embargo, debido al poder computacional con que cuentan y sus capacidades de comunicación a través de enlaces inalámbricos, es posible utilizar estos dispositivos como herramientas de colaboración digital en ambientes móviles. No obstante para hacer realidad estos escenarios que llamaremos escenarios de *colaboración digital espontánea* se necesita construir una capa intermedia de software que: gestione la administración y configuración de la red para habilitar la comunicación entre los participantes (la cual en los anteriores escenarios de colaboración era definida a priori por un experto). Así como el manejo del contexto de colaboración que involucra el conocimiento de la llegada y salida de los dispositivos participantes que colaboran en la tarea.

En esta tesis se discute el diseño e implementación de la capa intermedia de software **IRIS** que sirve como base para la construcción de aplicaciones de colaboración digital espontánea y que trabaja sobre dispositivos móviles heterogéneos con tecnología inalámbrica IEEE 802.11b. **IRIS** está formada por un conjunto de protocolos y mecanismos de colaboración para ambientes móviles que efectúan: la detección, asociación o creación de un grupo de trabajo, la entrada y salida de los participantes y la difusión de las acciones de cada participante al grupo de colaboración. Finalmente, para mostrar la funcionalidad de la infraestructura de software **IRIS**, se presenta la implementación de un pizarrón compartido para el intercambio de información en grupos de trabajo.

Abstract

Traditional applications of digital collaboration are restricted to scenarios that assume previously established network infrastructure under which the collaboration is carried out.

With the current advances in VLSI (Very large scale of integration), microelectronics and telecommunications, the construction of a wide variety of mobile devices has been possible. The most popular are cell phone and PDA's (Portable Digital Assistants). These mobile devices due to their: proliferation, computational power and their recent wireless communication capabilities are suitable tools for digital collaboration in a mobile environment. However, to achieve these scenarios, that we call "spontaneous digital collaboration scenarios", we design and implement a middleware that handles: the network configuration and administration, the collaboration context and the mobility of nodes in the spontaneous networks. All these mechanisms were created in order to allow the easy set up and the interoperation between heterogeneous devices. Therefore, the work group users don't have to worry about nodes configuration or application set up.

In this thesis we discussed the design and the implementation of IRIS, a middleware that is the base for the construction of collaborative applications that will run on spontaneous networks of mobile devices equipped with wireless technology IEEE 802.11b. IRIS is composed by a suite of protocols and mechanisms that handle: the detection, association or creation of a work group; the entrance and exit of the participants and the diffusion of the actions of each participant to the collaboration group.

Finally, to show the functionality of the IRIS middleware, we present the implementation of a shared blackboard that has been to exchange of information in work groups.

Índice General

Índice de Figuras

Índice de Tablas

1

Introducción

La colaboración en grupos de personas que trabajan persiguiendo metas comunes se caracteriza por el gran intercambio de información que se da entre los participantes. Cuando no existía ninguna tecnología de información, los escenarios de colaboración se limitaban a una reunión en donde se realizaba un intercambio de ideas a través del lenguaje hablado y esta información se almacenaba en la mente de los participantes. Mark Weiser declaró que quizás la escritura fue la primera tecnología de información y aseguró que "La capacidad de representar de una manera simbólica el lenguaje hablado permitió que éste se almacenara para que perdurara liberando a la información de los límites de la memoria individual" [1]. Gracias a la invención de la escritura, los escenarios de colaboración cambiaron y se utilizó como herramientas la pluma y el papel para almacenar y modificar la información que se generaba. Luego, con la aparición de los dispositivos que almacenan información en formato digital nuevamente cambiaron los límites del espacio de almacenamiento y más aún se hizo posible el procesamiento de la información de manera automática dando paso a los escenarios de colaboración digital.

1.1. Colaboración digital

La *colaboración digital* como se muestra en la figura 1.1 es un conjunto de personas que a través de dispositivos digitales conectados en red cooperan para la realización de una meta común.

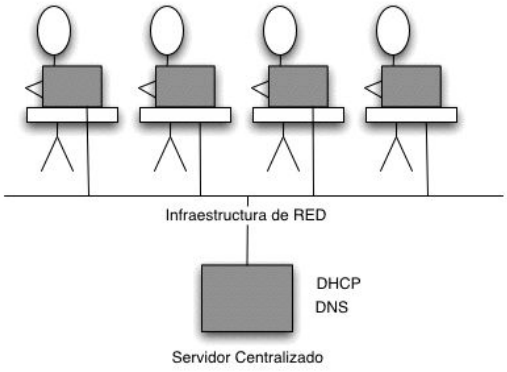

Figura 1.1: Colaboración digital

Muchas aplicaciones para colaboración digital han sido desarrolladas para ayudar en algún escenario de colaboración por ejemplo [2]:

- Sistemas de conferencias computarizadas
- Sistemas de charla
- Sistemas de pizarrón compartido
- Sistemas de audioconferencia
- Sistemas de vídeo conferencia
- Sistemas de administración de flujo de trabajo
- Sistemas de coautoría

Una de las clasificaciones de las aplicaciones de *colaboración digital* más conocida y empleada es la de espacio-temporal que divide las aplicaciones por el lugar y el momento en que ocurre la colaboración [2]:

- En un mismo sitio (local) y a un mismo tiempo (síncrono)
- En un mismo sitio (local) y a diferente tiempo (asíncrono)
- En diferentes sitios (remoto) y a un mismo tiempo (síncrono)

- En diferentes sitios (remoto) y a diferente tiempo (asíncrono)

Sin embargo en está clasificación se supone la existencia de una red fija en los sitios donde se lleva a cabo la colaboración ya sea en forma remota o local. Las aplicaciones de *colaboración digital* desarrolladas para los tradicionales ambientes de computadoras de escritorio, están limitadas a los escenarios que establecen a priori la infraestructura sobre la cual se realiza la colaboración, es decir, estas aplicaciones se desarrollan suponiendo que existe una red local a la cual todos los participantes tienen acceso de manera permanente (por ejemplo la red interna de una empresa o de una universidad), o vía internet para lo que también es necesario contar con una infraestructura la cual proporcione el acceso. Sin embargo, debido a los avances en la tecnología de integración a gran escala, la microelectrónica y las telecomunicaciones que han permitido la construcción y la proliferación de una gran variedad de dispositivos móviles equipados con tecnología inalámbrica, se vislumbra que los usuarios se desplazarán de los tradicionales ambientes de computadoras de escritorio conectadas por redes fijas, hacia ambientes con usuarios nómadas y dispositivos móviles que colaboran formando parte de redes inalámbricas con nodos que llegan a estar disponibles o que desaparecen en un cierto período como se muestra en la figura 1.2.

1.2. Colaboración digital espontánea

En la actualidad los dispositivos móviles son útiles almacenando y procesando información de manera individual, o si acaso, transfiriendo la información a otro dispositivo. Pero en los casos en donde se necesita colaborar en escenarios en donde no se cuenta con una infraestructura de red estos dispositivos no son explotados como herramientas de colaboración.

La proliferación de estos dispositivos debido a su éxito en el mercado ha dado lugar a escenarios en donde surge la necesidad de compartir información entre ellos de manera espontánea. A estos escenarios los llamamos escenarios de *colaboración digital espontánea*.

En este nuevo tipo de *colaboración digital espontánea* (figura 1.2) destaca la carencia de una infraestructura de red fija, así como la falta de un administrador que ayude a configurar la red en cualquier momento. Esto ocasiona que las aplicaciones tradicionales de colaboración no se puedan utilizar para este tipo de escenarios.

Un claro ejemplo de un escenario de colaboración digital espontánea tiene lugar en las juntas fortuitas que involucran discusiones acerca del diseño o definición de algún proceso en donde los participantes dibujan esquemas por ejemplo en un pizarrón, mientras otro participante puede referirse a tales esquemas y propone modificaciones por alterar los dibujos. La característica esencial de este escenario es que los participantes se encuentran en un lugar en donde no cuentan con computadoras de escritorio ni con infraestructura de red, por lo que no es posible utilizar un sistema de pizarrón compartido. Sin embargo cada uno de los participantes cuenta con un dispositivo digital móvil (por ejemplo una computadora portátil o un asistente digital) habilitado con tecnología de red inalámbrica los cuáles si formasen una red de manera espontánea podrían ser usados para colaborar.

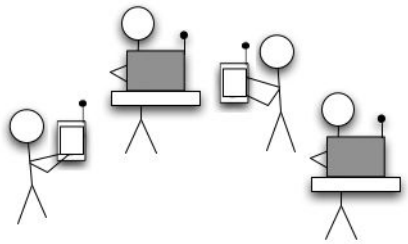

Figura 1.2: Colaboración digital espontánea

Así como éste, existen más escenarios de colaboración digital espontánea, por ejemplo en las zonas de desastres o en campos militares y, en general, nos referimos a todos aquellos escenarios de colaboración en los cuales:

- La existencia se da de una manera casual
- Los participantes se encuentran en el mismo sitio y en el mismo momento
- Se cuenta con dispositivos heterogéneos con recursos limitados
- Los dispositivos participantes tienen sistemas operativos heterogéneos
- No se cuenta con una infraestructura eléctrica
- No se pueda utilizar una infraestructura de comunicaciones basadas en cable

- No se puede hacer uso o no existen servicios computacionales como un servidor DNS o DHCP

La importancia de poder contar con aplicaciones de colaboración para este tipo de escenarios digitales ha creado la necesidad de desarrollar aplicaciones de colaboración que no estén limitadas a contar con una infraestructura de red preestablecida. Para lograr este objetivo es necesario dotar a los dispositivos móviles de mecanismos para establecer redes sin infraestructura.

1.3. Redes sin infraestructura preestablecida

A las redes que se forman entre dispositivos móviles sin la ayuda de una infraestructura preestablecida se les conoce como *redes* Ad-Hoc. Feeney [3] identifica las dificultades que existen para construir estas redes:

- *La red debe operar independientemente de una infraestructura de red preestablecida.*

  No se puede suponer la existencia de medios físicos como cables o un punto de acceso inalámbrico para que la red pueda generarse entre los diferentes dispositivos.

- *A pesar de que no se cuenta con una infraestructura, debe ser posible que las aplicaciones cuenten con los servicios administrativos de red.*

  Deben existir mecanismos que permitan a las aplicaciones contar con los servicios administrativos con los que se dispone en una red tradicional.

Para poder tener este tipo de redes, es necesario desarrollar protocolos eficientes de ruteo. Las investigaciones en el tópico de redes *Ad-Hoc* están enfocadas al desarrollo de algoritmos de ruteo [4, 5, 6, 7, 8, 9]. Este trabajo está enfocado a un subconjunto de las redes *Ad-Hoc* llamadas *redes espontáneas*. Existen diferentes definiciones de lo que es una *red espontánea*, primero Preuß y Cab [10] la definen como la integración de servicios y dispositivos dentro de un ambiente de red con el objetivo de habilitar un servicio instantáneo sin ninguna intervención manual. Un año después aparece otra definición de *red espontánea* pero enfocada a la colaboración digital y es de Feeney [3] que las identifica como las redes que se establecen con el único fin de realizar una colaboración digital y presentan las siguientes características:

- Ocurren en espacio y tiempo bien delimitados
- Los nodos inalámbricos que formarán la red son computadoras portátiles, asistentes digitales y teléfonos celulares
- Los nodos inalámbricos utilizan alguna tecnología inalámbrica como 802.11b, infrarrojo o Bluetooth
- A pesar de la naturaleza espontánea de la red, se espera que ocurran cambios pero que no sean tan frecuentes

La naturaleza espontánea de estas redes imponen retos para su construcción como [3]:

- *La red no está planeada*

  Las aplicaciones desconocen cuales van a ser los nodos que serán parte de la red.

- *Las fronteras de la red están pobremente definidas.* En cualquier momento, diferentes nodos de la red pueden entrar o salir.

- *Los nodos no están pre-configurados*

  La naturaleza espontánea de la colaboración limita el conocimiento a priori de que nodos participarán.

- *No hay servidores centrales*

  Las aplicaciones no pueden depender de la existencia de nodos que proporcionen servicios centralizados debido a que los nodos pueden llegar a formar particiones de la red o salir por completo.

- *Los usuarios no son expertos*

  El uso de estas redes debe ser transparente para el usuario y se deben evitar configuraciones engorrosas.

Cada uno de estos retos añade complejidad a la programación de las aplicaciones y esto ocasiona que su configuración y uso sea muy distinto al de las tradicionales. Por ejemplo, para el uso de un servicio en una red tradicional es necesario conocer la dirección IP y el puerto, y en caso de que se desconozca, se pregunta al servidor de nombres (DNS) dicha información.

En las redes espontáneas esto no es posible, la red no está planeada y no se puede suponer la existencia de servidores centrales como lo es el DNS, por lo tanto se necesita implementar protocolos para obtener la localización de un servicio en la red, a estos protocolos se les llama *Protocolos de descubrimiento de servicios (SDP)* [11] los cuáles se discuten en el capítulo 3.

Otro aspecto muy importante es que las fronteras en la red espontánea están pobremente definidas y es muy probable que la red sufra una división, lo cual es poco probable en una red tradicional.

Otro problema, no menos importante, es que por la naturaleza espontánea de la colaboración es muy factible que los dispositivos sean heterogéneos y cada uno de ellos tenga diferentes capacidades, por lo que es necesario considerar:

Sistemas operativos heterogéneos

Los dispositivos son diferentes y también los sistemas operativos. Generalmente, estos tienen características que permiten aprovechar las capacidades de los dispositivos, por lo cual, el manejo de los recursos se realiza de forma muy diferente.

Manejo de recursos eficiente

Los dispositivos pequeños tienen menos capacidad de procesamiento, menos capacidad de almacenamiento y menos capacidades de entrada y salida. Para programarlos hay que hacer uso eficiente de los recursos ya que son escasos.

Aplicaciones robustas

Este punto está muy ligado con el anterior, las aplicaciones en los dispositivos móviles tienen que ser aún más robustas que las aplicaciones para computadoras de escritorio. Una mala programación puede hacer que el dispositivo deje de operar.

Interfaces de usuario adecuadas

Los dispositivos móviles pequeños como asistentes digitales o teléfonos celulares tienen pantallas más pequeñas y generalmente carecen de periféricos como teclados. Es importante tomar en cuenta lo molesto que es para el usuario tener que llenar varios campos de texto. El modelo de interfaz gráfica de usuario que usan las computadoras de escritorio, quizás, no es el más adecuado para los dispositivos pequeños.

1.4. Planteamiento del proyecto

Como mencionamos anteriormente gracias a los avances en la tecnología de integración a gran escala, la microelectrónica y las telecomunicaciones fue posible la construcción y la proliferación de una gran variedad de dispositivos móviles equipados con tecnología inalámbrica; esto abrió paso a la posibilidad de hacer realidad escenarios de colaboración digital espontánea. Este trabajo propone una capa intermedia de software constituida por un conjunto de protocolos que complementen la carencia de una infraestructura de red con servicios centralizados, así como eliminar la necesidad de un experto para la configuración de la red. Además debe proporcionar mecanismos adecuados para colaborar en un ambiente dinámico y con restricción de recursos, todo esto con el fin de lograr adicionar funcionalidad a los dispositivos personales como computadoras portátiles y asistentes digitales que a través de aplicaciones y sobre redes espontáneas, pueden servir como herramientas de colaboración sin necesidad de configuración por parte del usuario.

Específicamente se propone utilizar la tecnología de red IEEE 802.11b en su modo *Ad-Hoc* y los protocolos de configuración nula para la generación de la red espontánea.

Para el desarrollo de este proyecto se contempló cumplir con las siguientes etapas:

- Prueba y uso de la tecnología IEEE 802.11b en los diferentes dispositivos
- Implementación de los mecanismos de configuración nula: auto-asignación de IP y descubrimiento de servicios en el asistente digital
- Diseño y desarrollo del conjunto de clases que hagan una abstracción de las dos capas anteriores y faciliten la utilización de las mismas en los diferentes dispositivos móviles
- Desarrollo, diseño e implementación de los mecanismos que servirán como base para realizar la colaboración entre las diferentes aplicaciones que tienen interacción en la red en los diferentes sistemas móviles
- Empleo de los componentes desarrollados en las etapas anteriores para el desarrollo de una aplicación de *colaboración digital espontánea*

1.5. Contribuciones de la tesis

En esta investigación se demuestra que con dispositivos portátiles y mediante el empleo de sus capacidades para comunicación inalámbrica, es posible formar una red en cualquier lugar y en cualquier momento.

La principal aportación alcanzada es que se diseñó e implementó un conjunto de componentes de software especializados que conforman la capa intermedia **IRIS** que junto con la tecnología inalámbrica IEEE 802.11b proporcionan a las aplicaciones los elementos necesarios para que puedan colaborar sobre redes espontáneas. Con esto, es posible desarrollar escenarios en los que varios usuarios se reúnen en cualquier lugar y a través de sus dispositivos desempeñan alguna actividad de colaboración, por ejemplo, la edición de un texto en forma colaborativa o un pizarrón compartido.

Además, se diseñó e implementó **IRIS** con el propósito de permitir a los desarrolladores crear de manera sencilla las aplicaciones de colaboración espontánea sin que se preocupen por la programación de los protocolos de comunicación y movilidad entre los nodos heterogéneos, permitiendo que se enfoquen primordialmente en crear aplicaciones más robustas de colaboración, orientando sus esfuerzos a la parte de la funcionalidad de la aplicación, lo cual repercute en la disminución del tiempo invertido en el desarrollo de este tipo de aplicaciones.

Por último se diseño e implementó un pizarrón compartido para actividades de colaboración espontánea. La pagina web del proyecto es:

http://homepage.mac.com/jabriones/sbravo/proyectos/IRIS/IRIS.html

1.6. Trabajo relacionado

Este proyecto se realizó en paralelo con el proyecto "Lucrn: Un middleware para el desarrollo de aplicaciones en redes espontáneas de dispositivos móviles Bluetooth" [12] realizado por el Maestro en Ciencias Jorge Alfonso Briones García. Desde el inicio de ambos proyectos se tenía claro que las tecnologías de comunicación imponen diferentes rangos de aplicaciones, sin embargo es interesante puntualizar las características que los distinguen, éstas se muestran en la tabla 1.1:

	Lucrn	IRIS
Tecnología de comunicación	Bluetooth	WiFi
Alcance	10 mts	100mts
Tasa de transmisión de datos	1Mbps	11Mbps
Costo del adaptador	$23.60	$80
Protocolo TCP/IP	No	Si
Red de multisalto	Si	No
Topología de red	maestro-esclavo	canal compartido
Usa infraestructura	No	Opcional
Rango de aplicacione	No adecuado para aplicaciones que transfieran grandes volumenes de información	Adecuado para aplicaciones que transfieran video y multimedia
Plataformas	Palm OS, Mac OSX	Familiar Linux, Mac OS X

Tabla 1.1: Tabla comparativa de **IRIS** y **Lucrn**

1.7. Contenido del documento

El documento está organizado como sigue. En este capítulo motivamos la noción de colaboración digital espontánea a través del análisis de los escenarios de colaboración que son posibles gracias a la proliferación de los dispositivos móviles y a la disponibilidad de tecnologías de red inalámbrica. En el capítulo 2 presentamos la tecnología de comunicación inalámbrica que utilizamos para la construcción de la infraestructura de software y especificamos las razones por las que se eligió. En el capítulo 3 se discute el diseño en capas de la infraestructura de software **IRIS** que sirve como base para el desarrollo de las aplicaciones de colaboración espontánea y se presenta cada uno de los protocolos y mecanismos que la componen. En el capítulo 4 se discute la implementación de la infraestructura de software **IRIS** y se argumenta la elección de los dispositivos, las plataformas y los ambientes de desarrollo utilizados. En el capítulo 5 se presenta un pizarrón compartido para el intercambio de información en grupos de trabajo que hacen uso de la implementación de la infraestructura de software **IRIS**. Finalmente, el capítulo 6 presenta las conclusiones y las perspectivas de extender este trabajo.

1.8. Convenciones usadas en este documento

Este documento tiene las siguientes convenciones tipográficas:

Itálica

Usada al escribir palabras técnicas en otro idioma y para definir términos nuevos o conceptos a los que le sigue una explicación.

Bold

Usada al escribir nombres de programas y bibliotecas.

inclinada

Usada al escribir nombres de clases.

`tipo molde`

Usada al escribir nombres de métodos y funciones.

2

Tecnologías para establecer redes espontáneas

Para hacer uso de una tecnología es importante conocerla a fondo, ya que conocer a fondo una tecnología permite saber cuáles son sus alcances y sus limitaciones. Para este trabajo es importante saber si la tecnología de red inalámbrica proporciona o no el soporte para la creación de redes espontáneas, es decir, que cumpla como se menciona en el capítulo 1 con las siguientes características:

- *La red debe operar independientemente de una infraestructura.*

  Para resolver este punto se propone el uso de la tecnología de red inalámbrica IEEE 802.11b que proporciona un tipo de configuración para redes *Ad-Hoc*.

- *A pesar de que no se tenga una infraestructura, debe ser posible que las aplicaciones cuenten con los servicios administrativos de red.* Para resolver este punto y debido a las limitaciones del estándar IEEE 802.11b se propone usar los protocolos de configuración nula [13] con el fin de proporcionar a las aplicaciones los servicios administrativos con los que se dispone en una red tradicional los cuáles se presentan en el capítulo siguiente.

En este capítulo hacemos una breve revisión de la tecnología de comunicación inalámbrica IEEE 802.11 que es parte de la arquitectura propuesta por este trabajo en el capítulo 3. Para la discusión de esta tecnología se presenta su historia, se explica la capa física y la capa de acceso al medio que

la distingue de las otras tecnologías IEEE 802. Además, se presentan los dos modos que el estándar proporciona para formar una red.

2.1. Comunicación inalámbrica

Existen diferentes tecnologías de radio frecuencia, pero las más populares son IEEE 802.11 y Bluetooth [14]. En el mercado ya se encuentran dispositivos con estas tecnología y, en particular, a los dispositivos basados en la especificación estándar IEEE 802.11b se les llama Wi-Fi (por las palabras en inglés *Wireless Fidelity*).

Durante la primera mitad del 2000, las unidades vendidas de integrados IEEE 802.11b fueron 7.24 millones [15], ubicando al estándar IEEE 802.11b como la tecnología inalámbrica más utilizada, lo que se traduce en mayor disponibilidad. Por tal razón, en un futuro cercano es muy probable que la mayoría de los dispositivos portátiles estén habilitados con esta tecnología. Para los fines de este trabajo, está claro que esta gran disponibilidad es una razón poderosa para incluirla en la arquitectura.

2.2. El estándar IEEE 802.11

La familia IEEE 802 es un conjunto de especificaciones para tecnología de redes de área local. En está se especifican las dos primeras capas del modelo OSI como se muestra en la figura 2.1. La especificación IEEE 802.11 es otra especificación de la capa de enlace de datos que usa la subcapa 802.2 de control lógico de enlace (LLC). La especificación 802.11 básica incluyen la capa de acceso al medio (MAC) y dos capas físicas [16].

En 1987 se formó el grupo de trabajo para el estándar 802.11 de redes de área local inalámbrica y en 1997 se presentó una primera versión del estándar 802.11 para redes inalámbricas de área local en la banda sin licencia 2.4Ghz, con una tasa de transmisión de datos de 1 a 2Mbps usando *direct-sequence spread spectrum* ó *frequency-hopping spread spectrum*. En 1999 fue adoptado el IEEE 802.11b que tiene una tasa de transmisión de datos de 11Mbps.

2.2.1. Capa física

La tecnología *spread-sprectum* es un requerimiento para dispositivos que operan en una banda sin licencia. La capa física basada en radio frecuencia

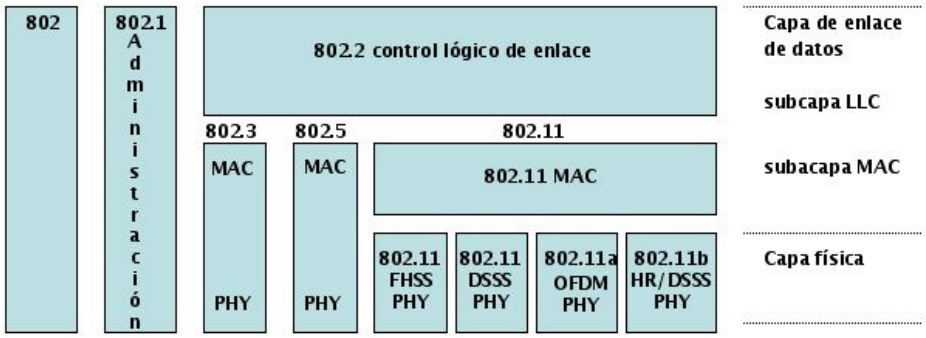

Figura 2.1: La familia IEEE 802

en el 802.11 usa tres diferentes técnicas *spread-sprectum* [16].

- *Frequency hopping (FH)*

  En esta técnica se salta de una frecuencia a otra con un patrón preestablecido, transmitiendo en intervalos cortos de tiempo en cada subcanal.

- *Direct sequence (DS)*

  En esta técnica se propaga la señal sobre el ancho de banda de la frecuencia aplicando transformaciones matemáticas.

- *Orthogonal Frecuency Division Multiplexing (OFDM)*

  En esta técnica se divide el canal disponible en varios subcanales y se propaga una porción de la señal a través de cada subcanal en paralelo.

En la tabla 2.1 se muestra una comparación de las diferentes versiones del estándar IEEE 802.11 [17]. La primera especificación de IEEE 802.11 incluía una capa con tecnología de luz infrarroja que no fue muy utilizada en comparación con las otras dos capas físicas con tecnología *spread-sprectum* FH y DS, ya que la tasa de transmisión de datos estaba limitada a 2Mbps. En 1999 los productos 802.11b podían operar con una tasa de transmisión de datos de hasta 11Mbps. En 2000 se liberan productos de la versión 802.11a que usa la tecnología *spread-sprectum* OFDM que incrementa la tasa de transmisión de datos a 54Mbps sin embargo opera una banda de frecuencia de 5GHz que está aprobada por las autoridades reguladoras sólo en Estados Unidos. Aunque la versión 802.11g utiliza la misma técnica *spread-sprectum* opera en la

banda de frecuencia 2.4GHz. La banda de frecuencias de 2 400 a 2 500 MHz está atribuida en otros países para los sistemas *spread-sprectum*. En México sólo se considera el segmento de 2 400 a 2 483,5 MHz sin licencia [18].

Estándar IEEE	Capas físicas	Velocidad	Frecuencia	Liberación
802.11	FHSS DSSS	1 Mbps 2 Mbps	2.4 GHz	Primer estándar (1997)
802.11a	OFDM	hasta 54 Mbps	5 GHz	Segundo estándar (1999)
802.11b	HR DSSS	5.5 Mbps 11 Mbps	2.4 GHz	Tercer estándar, es el más popular.
802.11g	OFDM	hasta 54 Mbps	2.4 GHz	Cuarto estándar(2001), puede interoperar con nodos 802.11b

Tabla 2.1: Comparación de estándares IEEE 802.11

2.2.2. Capa MAC

El método de acceso del protocolo de la capa MAC IEEE 802.11b es el DCF *Distributed Coordination Function* que es un protocolo MAC CSMA/CA *Carrier Sense Multiple Access with Collision Avoidance*.

A diferencia de otros protocolos de la capa de enlace, 802.11 incorpora acuse de recibo (ack). Todas las tramas transmitidas deben tener un acuse de recibido (ack). La secuencia del envío de trama y recepción del acuse es una operación atómica. Si cualquier parte de la transmisión falla, la trama se considera perdida. El protocolo permite a las estaciones 802.11 bloquearse durante operaciones atómicas así que la secuencia atómica no es interrumpida por otra estación intentando usar el medio de transmisión.

CSMA/CA es derivado de CSMA/CD (*Collision Detection*), el cual se usa en Ethernet. La principal diferencia es evitar colisiones (CA). En un cable, los que transmiten están habilitados para escuchar mientras transmiten y así detectan las colisiones, pero incluso si un nodo inalámbrico pudiera escuchar en el canal mientras transmite, la fuerza de su propia transmisión enmascararía todas las otras señales en el aire, por lo que el protocolo no puede directamente detectar colisiones como en Ethernet. Para prevenir colisiones las estaciones usan señales RTS *Request to Send* y CTS *Clear to Send* para reservar el canal de transmisión.

El protocolo empieza por escuchar sobre el canal. Si el canal está desocupado, el nodo envía el primer paquete de su cola de transmisión. Si el canal

está ocupado como se muestra en la figura 2.2 con el nodo 1, el nodo espera el final de la transmisión y entonces empieza la contención esperando un tiempo aleatorio. Cuando el tiempo termina y si el canal está desocupado el nodo envía su paquete. Debido a que la contención es un número aleatorio y es diferente por cada paquete, cada nodo tiene una oportunidad equitativa para acceder al canal.

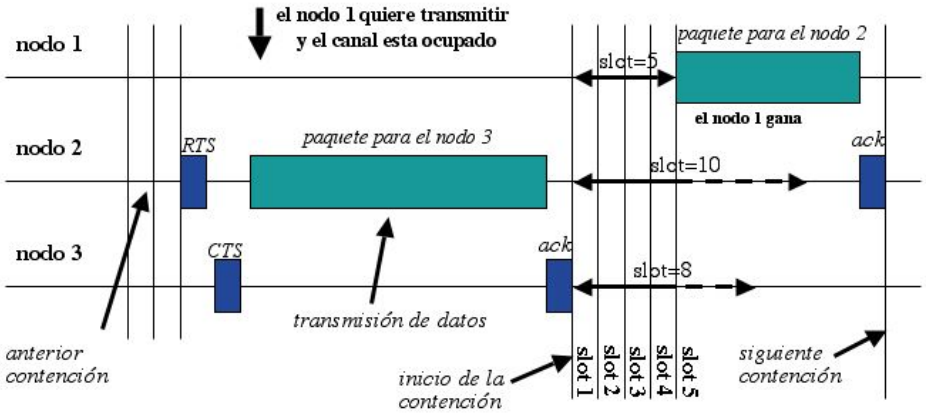

Figura 2.2: Mecanismo de acceso al canal CSMA/CA

2.2.3. Componentes de una red IEEE 802.11

Las redes 802.11 consisten de 4 componentes principales que se muestran en la figura 2.3 y que definimos a continuación:

Punto de acceso

Es un dispositivo que tiene como función principal ser el puente entre las red con cable y la red inalámbrica.

Sistema de distribución

Cuando varios puntos de acceso son conectados con el fin de cubrir una área grande, se pueden comunicar unos con otros gracias a este componente lógico que 802.11 usa para re-enviar las tramas a sus destinos.

Medio inalámbrico

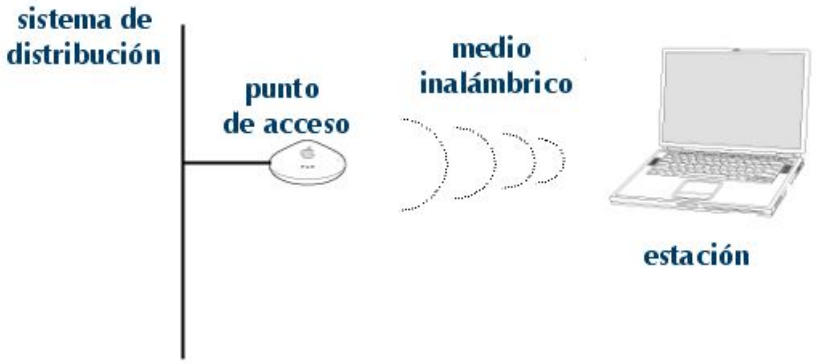

Figura 2.3: Componentes de una red 802.11

Para mover las tramas de una estación a otra, el estándar usa un medio inalámbrico que es el canal por el que se transmiten las tramas. Como se mencionó anteriormente existen varios componentes en la capa física que utilizan diferentes técnicas de radio frecuencia.

Estación

Las estaciones son computadoras con interfaces de red inalámbrica. Típicamente, las estaciones son computadoras portátiles o asistentes digitales.

2.2.4. Tipos de red de área local IEEE 802.11

El bloque de construcción básico de red IEEE 802.11 es el *basic service set* (BSS), que es un grupo de estaciones que se comunican con cada una de las otras estaciones del grupo. La comunicación se lleva a cabo dentro de un área, llamada el *área de servicio básica*, que se define por las características del medio de comunicación inalámbrico. Cuando una estación se encuentra en el área de servicio básica, ésta puede comunicarse con los otros miembros del BSS de dos maneras:

- *BSS con infraestructura*

  Ésta se distingue por el uso de un punto de acceso para cualquier comunicación incluyendo la comunicación entre nodos en la misma área

de servicio.

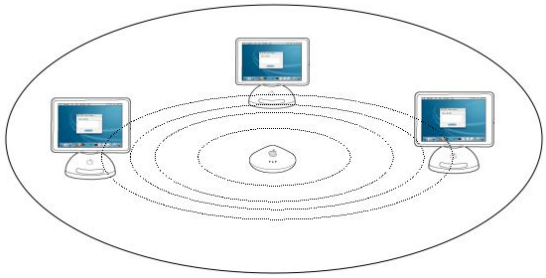

Figura 2.4: BSS con Infraestructura

- *BSS independiente (IBSS)*

  Los nodos en una red IBBS se comunican directamente con cada uno de los otros nodos y por eso deben encontrarse en la misma área de servicio.

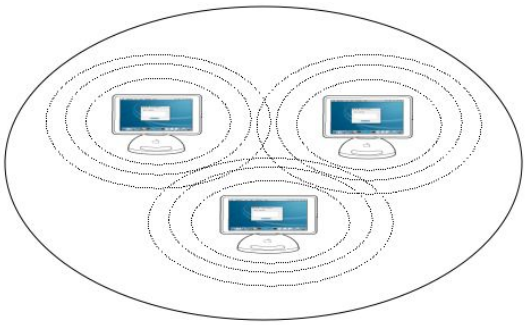

Figura 2.5: BSS Independiente (IBSS)

2.2.5. Operaciones de administración

Antes de usar cualquier red es necesario encontrarla. Al proceso de identificar si existen redes en el área es llamado *escaneo*. Existen dos formas para realizar el escaneo en el estándar 802.11 que son:

Escaneo pasivo

El escaneo pasivo ahorra energía porque no requiere transmisión. La estación se mueve solamente a través de una lista de canales y espera por mensajes intermitentes que envían las estaciones para notificar que existe una red. En la figura 2.6 la estación usa un escaneo pasivo para encontrar las redes activas en el área y recibe las tramas intermitentes de tres puntos de acceso.

Figura 2.6: Escaneo pasivo

Escaneo activo

En el escaneo activo como se muestra en la figura 2.7 la estación toma un rol más activo. En cada canal manda mensajes de solicitud por información de las redes activas. Una estación de cada red es responsable de responder al mensaje de solicitud.

Figura 2.7: Escaneo activo

Una vez que se encontró la red a la que se quiere pertenecer se cuenta con los servicios de administración que la especificación 802.11b proporciona [16]:

- *Distribución*

  Un punto de acceso asociado a una estación móvil que quiere enviar datos a través de la red, usa el servicio de distribución para entregar los datos a su destino.

- *Integración*

  Este servicio permite la conexión del sistema de distribución a un red que no es IEEE 802.11b.

- *Asociación*

  Este servicio le permite a las estaciones móviles registrarse con un punto de acceso que utiliza la información para la distribución de datos en la red.

- *Re-asociación*

  El uso de este servicio es iniciado por las estaciones móviles cuando las condiciones de la señal indican que una asociación diferente puede beneficiar.

- *Disociación*

  Las estaciones móviles utilizan este servicio para terminar una asociación existente con un punto de acceso.

- *Autenticación*

  La autenticación es un requisito previo a la asociación por que sólo los usuarios autenticados están autorizados a usar la red. Existen dos tipos de autenticación especificadas por el estándar 802.11: la autenticación de sistema abierto y la autenticación de llave compartida.

- *De-autenticación*

  Este servicio da por terminada cualquier asociación en curso.

- *Privacidad*

  Para ofrecer un nivel similar de privacidad al que ofrecen las redes con cable, 802.11b proporciona un servicio opcional de privacidad llamado *Wired Equivalent Privacy*(WEP).

2.3. Limitaciones de la tecnología IEEE 802.11

El estándar IEEE 802.11b, en su modo llamado *Ad-Hoc* ó IBSS, es el que nos interesa por que no necesita infraestructura. La arquitectura es punto a punto ya que con el estándar 802.11b no se pueden hacer redes *Ad-Hoc multi-hop*, ya que sólo los dispositivos dentro del área de servicio básica pueden comunicarse. Para poder tener este tipo de redes, es necesario desarrollar protocolos eficientes de ruteo [19, 20, 21]. Esta limitación no afecta a esté trabajo ya que está enfocado sólo a redes locales, porque el fin principal es construir aplicaciones colaborativas locales.

El IEEE 802.11b trabaja en las bandas ISM(*Industrial Scientific and Medical*) que es 2.4GHz. Debido a su naturaleza que no requiere licencia, estas bandas son ocupadas por dispositivos que implementan otras especificaciones inalámbricas, así la coexistencia es un aspecto importante a considerar.

El estándar 802.11 no tiene protocolos de configuración nula que proporcionen a las redes espontáneas los servicios de:

- Asignación de dirección
- Traducción de nombres
- Descubrimiento de servicios

Todas estas limitaciones de la tecnología IEEE 802.11 para la construcciones de redes espontáneas son subsanadas, como se presenta en el siguiente capítulo, al adicionar protocolos que conforman una arquitectura que permite la construcción de redes espontáneas entre dispositivos habilitados con la tecnología IEEE 802.11b y que, unidos a mecanismos de colaboración y movilidad, permiten la construcción de escenarios de colaboración digital espontánea.

3

La infraestructura de software IRIS

En el capítulo 1 describimos por qué las aplicaciones de *colaboración digital espontánea* son un campo de investigación interesante y relevante en computación y por qué estas aplicaciones son difíciles de construir. En el capítulo 2 discutimos la tecnología de red inalámbrica que usamos como la base de comunicación de las aplicaciones de *colaboración digital espontánea*. En este capítulo presentamos la capa intermedia **IRIS** que proporciona las características necesarias para hacer posible el desarrollo de este tipo de aplicaciones.

La infraestructura de software **IRIS** permite que los diseñadores de aplicaciones de *colaboración digital espontánea* gasten menos tiempo y esfuerzo en los detalles que son comunes como: la administración y configuración de la red para habilitar la comunicación entre los participantes (la cual en los anteriores escenarios de colaboración era definida a priori por un experto [2]), así como el manejo del contexto de colaboración que involucra el conocimiento de la llegada y salida de los dispositivos que participan en la colaboración. De esta manera los desarrolladores pueden orientar sus energías a la meta principal de estas aplicaciones, es decir, especificar las acciones y el comportamiento que definen la funcionalidad de la aplicación.

3.1. Ubicación de la infraestructura de software IRIS

La infraestructura de software **IRIS** es un conjunto de protocolos que proporciona los mecanismos que hacen falta para el desarrollo de las aplicaciones de *colaboración digital espontánea* que surgen principalmente a raíz de la falta de una infraestructura de red centralizada y la carencia de un administrador que configure cada uno de los dispositivos de los participantes en la colaboración. Estos mecanismos deben enfrentar los retos impuestos por la naturaleza espontánea de la colaboración que son:

- La detección, asociación y creación de una red entre los dispositivos
- La configuración de cada uno de los nodos
- La integración de cada uno de los nodos a la red
- La salida de cada uno de los nodos de la red
- La difusión de mensajes en la red
- La colaboración entre las aplicaciones

En el modelo de capas **IRIS** se sitúa como se muestra en la figura 3.1 entre el sistema operativo del dispositivo y las aplicaciones de *colaboración digital espontánea* de tal manera que los recursos que utiliza la aplicación son administrados por el sistema operativo a través de la interfaz que es proporcionada por la capa **IRIS**. Además **IRIS** evita al programador las tareas que involucran la creación de la red espontánea, y el manejar el contexto de colaboración que es indispensable para cualquier aplicación de colaboración digital espontánea.

3.2. El diseño de IRIS

Debido a que la técnica de diseño orientada al uso de capas genera diseños sencillos que pueden ser comprendidos más fácilmente, que sirven para la definición de estándares y en los cuáles puede ser posible la sustitución o el intercambio de ciertas capas[22] se decidió utilizar este enfoque de diseño para realizar la infraestructura de software **IRIS**.

Figura 3.1: La ubicación de IRIS en un dispositivo móvil

En la figura 3.2 se muestran cada una de las capas que componen a la infraestructura de software **IRIS**, estás son nuestra propuesta para enfrentar los retos impuestos por la naturaleza espontánea de la colaboración que mencionamos en la sección anterior. A continuación presentamos la funcionalidad que cada una de las capas ofrece.

3.2.1. La capa para establecer la red

Esta capa contiene la tecnología inalámbrica que se utiliza para establecer la comunicación entre los nodos de la red. Como se discutió en el capítulo 2, la tecnología IEEE 802.11b tiene un modo de configuración *Ad-Hoc*, que permite la creación de una red sin la necesidad de tener un punto de acceso. La configuración *Ad-Hoc* garantiza con su método distribuido de acceso al medio, que con la salida de cualquier nodo, la red continua funcionando sin problemas.

3.2.2. La capa para lograr configuración nula

Esta capa contiene los protocolos de configuración nula [23] que se necesitan para evitar que los usuarios tengan que configurar sus dispositivos para participar en la red. Además permite que la aplicación publique y descubra

Figura 3.2: La arquitectura de IRIS

servicios de colaboración dentro de la red. Es muy probable que esta capa no esté implementada en todos las plataformas de los dispositivos móviles. En el caso especifico del sistema operativo Mac OS X ya se cuenta con la implementación de los protocolos de configuración nula, sin embargo es necesario

integrarlos a las aplicaciones, de tal forma que debemos tomarlos en cuenta como componentes necesarios de la arquitectura **IRIS**.

3.2.3. La capa de movilidad

Esta capa contiene la implementación de las acciones que se realizan cuando un nodo se adiciona como participante de la colaboración publicando su llegada y su salida. En el momento que sucede un evento de este tipo se avisa a los otros nodos participantes y esta capa es la encargada de actualizar el contexto de colaboración que contiene información de cada uno de los participantes y los canales de comunicación de entrada y salida, con cada uno de ellos.

3.2.4. La capa de colaboración

En esta capa se encapsula toda la utilización de las capas inferiores de manera que el programador no se preocupe por la forma como se habilita (creación de la red de manera espontánea) y se realiza la comunicación entre los nodos, así como tampoco de la detección de eventos de entrada y salida de los nodos que toman parte en la colaboración. Se encarga principalmente de recibir y validar las acciones de colaboración enviadas por los otros nodos participantes. Además contiene el estado de la colaboración, que es, cada una de las acciones que se ejecutaron desde el inicio de la colaboración.

3.2.5. La capa de funcionalidad de la aplicación

Esta capa contiene la implementación de las acciones que se realizan sobre el área compartida. Por ejemplo, en el caso de un pizarrón compartido, en esta capa se implementa las acciones de dibujo y la interfaz de la aplicación. De esta manera, el desarrollo de la aplicación en las diferentes plataformas, se enfoca a la implementación de las acciones para la colaboración, con la única restricción de que las acciones se llamen de la misma manera en la implementación de cada una de las plataformas, ya que el nombre de las acciones define el protocolo de colaboración de la aplicación.

3.3. La pila de protocolos de IRIS

Cada capa de **IRIS** proporciona un servicio a través de un protocolo que define un conjunto de reglas que rige la interacción entre procesos que se ejecutan concurrentemente en los diferentes dispositivos móviles. Para explicar cada uno de los protocolos se presentarán los cinco elementos que una especificación de un protocolo debe tener para estar completa [24]:

- El servicio que es ofrecido por el protocolo
- Las suposiciones acerca del ambiente en el cual se va a ejecutar
- El tipo de mensajes válidos para el protocolo (vocabulario)
- El formato de cada uno de los mensajes
- Las reglas que mantienen la consistencia del intercambio de mensajes que se lleva a cabo entre las diferentes entidades

3.3.1. Auto-asignamiento de dirección IP

La actual especificación del método de Auto-asignamiento de dirección IP se encuentra en [25]. Brevemente explicaremos este método por el cual un nodo puede automáticamente configurar una interfaz con una dirección IPv4 en el prefijo 169.254/16 que es válido para comunicación en una red local. Este mecanismo es especialmente valioso en ambientes donde ningún otro mecanismo de configuración está disponible.

Servicio

Proporciona una dirección IP para hacer posible la localización de este nodo en la red.

Suposiciones

- Este protocolo se aplica a cualquier medio IEEE 802 incluyendo IEEE 802.3, y otras tecnologías de red similares que operan a una tasa de transmisión de datos de al menos 1Mbps, que tengan una latencia de ida y vuelta de a lo más un segundo y que soporten el protocolo de resolución de direcciones (ARP *Address Resolution Protocol*).

- Este protocolo está pensado para pequeñas redes *Ad-Hoc*. A pesar de que existen en principio 65024 direcciones disponibles en el rango 169.254.x.x, intentar usar todas estas direcciones en una red local resultaría en períodos largos de tiempo para encontrar una dirección disponible. Con un número de nodos no mayor a 1300 conformando la red *Ad-Hoc* se tiene un 98 % de probabilidad de seleccionar una dirección no asignada en el primer intento.

- La semilla del algoritmo de generación de números pseudo aleatorios se debe escoger del tal manera que diferentes nodos no generen la misma secuencia de números. Si el nodo tiene acceso a información persistente que es diferente para cada nodo, tal como su dirección hardware de Ethernet, entonces el generador de números pseudo aleatorios podría usar como semilla un valor derivado de esta información. Los nodos que están equipados con almacenamiento persistente, pueden registra la dirección IP que ellos han seleccionado, y esta dirección debe ser usada como la primer candidata cuando se prueba por una dirección, con lo cual se incrementa la estabilidad del direccionamiento.

Vocabulario

- *ARP de prueba*

  Se usa para hacer referencia al paquete con formato ARP(figura 3.3) que es una petición difundida sobre la red local con el campo de dirección de envío IP en ceros. El campo de dirección hardware de envío debe contener la dirección hardware de la interfaz que envía el paquete. El campo de dirección hardware destino se ignora y debe contener ceros. El campo de dirección destino IP debe contener la dirección a ser probada.

- *ARP de anuncio*

  Se usa para hacer referencia al paquete ARP para difundir un anuncio sobre la red local, idéntico al ARP de prueba, excepto que los campos de dirección de envío y destino IP contienen la dirección IP a ser anunciada.

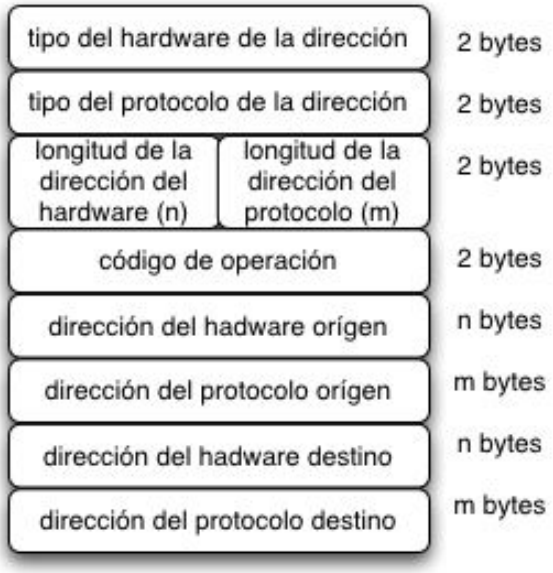

Figura 3.3: Formato de un mensaje ARP

Reglas

- *Selección de la dirección*

  Se selecciona una dirección usando un generador de números pseudo aleatorio con una distribución uniforme en el rango de 169.254.1.0 a 169.254.254.255. La semilla del algoritmo de generación de números pseudo aleatorios se debe escoger del tal manera que diferentes nodos no generen la misma secuencia de números.

- *Probar la dirección*

  El nodo debe esperar un intervalo de tiempo aleatorio seleccionado uniformemente en el rango de cero a dos segundos, y debe entonces enviar cuatro paquetes ARP de prueba, a intervalos de dos segundos. Si después de dos segundos de que se transmitió el último paquete ARP de prueba no se recibe ningún paquete ARP de respuesta por conflicto, entonces el nodo ha declarado exitosamente la dirección de enlace local.

- *Anunciar la dirección*

  El nodo envía dos paquetes ARP de anuncio a toda la red, espaciados

por dos segundos. Un paquete ARP de anuncio es idéntico a un paquete ARP de prueba, excepto que las direcciones IP de envío y destino contienen la dirección IP seleccionada recientemente.

- *Defender la dirección*

  La detección de colisiones de dirección es un proceso constante que está en efecto todo el tiempo que un nodo está usando la dirección de enlace local IPv4.

  En cualquier momento, si un nodo recibe un paquete ARP (de respuesta o solicitud) donde la dirección de envío IP es su propia dirección IP, pero la dirección hardware de envío no empata con la de su interfaz, entonces éste es un paquete ARP de conflicto, indicando una colisión de dirección. Un nodo debe responder a un paquete ARP de conflicto de una de las siguientes maneras:

  1. Optar por configurar inmediatamente una nueva dirección IP de enlace local como se describió anteriormente.
  2. Preferir mantener la misma dirección IP, entonces registra el tiempo en que recibió el paquete ARP de conflicto, y envía un simple ARP de anuncio difundido a toda la red, dando su propia dirección IP y dirección hardware como las direcciones de envío del ARP.

Es importante mencionar que los parámetros definidos en la especificación [25] y mencionados en las anteriores reglas acerca de:

- El número de paquetes ARP de prueba enviados y el intervalo entre cada uno
- El número de paquetes ARP de anuncio enviados y el intervalo entre cada uno
- El intervalo de tiempo que se espera para declarar una dirección IP como asignada
- El intervalo de tiempo entre los paquetes ARP de conflicto bajo el cual un nodo debe optar por configurar una nueva dirección IP

Están ligados a la suposición de que la tecnología de red opera a una tasa de transmisión de datos de al menos 1Mbps y tiene una latencia de ida y vuelta de a lo más un segundo.

Diagrama de flujo

En las figuras 3.4 y 3.5 se presentan, en un subconjunto del lenguaje SDL(*Specification and Description Language*)[24], los diagramas de flujo que corresponden a las reglas anteriormente expuestas.

Ejemplo

El proceso de auto-asignación de dirección IP que realiza un dispositivo para formar parte de una red comienza como se muestra en la figura 3.6 cuando el dispositivo genera aleatoriamente la dirección IP 169.254.150.84 y envía cuatro paquetes ARP de prueba para saber si la dirección es utilizada por algún dispositivo en la red. Cuando no se recibe después de dos segundos de que se transmitió el último paquete ARP de prueba ningún paquete ARP de respuesta por conflicto, entonces el dispositivo se asigna la dirección IP 169.254.150.84.

Si la dirección 169.254.150.84 está asignada a algún dispositivo de la red como se muestra en la figura 3.7 se recibe un paquete ARP de respuesta por conflicto y entonces el dispositivo intentará por otra dirección como se muestra en la figura 3.8. El dispositivo genera aleatoriamente la dirección IP 169.254.70.101 y envía cuatro paquetes ARP de prueba para saber si la dirección es utilizada por algún dispositivo en la red. Cuando no se recibe después de dos segundos de que se transmitió el último paquete ARP de prueba ningún paquete ARP de respuesta por conflicto, entonces el nodo se asigna la dirección IP 169.254.70.101.

3.3.2. Protocolo de descubrimiento de servicios

La actual especificación del protocolo de descubrimiento de servicios se encuentra en [26]. Este protocolo está diseñado para tener el mayor parecido posible con el DNS convencional. La diferencia principal es que las solicitudes se envían vía *multicast* a todos los nodos locales, en vez de enviarse vía *unicast* a un especifico servidor conocido, lo cual no es posible saber dado que la red es espontánea.

Servicio

- Traducción del nombre a la dirección

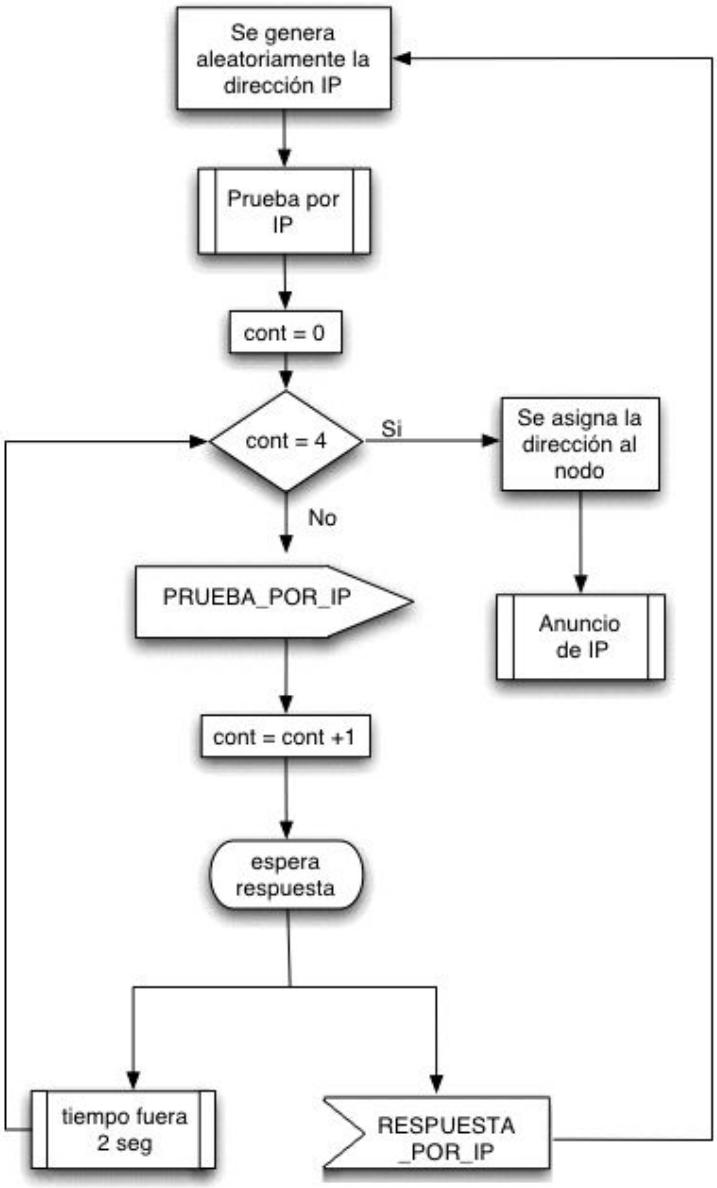

Figura 3.4: Diagrama de flujo para probar por la dirección IP

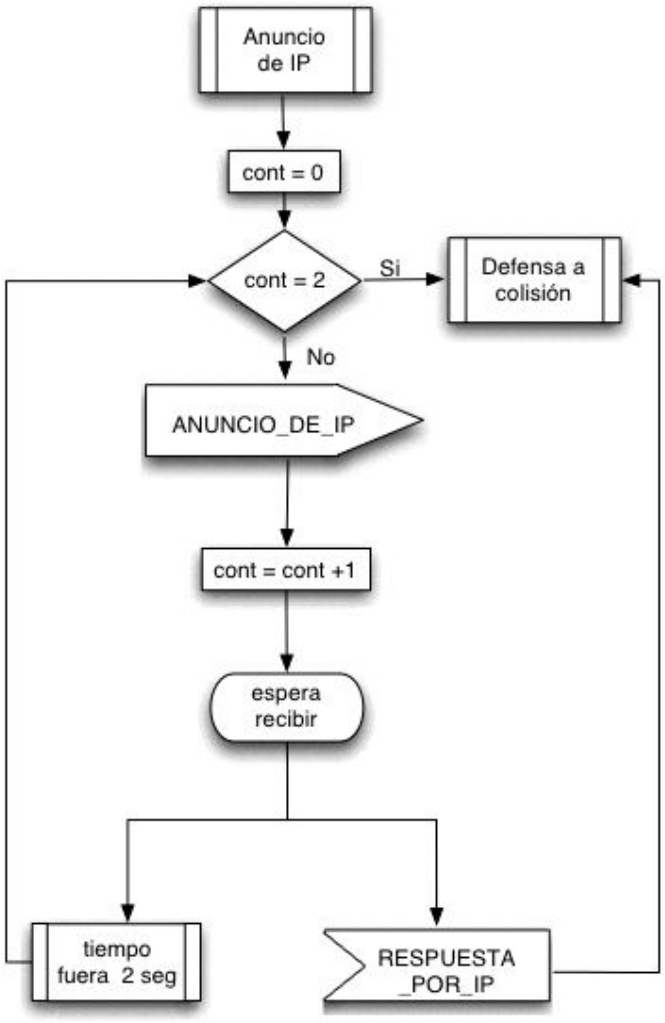

Figura 3.5: Diagrama de flujo para anunciar la dirección IP

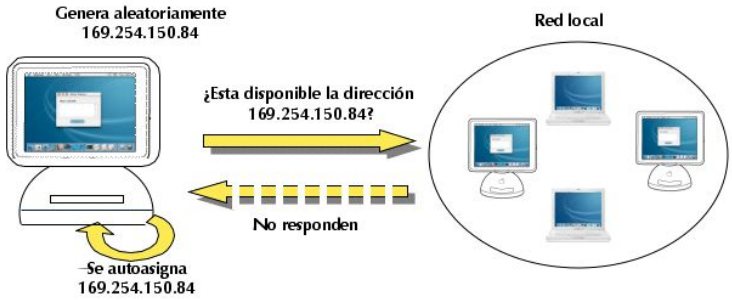

Figura 3.6: Auto-asignación de IP

Figura 3.7: Prueba con una dirección IP ya asignada

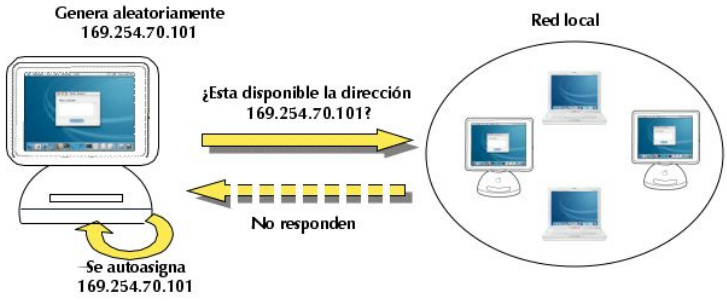

Figura 3.8: Prueba a otra dirección IP

- Publicación y descubrimiento de servicios en la red

Conceptos básicos

- El protocolo usa el mismo formato de paquete como el *unicast* DNS que se muestra en la figura 3.9, el mismo nombre de estructura y el mismo tipo de registro DNS

- *multicast DNS Responder* (mDNSResponder) es el componente que escucha por paquetes DNS[27], enviados vía IP *multicast* al puerto UDP(User Datagram Protocol) 5353. A estos paquetes les llamamos mDNS

Suposiciones

- El dominio es *.local.* y todos sus subdominios pertenecen al rango 169.254/16 y tienen significado sólo en el enlace local donde se generaron

- Cada nodo en la red tiene un mDNSResponder activo

Reglas

- *Dirección de envío de solicitud*

  Cualquier solicitud DNS por un nombre que termina con *.local.* se debe enviar a la dirección *multicast* 224.0.0.251.

- *Probar por el nombre del servicio*

  El primer paso para anunciar un servicio es mandar un paquete mDNS de consulta preguntando por el nombre del servicio. Si después de realizar la consulta se recibe un paquete de respuesta de conflicto mDNS entonces se debe escoger otro nombre para el servicio y realizar el primer paso para anunciar un servicio. Si después de 250ms no se recibe un paquete de respuesta de conflicto mDNS, se envía un segundo y se esperan otros 250ms para enviar un último paquete de consulta mDNS. Si después de 250ms no se recibe ningún paquete de respuesta de conflicto mDNS, ya se puede anunciar el servicio.

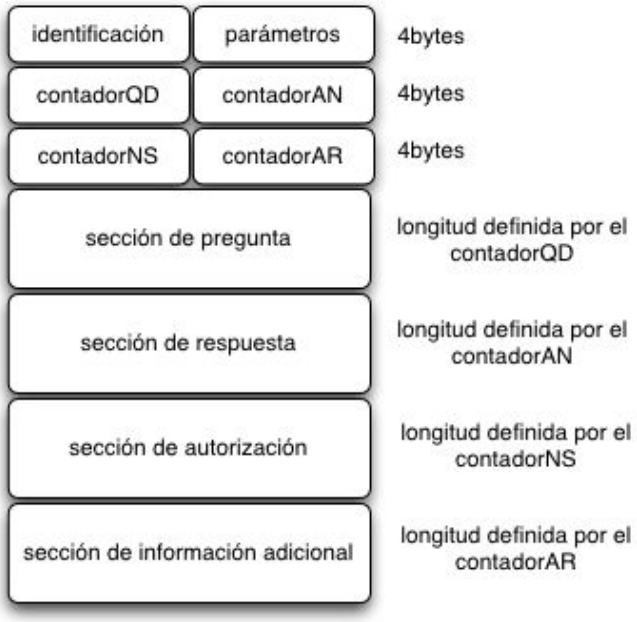

Figura 3.9: Formato de un mensaje DNS

- *Anunciar el servicio*

  Para anunciar el servicio se manda un paquete de respuesta gratuito mDNS, que contiene en la sección de respuesta, el registro DNS, del servicio que se va a publicar.

- *Descubrir el servicio*

  Se envía un paquete de consulta mDNS, que contiene en el registro DNS, el tipo del servicio. Entonces, espera por un paquete de respuesta mDNS que los mDNSResponder de cada nodo de la red que proporcione un servicio de ese tipo le envía.

Ejemplo

En la figura 3.10 se muestra un ejemplo de la publicación del servicio pizarra en el nodo apolo. Primero se mandan tres paquetes de consulta mDNS a la red a la dirección *muilticast* 224.0.0.251 al puerto 5353.

Todos los mDNSResponder de cada nodo de la red reciben el paquete pero ninguno manda un mensaje de respuesta, ya que ningún nodo tiene activo un servicio con ese nombre. Como no se recibe respuesta se activa el servicio pizarra en el nodo apolo en el puerto 1010 y después se envía un paquete de respuesta gratuito mDNS a la red a la dirección *muilticast* 224.0.0.251 al puerto 5353. Todos los mDNSResponder de cada nodo de la red reciben el paquete y almacenan esta información.

En la figura 3.11 se muestra un ejemplo del descubrimiento de un servicio. En este caso se quiere utilizar un servicio de tipo pizarra pero no sabemos en que dirección ni en que puerto se encuentra un servicio de este tipo. Entonces, enviamos un paquete de consulta mDNS a la red a la dirección *muilticast* 224.0.0.251 al puerto 5353. Todos los mDNSResponder de cada nodo de la red reciben el paquete pero solo el nodo apolo que tienen activo el servicio pizarra manda una respuesta mDNS que contiene el registro DNS que contiene la dirección IP y el puerto del servicio.

3.3.3. Protocolo de movilidad

El protocolo de movilidad trata principalmente con el problema de la movilidad de los nodos. Auxiliado por el protocolo de descubrimiento de servicio, se detecta eventos de salida y entrada de un servicio. Sin embargo, existen acciones que un nodo debe ejecutar como participante en una tarea de colaboración, por ejemplo, necesita obtener el estado de la colaboración ya que es muy probable que un participante llegue después de iniciada la colaboración. También, se utiliza para modificar dinámicamente la información de los nodos que han entrado o salido y que cada uno de los participantes almacena en lo que llamamos contexto de colaboración.

Servicio

El servicio que ofrece este protocolo es la administración del contexto de colaboración cuando se anuncia que un participante en la colaboración entró o salió.

Suposiciones

- El dispositivo recibe notificaciones de la capa de configuración nula de que un servicio está publicando la salida o la entrada

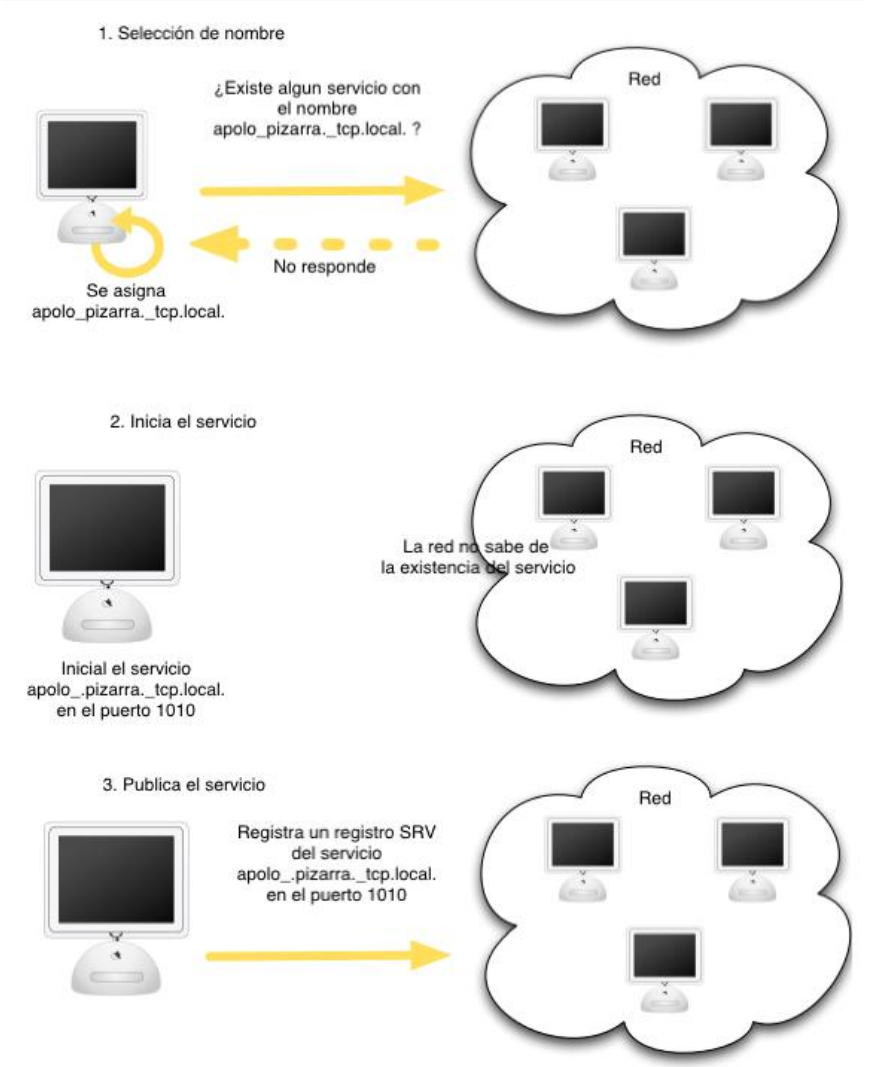

Figura 3.10: Publicación de un servicio.

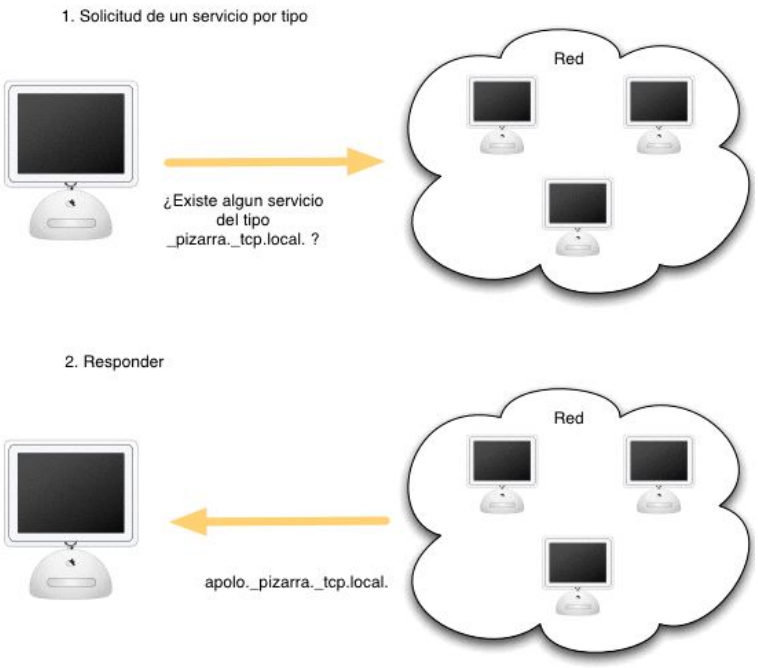

Figura 3.11: Descubrimiento de un servicio.

- Los dispositivos tienen una lista de los sockets que tienen abiertos
- Los dispositivos tienen una tabla la cual está formada por el nombre, la dirección IP y el puerto del servicio que proporciona el nodo y el socket por el cual se puede tener acceso a él

Mensajes válidos

Los mensajes válidos para este protocolo son:

- *IRIS_NEW_DEVICE*
- *IRIS_DISCONNECTING_DEVICE*

Reglas

Una vez que el dispositivo ha recibido la notificación del nodo que publicó el servicio, se deben cumplir las siguientes reglas:

1. Crear un socket para conectarse como cliente con el servicio recién publicado.

2. Agregar una entrada a la tabla con la dirección IP y el puerto del servicio que se acaba de formar entre ellos.

3. El dispositivo pasa una lista de acciones que conforman el estado de la colaboración al nuevo dispositivo.

Una vez que el dispositivo ha recibido la notificación del protocolo de configuración nula acerca de la eliminación de un servicio se deben llevar a cabo las siguientes reglas:

1. Eliminar al canal recién eliminado de la lista de sockets.

2. Eliminar de la tabla el registro asociado con la dirección IP del dispositivo que se ha desconectado.

3.3.4. Mecanismos de colaboración

Una vez que ya se logró establecer la comunicación entre los nodos de la red y además es posible detectar la entrada y salida de los dispositivos a la red, este mecanismo permite que la colaboración sea posible.

Servicio

El servicio que ofrece este mecanismo es la cooperación entre las diferentes aplicaciones que conforman la red espontánea a través del envío de acciones.

Suposiciones

- Las aplicaciones fueron construidas haciendo uso de **IRIS**.

- Los dispositivos han formado una red espontánea y es posible la transferencia de información entre todos los nodos a través de los mecanismos proporcionados por **IRIS**.

- Los dispositivos cuentan con una tabla que almacena las acciones públicas de la aplicación. Esa tabla está compuesta por campos para almacenar las acción y una referencia al objeto que ejecuta cada acción.

- Dado que debe ser posible implementar este mecanismo en los diferentes dispositivos móviles se supone que las diferentes aplicaciones saben de antemano cuáles son las acciones disponibles, ya que éstas no van a ser resueltas en tiempo de ejecución.

Mensajes válidos

Los mensajes válidos para este mecanismo son:

- *IRIS_COLL_ACTION*

Reglas

Una vez que se ha recibido un mensaje del tipo *IRIS_COLL_ACTION* se deben cumplir las siguientes reglas:

1. Extraer la acción y los argumentos del mensaje.
2. Extraer de la tabla de acciones públicas el objeto que se invoca para que ejecute la acción.
3. Ejecutar la acción.
4. En caso de que no exista en la tabla de acciones el objeto asociado a la acción, entonces debe desecharse el paquete.

Diagrama de flujo

En la figura 3.12 se presenta el diagrama de flujo que corresponde a las reglas anteriormente expuestas.

Ejemplo

En las figuras 3.13 y 3.14 se muestra un ejemplo del mecanismo de colaboración dentro de una aplicación de pizarrón compartido. Primero un conjunto de personas que quieren colaborar se reúnen dentro de una área, como se muestra en la figura 4.1, suponiendo que se estableció la red espontánea con la ayuda de los protocolos antes presentados. Dentro de la IBSS de dispositivos IEEE 802.11b, cada persona que tiene un dispositivo puede colaborar

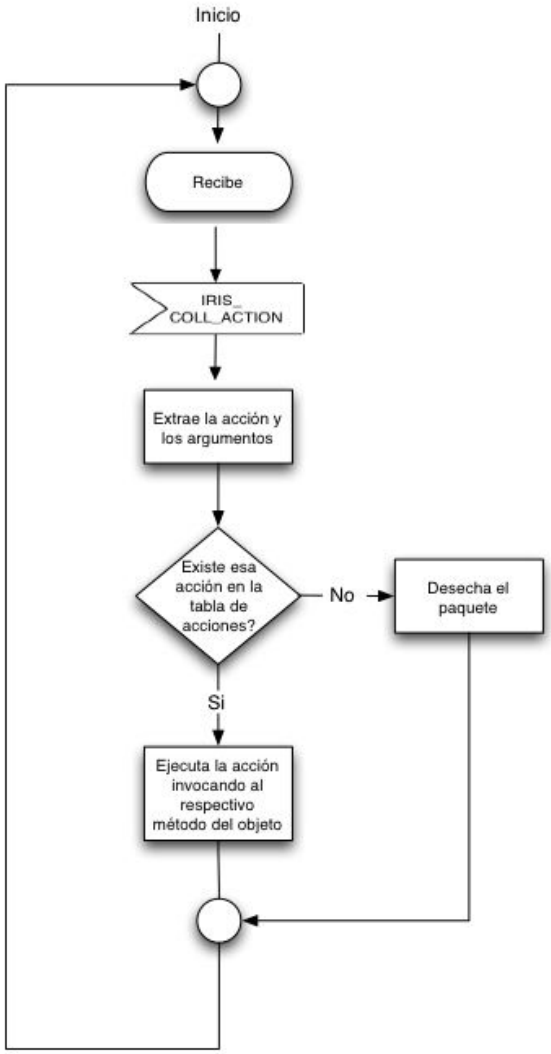

Figura 3.12: Diagrama de flujo del mecanismo de colaboración

haciendo uso del área de dibujo del dispositivo. Cuando alguno de los participantes traza una figura sobre su área de dibujo, se manda un mensaje a todos los demás participantes para que se replique la acción y con esto se logre que cada uno de los participantes visualicen en el área de dibujo del dispositivo el conjunto de figuras. Si por ejemplo, el usuario trazó un círculo de radio 4 en la posición (2,10), esta acción se manda a las demás aplicaciones remo-

tas, éstas analizan el paquete, extraen la acción que se debe llevar a cabo, la ejecutan y por consiguiente el área de dibujo se actualiza. En la figura 3.14 a un lado de cada enlace de comunicación (linea punteada) se encuentra un etiqueta con el texto *circulo: 2,10 r: 4* el cual corresponde a la acción llevada a cabo por el usuario A y la cual fue enviada a los demás dispositivos.

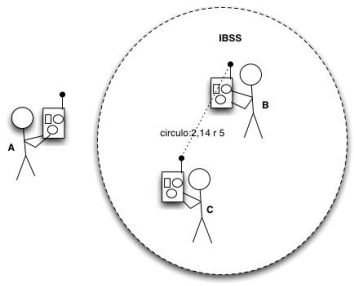

Figura 3.13: Un conjunto de usuarios que quieren colaborar a través de sus dispositivos móviles

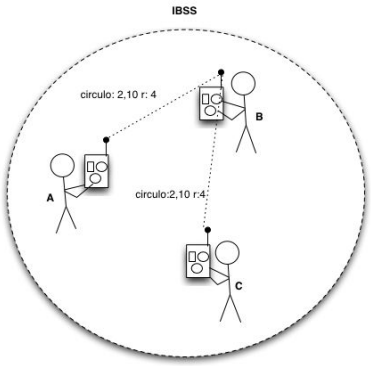

Figura 3.14: Un pizarrón electrónico compartido por varios usuarios

3.4. Resumen

En este capítulo se describieron los elementos de **IRIS** que dan soporte al desarrollo de las aplicaciones de *colaboración digital espontánea* con el fin de fomentar la construcción de este tipo de aplicaciones sobre una variedad

de plataformas que utilicen protocolos estándares de red local inalámbrica IEEE 802.11b.

La variedad de dispositivos móviles y por ende de plataformas sobre las cuales se ejecutarán las aplicaciones de colaboración espontánea es la principal razón para definir una arquitectura por capas que constituye una clara posibilidad de permitir integrar nuevos dispositivos con diferentes plataformas y además que sean fáciles de desarrollar, lo cual se mostrará en el capítulo 5.

Este es un intento temprano de permitir estandarizar la arquitectura de las posteriores aplicaciones de *colaboración digital espontánea* y con esto lograr una fácil integración entre este tipo de aplicaciones. En el siguiente capítulo se discute la implementación de **IRIS**.

4

La implementación de IRIS

En el capítulo anterior introdujimos conceptualmente cada uno de los componentes de la capa intermedia **IRIS**. En este capítulo discutimos la implementación de estos componentes en dos diferentes plataformas con el fin de cubrir un rango de dispositivos móviles, además de presentar los argumentos de la elección de las plataformas y de los ambientes de desarrollo de cada una de ellos.

4.1. Selección de los dispositivos

Los dispositivos digitales móviles como computadoras portátiles, asistentes digitales y teléfonos celulares son fáciles de adquirir, sin embargo, cada uno de estos dispositivos cuenta con diferentes capacidades. Con el fin de proporcionar un escenario de colaboración en el que los dispositivos proporcionen al usuario una interfaz mínima adecuada para un ambiente de colaboración, se eligieron dos tipos de dispositivos móviles que son computadoras portátiles y asistentes digitales con la siguientes características:

- Una computadora portátil iBook con procesador PowerPC G3, 600Mhz, 384 MB de memoria, 20 Gigabytes de disco duro y con tarjeta Airport.
- Una computadora de escritorio de las cuales una es iMac PowerPC G4, 700Mhz, 384 MB de memoria, 40 Gigabytes de disco duro y con tarjeta Airport.
- Una computadora con procesador intel Pentium III, 700Mhz, 128 MB de memoria y 8 Gigabytes de disco duro.

- Una PDA COMPAQ iPAQ 3870 con procesador ARM y una tarjeta CompacFlash IEEE 802.11b de LINKSYS.

Con el fin de dar flexibilidad y robustez a la capa intermedia **IRIS** y pensando en favorecer la simplicidad de su uso se decidió utilizar el paradigma orientado a objetos en las dos plataformas.

4.2. Implementación en Mac OS X

La principal razón que nos impulsó a elegir el sistema operativo Mac OS X es que proporciona:

- La implementación del estándar IEEE 802.11b
- Los protocolos de configuración nula, específicamente la implementación del protocolo de descubrimiento de servicios llamada *Rendezvous*
- Un ambiente de desarrollo muy adecuado para el paradigma orientado a objetos gracias a las herramientas que proporciona
- Una plataforma la cual tiene las bondades de los sistemas UNIX

4.2.1. Ambiente de desarrollo Cocoa

Este ambiente de desarrollo proporciona un conjunto de herramientas, interfaces de programación de aplicaciones (APIs) y clases que facilitan considerablemente la construcción de aplicaciones.

Las aplicaciones más importantes del ambiente de desarrollo Cocoa son [28]:

Project Builder

Es la principal aplicación para el manejo, depuración y mantenimiento de proyectos de software. Maneja todos los archivos y recursos asociados con una aplicación. Además proporciona un ambiente gráfico.

Interface Builder

Es una herramienta para crear prototipos rápidamente. Proporciona un editor de objetos que permite designar las relaciones entre los objetos de

una aplicación de manera gráfica. Incluye una herramienta para crear subclases de objetos existentes. Además de que permite al programador cargar sus propias paletas de objetos.

4.2.2. Componentes de IRIS en Mac OS X

En la figura 4.1 se muestran los componentes de **IRIS** en el primer nivel se encuentra el API de *Rendezvous* que es la implementación de Apple del protocolo de descubrimiento de servicios. Las dos principales clases *NSNetServices* y *NSNetServicesBrowser* proporcionan la base para publicar y descubrir servicios. En el mismo nivel se encuentra el **Apple802.11 Framework** un conjunto de funciones para detección y creación de redes 802.11b en Mac OS X.

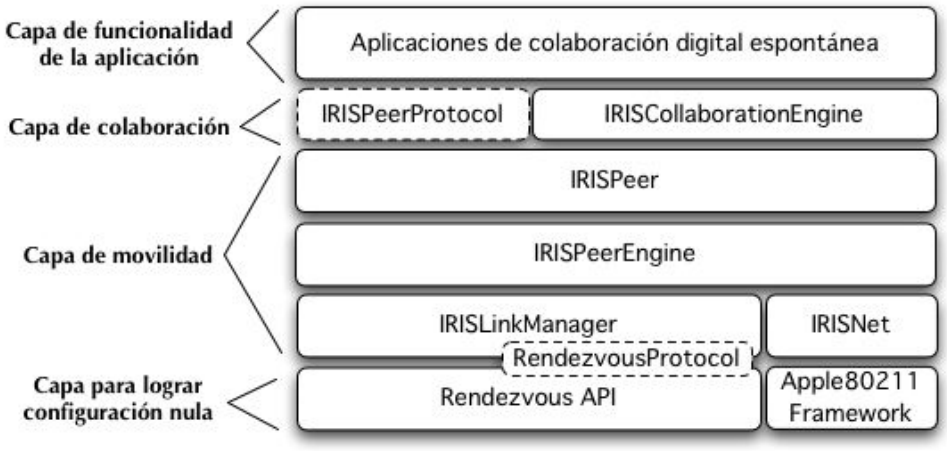

Figura 4.1: Principales componentes de la implementación en Mac OS X

4.2.3. Descripción de clases y protocolos

- La clase *IRISPeer* es la clase más importante para el desarrollador ya que es la clase que él utilizará. Tiene métodos para publicar servicios, remover un servicio, encontrar un servicio, conectarse a la red, desconectarse, mandar mensajes, administrar los sockets y realizar determinadas acciones cuando llega un nuevo dispositivo. Por supuesto, toda la funcionalidad es consecuencia de la composición de las otras clases

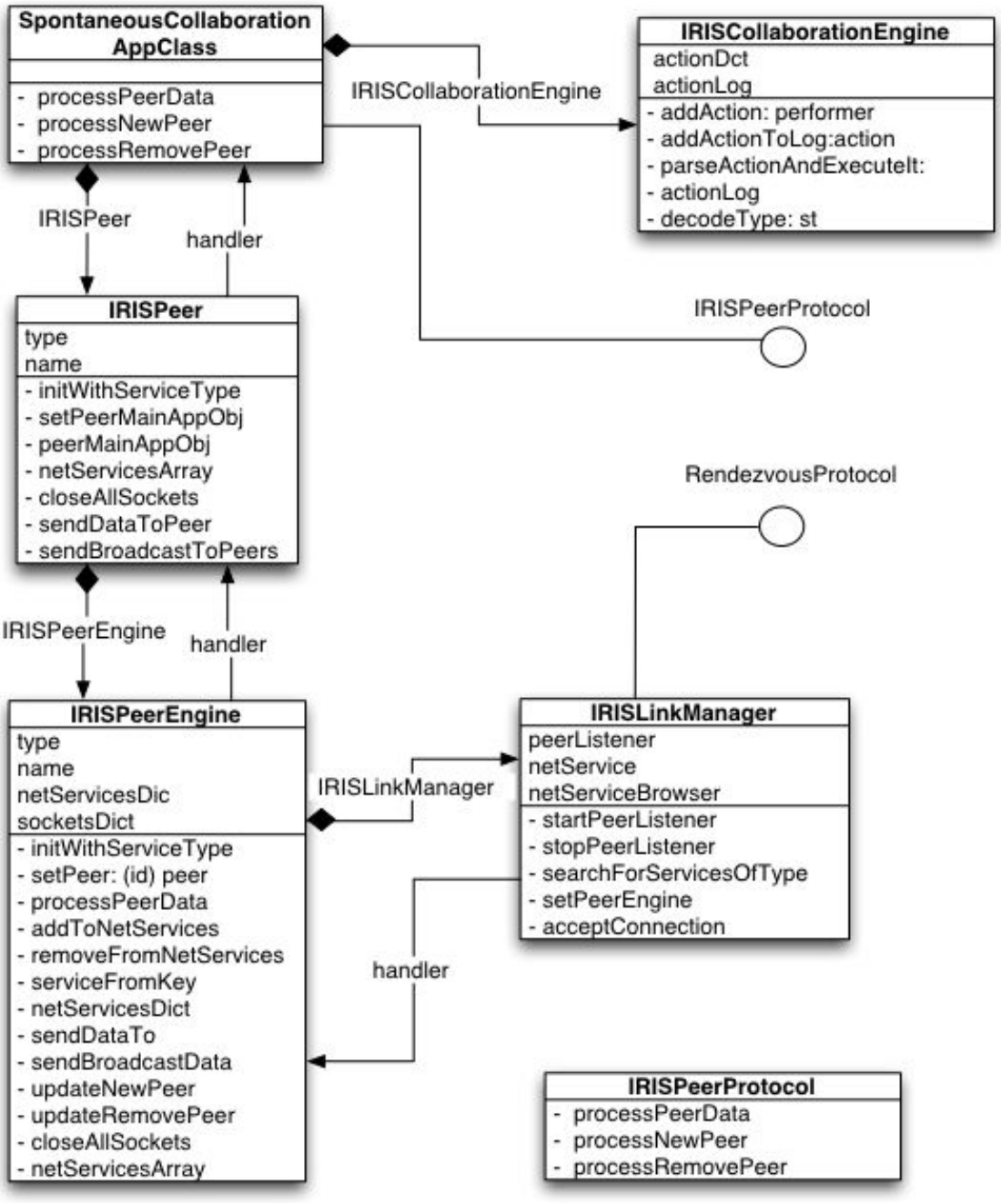

Figura 4.2: Diagrama UML de la implementación de IRIS

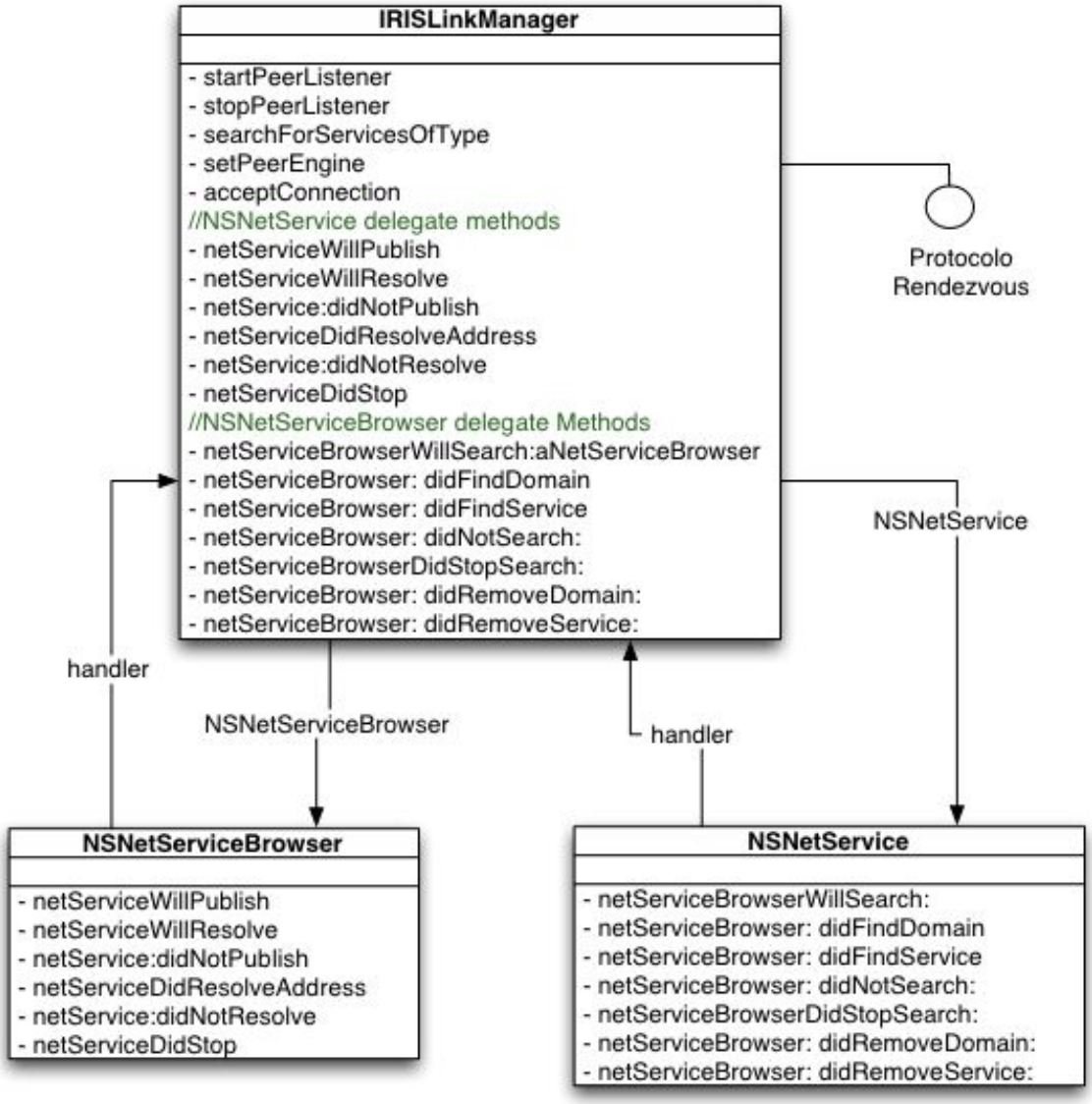

Figura 4.3: Continuación del diagrama UML de la implementación de IRIS

en **IRIS**. Las aplicaciones crearán una instancia de ésta para poder interactuar con la red, definir y utilizar servicios.

- `initWithServiceType: name:`

  Este método se ejecuta para iniciar el objeto y sus variables de instancia que definen el nombre y tipo del servicio, además de crear un objeto de la clase *IRISPeerEngine*.

- `setPeerMainAppObj:`

  Este método se invocará en la aplicación de colaboración digital espontánea como argumento se le pasa el objeto que controla la aplicación de colaboración digital espontánea que se almacena en una variable de instancia de esta clase.

- `peerMainAppObj`

  Este método regresa el objeto que controla la aplicación de colaboración digital espontánea.

- `netServicesArray`

  Este método regresa un arreglo de cadenas que corresponden a los servicios que están disponibles en la red.

- `closeAllSockets`

  Este método se invoca cuando termina la aplicación para cerrar todos los sockets que se utilizaron para la comunicación.

- `sendDataToPeer:name:`

  Este método se encarga de enviar un conjunto de datos contenidos en un objeto el cual se pasa como argumento en el método al miembro del grupo de colaboración que se especifica como argumento.

- `sendBroadcastToPeers:`

  Este método se utiliza para mandar a todos los miembros del grupo de colaboración un conjunto de datos contenidos en un objeto el cual se pasa como argumento en el método.

- `dealloc`

  Este método se llama cuando el objeto se libera y se encarga de liberar la memoria asignada a las variables de instancia del objeto.

- La clase *IRISCollaborationEngine* se encarga de implementar el mecanismo de colaboración manteniendo almacenadas las acciones y un apuntador al objeto que la debe ejecutar, además de almacenar una bitácora de las acciones ejecutadas en la colaboración con el fin de poder enviarla a los nuevos participantes.

  - `init`

    Este método se ejecuta para iniciar el objeto y sus variables de instancia que almacenan una tabla con las acciones a través de las que se realiza la colaboración y los objetos que las ejecutan y una bitácora de las acciones realizadas durante la colaboración.

  - `addAction: performer:`

    Con este método el programador de la aplicación de colaboración digital espontánea adiciona las acciones y los objetos que responden a ellas.

  - `addActionToLog:`

    Este método se ejecuta cuando se realiza una acción en el dispositivo donde se ejecuta la aplicación o cuando llega un mensaje que contiene una acción realizada en algún otro dispositivo con el que se está colaborando.

  - `parseActionAndExecuteIt:`

    Este método se encarga de comprobar la validez de la acción contenida en un mensaje contra la tabla de acciones.

  - `actionLog`

    Este método regresa la bitácora de las acciones realizadas durante la colaboración.

  - `decodeType:`

    Este método define el tipo de los datos que conforman los parámetros del método invocada dentro de un mensaje enviado desde otro dispositivo

  - `dealloc`

    Este método se llama cuando el objeto se libera y se encarga de liberar la memoria asignada a las variables de instancia del objeto.

- El protocolo *IRISPeerProtocol* es la declaración de los métodos que se deben implementar en la clase que utiliza la infraestructura **IRIS** para construir aplicaciones de colaboración espontánea.

  - `processPeerData: data`

    Este método lo debe implementar la clase que controla la aplicación de colaboración digital espontánea y definir las acciones que se ejecutarán al recibir un cierto mensaje. Por ejemplo, en un pizarrón el mensaje que llega desde algún participante es `circulo: 3: 4` el objeto de la clase *IRISPeer* manda este mensaje a el objeto que controla la aplicación de colaboración digital espontánea y éste ejecuta la acción de dibujar un círculo con centro (3,4) en el pizarrón.

  - `processNewPeer: peerName`

    Este método lo debe implementar la clase que controla la aplicación de colaboración digital espontánea y define el comportamiento de la aplicación cuando un nuevo participante se une al grupo de colaboración. Por ejemplo, en este método se puede actualizar la tabla donde se despliega la lista de participantes.

  - `processRemovePeer`

    Este método lo debe implementar el objeto que controla la aplicación de colaboración digital espontánea y define el comportamiento de la aplicación cuando un participante se sale del grupo de colaboración. Por ejemplo, aquí se puede actualizar la tabla donde se despliega la lista de participantes para remover al participante que salió.

- La clase *IRISPeerEngine* es la encargada de utilizar el protocolo de descubrimiento de servicios *Rendezvous* y crear el socket a través del cual se le comunica las acciones de colaboración que se llevaron acabo en otro dispositivo.

  - `initWithServiceType: name:`

    Este método se ejecuta para iniciar el objeto, definir el nombre y tipo del servicio como parámetros, además crea un objeto de la clase *IRISLinkManager*. A este objeto se le manda un mensaje para que inicie la búsqueda de los servicios que tengan el mismo

tipo que el definido como parámetro y se activa y publique el servicio.

- `setPeer:`

  Con este método se le manda el apuntador al objeto de la clase *IRISPeer* para posteriormente avisarle por los eventos de llegada y salida de los servicios.

- `addToNetServices:`

  Este método lo invoca el objeto de la clase *IRISLinkManager* para avisar que llegó un servicio y que se tiene que almacenar en un arreglo con el nombre y la información necesario para conectarse a esté.

- `removeFromNetServices:`

  Este método lo invoca el objeto de la clase *IRISLinkManager* para avisar que salió un servicio de la red y que se tiene que remover del arreglo que almacena los servicios activos en la red.

- `processPeerData:`

  Este método será invocado cada vez que exista información de entrada en algún socket.

- `netServicesDict`

  Este método regresa un diccionario con todos los servicios activos en la red y su correspondiente información contenida en objeto.

- `serviceFromKey:`

  Este método regresa la información del servicio que se especifica como parámetro.

- `sendDataTo: data:`

  Este método se encarga de enviar un conjunto de datos al servicio de la red el cual se pasa como argumento.

- `sendBroadcastData: data`

  Este método se utiliza para mandar un conjunto de datos a todos los servicios de la red que tiene el mismo tipo que nuestro servicio.

- `netServicesArray`

  Regresa el nombre de los servicios activos en la red en un arreglo.

- `updateNewPeer:`

  Este método se invoca cuando un servicio anuncia su entrada a la red. Este método es el encargado de crear un socket cliente y conectarse al servicio que se acaba de anunciar.

- `updateRemovePeer:`

  Este método se invoca cuando un dispositivo deja de participar en la colaboración y se encarga de liberar la memoria que almacena la información concerniente a este dispositivo.

- `closeAllSockets`

  Este método cierra todos los socket que se utilizaron para la comunicación con los servicios activos en la red.

- `dealloc`

  Este método se llama cuando el objeto se libera y se encarga de liberar la memoria asignada a las variables de instancia del objeto.

- La clase *IRISLinkManager* contiene el contexto de colaboración es decir almacena el conjunto de socket que son el canal de comunicación con cada uno de los dispositivos miembros del grupo de colaboración, por lo tanto esta clase es la encargada de enviar la información de colaboración a través de los sockets. Esta clase es usada por *IRISPeer* para administrar todos los servicios que están en la red, se encarga del contexto de colaboración.

  - `init`

    Este método se ejecuta para iniciar el objeto. Además se crea un objeto de la clase *NSNetServiceBrowser* y se establece como delegado de este objeto, con el fin de ser notificado cuando un servicio publica su entrada o su salida de la red.

  - `startPeerListenerType: name:`

    Este método activa el servicio con el nombre y el tipo que se pasan como parámetros. Además se publica utilizando el protocolo de *Rendezvous*.

  - `stopPeerListener`

    Este método detiene el servicio iniciado con el método `startPeer ListenerType: name:`. Además se publica la salida del servicio de la red utilizando el protocolo de *Rendezvous*.

- `searchForServicesOfType:`

  Este método le solicita al objeto de la clase *NSNetServiceBrowser* que se creó en el método init que busque en la red por todos los servicios del tipo que se pasa como parámetro. Entonces, se activa un proceso concurrente que se mantiene activo durante la ejecución de la aplicación y cuando llega un servicio del tipo especificado manda una notificación a este objeto.

- `setPeerEngine:`

  Este método recibe como parámetro el objeto de la clase *IRISPeerEngine* por que este procesará toda la información que se recibe de los diferentes sockets y es necesario poder invocarlo.

- `acceptConnection:`

  Este método se encarga de aceptar las conexiones con los clientes del servicio publicado.

- `dealloc`

  Este método se llama cuando el objeto se libera y se encarga de liberar la memoria asignada a las variables de instancia del objeto.

4.3. Implementación en Familiar Linux

El asistente digital iPAQ tiene pre-instalado *Microsoft's Windows* para Pocket PC, conocido como WindowsCE. No obstante *Linux embedded* nos permite hacer uso de sockets, proporcionando al programador un alto grado de libertad en el manejo a bajo nivel de los recursos del sistema, a diferencia del sistema operativo WindowsCE. Por tal razón se decidió instalar *Linux embedded* en la iPAQ(figura 4.4). Otra ventaja es que los componentes que conforman un sistema Linux en general (figura 4.5) son también los que conforman un sistema *Linux embedded* [29]. Familiar es una distribución de *Linux embedded* para asistentes digitales Compaq iPAQ.

4.3.1. Ambiente de desarrollo Qt/Embedded C++

Es una herramienta de desarrollo de aplicaciones e interfaces de usuario para dispositivos empotrados. Además de que funciona sobre una gran variedad de procesadores, es posible usar una versión de Qt desktop para la fase de desarrollo.

Figura 4.4: El asistente digital con Linux

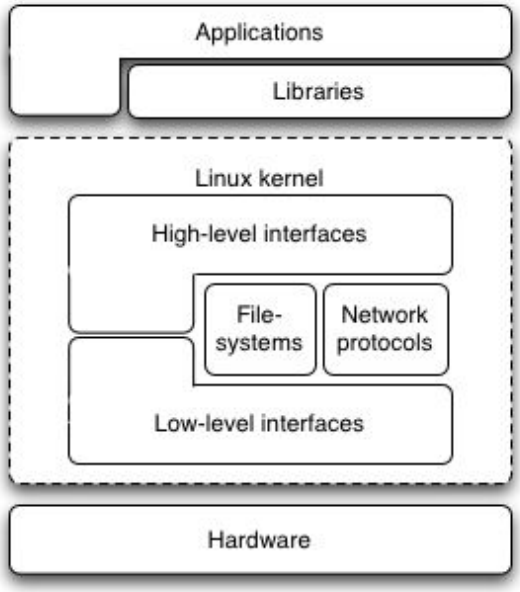

Figura 4.5: Arquitectura de un sistema Linux

- Proporciona el API estándar de **QT**
- Es completamente Orientado a Objetos
- Puede compilarse quitando características para minimizar el uso de memoria
- Proporciona su propio sistema de ventanas que es más compacto que **Xlib**
- Está disponible para todos los procesadores que soportan Linux, por ejemplo, Intel x86, MIPS, ARM, StrongARM, Motorola 68000 y Power PC
- Además proporciona muchos componentes no gráficos como: internacionalización, redes e interacción con base de datos

El sistema de ventanas de **Qt** consiste de uno o más procesos de los cuales uno actúa como servidor que guarda las regiones a ser desplegadas por los clientes y genera eventos del ratón y el teclado usando memoria compartida para comunicarse con los clientes. Las aplicaciones pueden ser compiladas sobre cualquier plataforma equipada con una herramienta *crooss-development chain*.

La opción más común es construir un compilador *crooss-platform* GNU de C++ (g++) con **libc** y utilerías binarias sobre un sistema Unix.

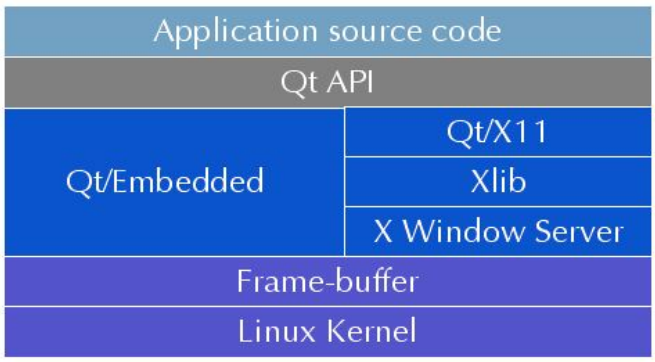

Figura 4.6: Arquitectura de Qt/Embedded

4.3.2. Componentes de IRIS en Familiar Linux

En la figura 4.7 se muestra los componentes que constituyen la implementación de **IRIS** en Familiar. En el primer nivel se encuentran las bibliotecas que dan soporte a las clases construidas para **IRIS**:

- **libnet**

  Proporciona un API de funciones de red de bajo nivel comúnmente usadas, principalmente para mandar paquetes a la red.

- **libpcap**

  Proporciona una interfaz de alto nivel para capturar paquetes en la red. Todos los paquetes en la red, aunque estos estén destinados a otro nodo, son accesibles a través de este mecanismo.

- **iwlib**

  Llamada también *Wireless Tools* es un conjunto de herramientas para manipular el API *Wireless Extensions* el cual permite al usuario configurar su manejador de la interfaz inalámbrica de manera indistinta para la variedad de manejadores soportados para Linux. Otra ventaja es que estos parámetros pueden cambiarse sobre la marcha sin iniciar de nuevo el manejador.

A partir del segundo nivel se encuentran las clases que construimos para implementar **IRIS**. En la implementación en Familiar, las clases *IRISPeer*, *IRISPeerProtocol*, *IRISPeerEngine*, *IRISLinkManager* y *IRISCollaborationEngine*, cumplen con la misma funcionalidad que las clases con esos nombres en la implementación en Mac OS X. Las clases restantes conforman la capa de configuración nula.

La clase *IRISScanningNet* se encarga de detectar todas las redes que se encuentran activas en el área. Para crear esta clase es necesario contar con las *wireless extensión* y las librería **iwlib** por lo que se aplicaron 2 parches al kernel:

- El parche para actualizar el *wireless extension* de 12 a 14 es iw_handless.w14-4.diff

- El parche para el manejo de la tarjeta de red inalámbrica es ori.11b.we14-2.mos.diff

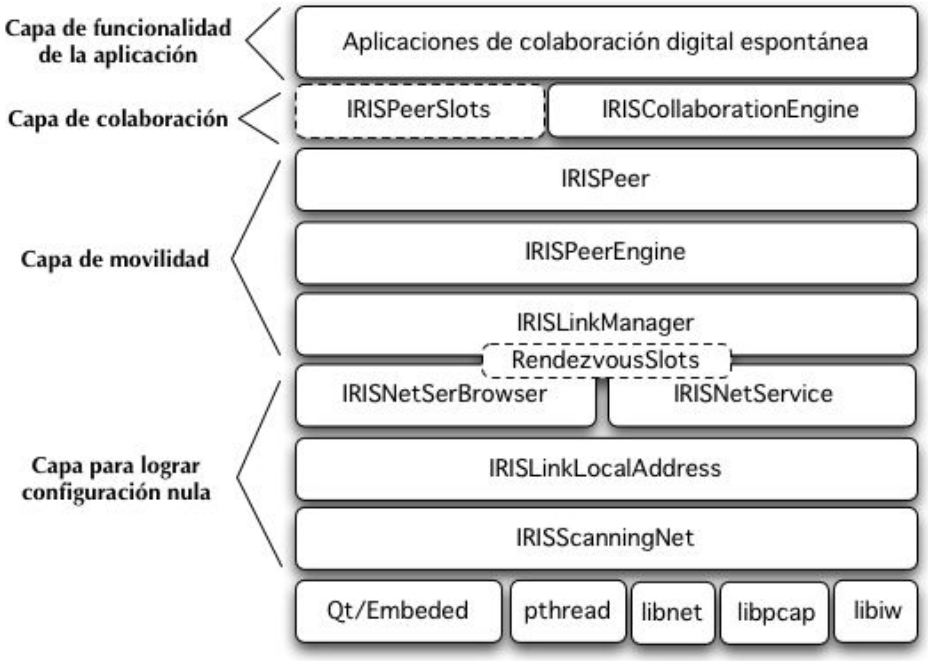

Figura 4.7: Principales componentes de la implementación en Familiar Linux

- `search`

  Este método ejecuta un proceso de búsqueda, con el fin de detectar las redes inalámbricas activas en el área.

- `connect`

  Este método inicia el proceso de unión a la red especificada como parámetro de entrada.

- `create`

  Este método sirve para crear una nueva red inalámbrica.

- `setIP`

  Este método llama al objeto de la clase *IRISLinkLocalAddress* para obtener la dirección IP a través del protocolo de autoasignación de dirección.

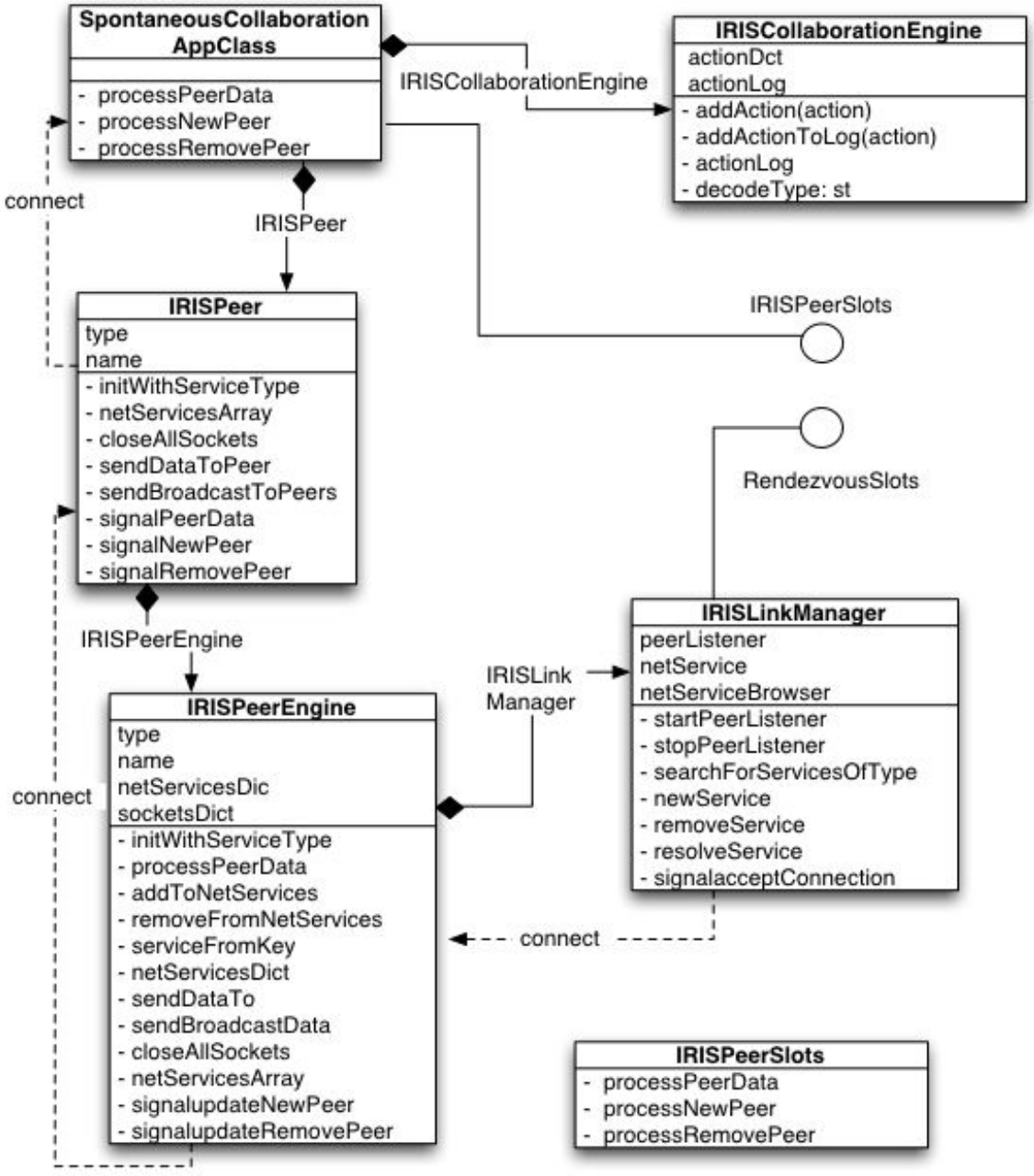

Figura 4.8: Diagrama UML de la implementación de IRIS

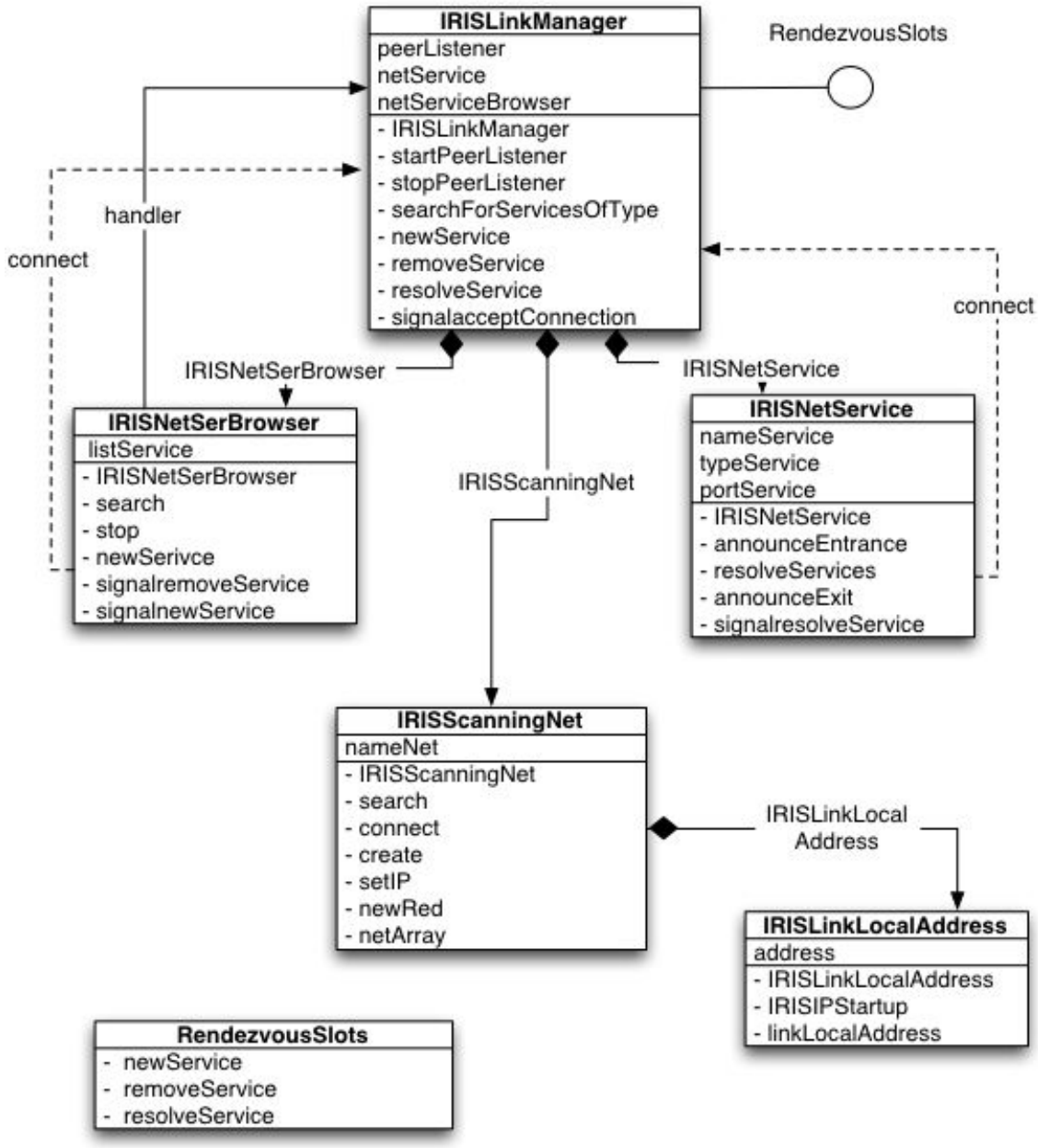

Figura 4.9: Continuación del diagrama UML de la implementación de IRIS

- `newRed`

  Está es una señal que se envía al objeto de la clase *IRISPeer* para avisar que se encontró una red inalámbrica activa en el área.

La clase *IRISLinkLocalAddress* es la encargada ejecutar el protocolo de auto-asignamiento de IP. En linux no está implementado el protocolo de auto-asignamiento de IP. Sin embargo ya existe un primer intento para implementar direccionamiento de enlace local sobre plataformas Linux, se llama zcip que tomamos como base en la creación de la clase *IRISLinkLocalAddress*.

Para implementar el protocolo de auto-asignamiento de IP se necesita:

- Poder construir los paquetes ARP
- Poder enviar paquetes ARP a la red
- Poder escuchar los paquetes ARP que trafican en la red

Para construir y enviar paquetes ARP se utilizó la biblioteca **libnet** y para poder escuchar los paquetes ARP se utilizó la biblioteca **libpcap**.

Por la naturaleza de los procesadores sobre los que corre Familiar se necesitó realizar modificaciones a algunos parámetros definidos en el protocolo:

- Se cambiaron los tiempos de prueba y de envío de los paquetes de ARP, debido a la naturaleza de los dispositivos
- Se instalaron y compilaron las librerías **libnet** y **libpcap** para el procesador ARM.
- Se compiló la implementación ya modificada con un compilador cruzado para el procesador ARM

- `IRISIPStartup`

  Este método inicia un proceso concurrente desde donde se debe ejecutar el método `linkLocalAddress`.

- `linkLocalAddress`

  Este método se invoca desde un proceso concurrente para obtener la dirección IP para la interfaz de red inalámbrica usando el protocolo de auto-asignación de IP para después continuar defendiendo la dirección.

La clase *IRISNetSerBrowser* es la implementación del protocolo de descubrimiento de servicios *Rendezvous* en C++. Esta clase se define una interfaz para encontrar servicios publicados sobre una red usando *multicast* DNS. Un objeto de esta clase ejecuta todas las búsquedas asíncronamente usando un pthread para ejecutar la búsqueda en proceso concurrente. El resultado de la búsqueda se manda a través de un señal. El proceso concurrente que corre con la menor prioridad sigue buscando hasta que el método `stop` se ejecuta.

- `IRISNetSerBrowser`

  Este método se ejecuta para iniciar el objeto definiendo el tipo del servicio que se buscará.

- `search`

  Esté método inicia un proceso concurrente que realiza la búsqueda de nuevo servicios.

- `stop`

  Esté método detiene la búsqueda de nuevos servicios.

- `newService`

  Esta señal avisa al objeto de la clase *IRISPeerEngine* que un nuevo servicio entro a la red.

- `removeService`

  Esta señal avisa al objeto de la clase *IRISPeerEngine* que un servicio salió de la red.

La clase *IRISNetService* representa un servicio de red que las aplicaciones publican o usan como un cliente. Antes de que el objeto *IRISNetService* pueda publicar el servicio la clase *IRISLinkManager* se debe encargar de configurar el servicio:

1. Creando un socket que escuche por peticiones de servicio, al cual los clientes se van a conectar.

2. Iniciar un objeto *IRISNetService* con el dominio, el tipo, el nombre y el puerto del servicio.

- `IRISNetService`

  Este método se ejecuta para iniciar el objeto y sus variables de instancia.

- `resolveServices`

  Este método hace uso del protocolo de descubrimiento de servicios basado en mDNS para obtener la información para conectarse con el servicio especificado como parámetro para conectarse a él.

- `announceEntrance`

  Este método hace uso del protocolo de descubrimiento de servicios basado en mDNS para anunciar la entrada del servicio que activó la aplicación.

- `announceExit`

  Este método hace uso del protocolo de descubrimiento de servicios basado en mDNS para anunciar la salida del servicio que activó la aplicación.

4.4. Resumen

En este capítulo se revisaron las implementaciones de **IRIS** en Mac OS X y Familiar Linux. Las dos implementaciones son orientadas a objetos y hace uso de protocolos para conectar a los diferentes niveles. El diseño ha mostrado ser flexible y permite la sustitución de clases en una capa siempre y cuando éstas adopten el protocolo de la capa. En la implementación en Familiar Linux se construyeron clases adicionales en comparación con la implementación en Mac OS X debido a la carencia de los protocolos de auto-asignación de dirección IP y de descubrimiento de servicios. En el siguiente capítulo se muestra como las implementaciones son útiles para desarrollar aplicaciones de colaboración digital espontánea.

5

Construyendo una aplicación con IRIS

En este capítulo, discutimos la aplicación que se diseño e implemento con la capa intermedia de software IRIS con el fin de mostrar que nuestra infraestructura de software puede ser utilizada para la construcción de aplicaciones de colaboración espontánea y además que con el uso de la capa intermedia IRIS el tiempo y el esfuerzo del proceso de diseño e implementación disminuye considerablemente.

5.1. Pizarrón compartido

5.1.1. Descripción de la aplicación

El objetivo principal es permitir la creación y modificación de un dibujo con la colaboración de varios usuarios a través de sus dispositivos. La aplicación fue útil para probar la formación de la red, la comunicación inalámbrica y el descubrimiento de servicios. La aplicación fue desarrollada en Mac OS X y Familiar. Los dispositivos debe estar equipados con tarjetas de red IEEE 802.11b. Las aplicaciones tanto en Mac OS X y Familiar tienen una interfaz gráfica (figuras 5.1 y 5.2) con los siguientes objetos:

- Una área gráfica para mostrar el dibujo
- Un conjunto de botones que definen la figura que se dibujará en el área
- Un botón para iniciar la búsqueda de grupo de colaboración

- Un botón para conectarse a un grupo de colaboración
- Un botón para crear un nuevo grupo de colaboración
- Un campo de texto para definir el nombre de un nuevo grupo de colaboración
- Un panel en el que se listan los grupos (las redes) de colaboración que existen en el área
- Un panel en el que se listan los diferentes dispositivos que se encuentran colaborando en el grupo al que se conecta el usuario

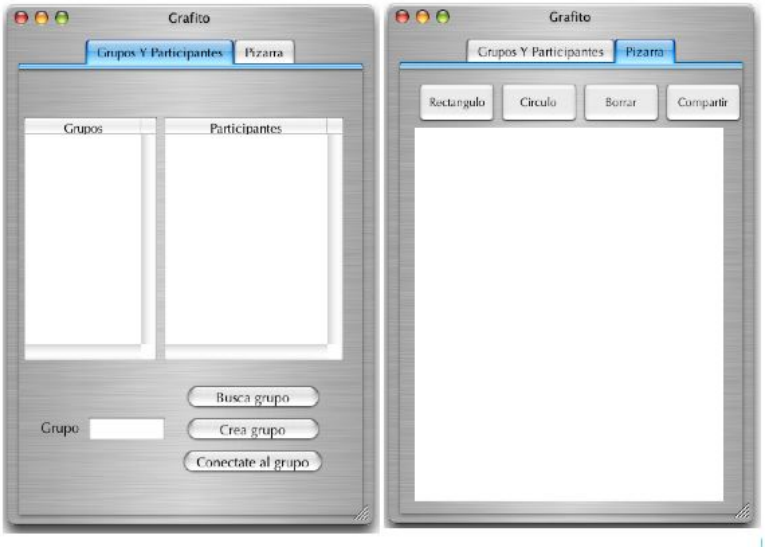

Figura 5.1: La interfaz de usuario de la aplicacón en Mac OS X

Un ejemplo del funcionamiento de la aplicación es como sigue: Una persona con una computadora portátil ejecuta la aplicación del pizarrón compartido. Al momento de ejecutar la aplicación se lleva a cabo un proceso de inicialización. En el panel de grupos de colaboración no se despliega ninguno. Entonces el usuario define el nombre y aprieta el botón para crear un grupo de colaboración. Una persona con un asistente digital entra al recinto, ejecuta la aplicación y aparece en el panel de grupos el creado por el usuario de la computadora portátil. El usuario selecciona el grupo y activa el botón de

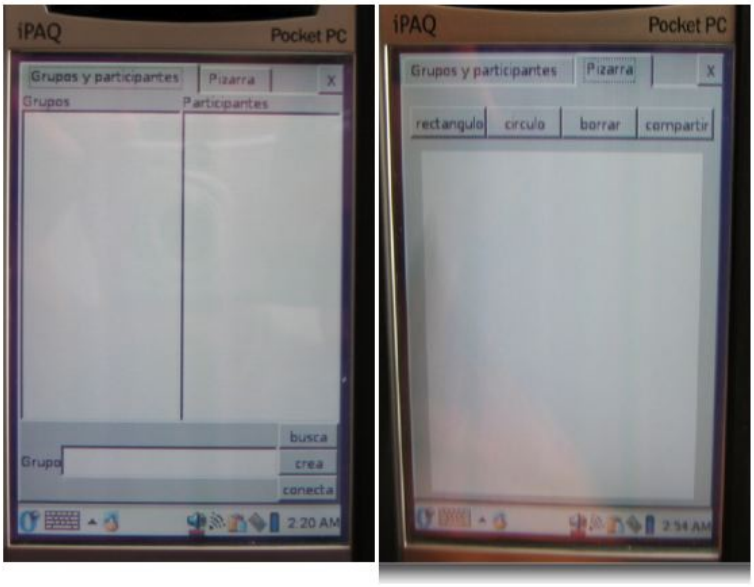

Figura 5.2: La interfaz de usuario de la aplicacón en Familiar linux

conexión. La aplicación muestra en otro panel la existencia de la computadora portátil. El usuario selecciona la computadora portátil, inmediatamente se establece una conexión entre los diferentes dispositivos, las aplicaciones habilitan el área gráfica. Los usuarios comienzan a dibujar y visualizar las acciones sobre el área gráfica (figuras 5.3) .

5.1.2. Diseño de la aplicación

La aplicación usa el patrón de diseño MVC(Model View Controller). Se construyeron dos clases *MyAppController* y la clase *MyAppView*.

- El objeto de la clase *MyAppController* controla la aplicación y es el puente de comunicación con el middleware IRIS instanciando un objeto de la clase *IRISPeer* y un objeto de la clase *IRISCollaborationEngine* como se ve en el siguiente código:

```
/* En Objective-C para Mac OS X */
#import <IRIS/IRISPeer.h>
#import <IRIS/IRISPeerProtocol.h>
```

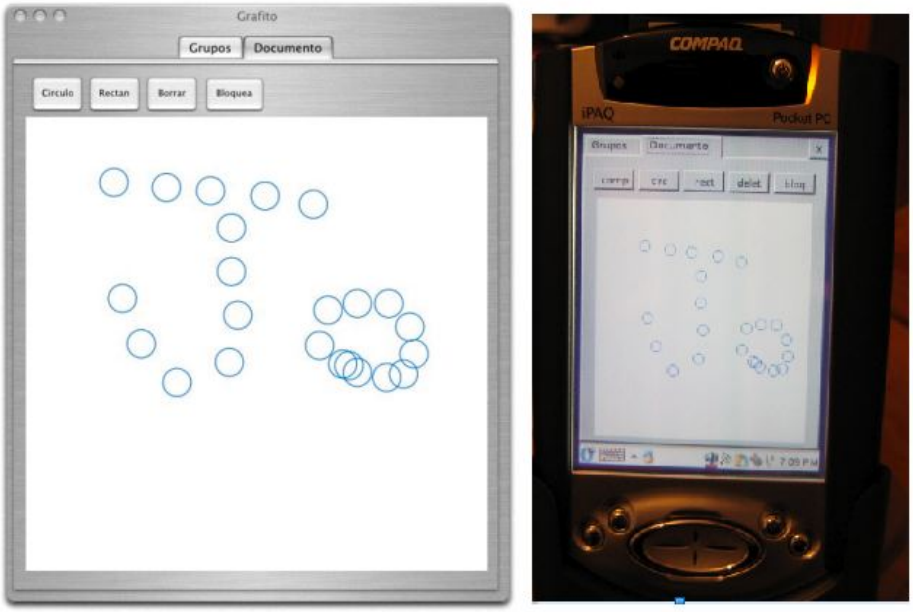

Figura 5.3: Intercambiando figuras geométricas entre dispositivos móviles

```
#import <IRIS/IRISCollaborationEngine.h>

@interface MyAppController : NSObject  <IRISPeerProtocol>
{
    IRISPeer * _peer;
    IRISCollaborationEngine *engine;
    IBOutlet NSView *grafito;
    IBOutlet NSTableView *tableView;
    IBOutlet NSTableView *tableViewG;
    IBOutlet NSTextField *textField;
    NSString *name;
 }


/* En Qt para Familiar Linux */
#include "MyAppView.h"
#include "IRISPeer.h"
```

```
#include "IRISCollaborationEngine.h"

class MyAppController : public QWidget{

  Q_OBJECT

  private:
IRISPeer *peer;
IRISCollaborationEngine *engine;
        MyAppView *grafito;
QListView *listView;
QListView *listViewG;

};
```

Lo primero que se necesita es iniciliazar los objetos de las clases *IRISPeer* y *IRISCollaborationEngine*. Al inicializar el objeto de la clase *IRISPeer* se pasan como argumentos el nombre y tipo del servicio que son la identificación del participante y el tipo de colaboración respectivamente. Es importante asignar el apuntador del objeto que controla la aplicación a la variable de instancia *_peerMainAppObj* de la clase *IRISPeer* ya que es el enlace de comunicación entre el middleware IRIS y la aplicación.

```
 /* En Objective-C para Mac OS X */
    _peer = [IRISPeer alloc];
    [_peer  initWithServiceType: @"_pizarra._tcp." name: name];
    [_peer setPeerMainAppObj: self];
     engine = [[IRISCollaborationEngine alloc] init];

/* En Qt para Familiar Linux */
    peer = new IRISPeerEngine("_pizarra._tcp.",name);
    connect( peer , SIGNAL(NewPeer(QString)),
      SLOT(processNewPeer(QString)));
    connect( peer , SIGNAL(PeerData(QString)),
     SLOT(processPeerData(QString)));
    connect( peer , SIGNAL(RemovePeer(QString)),
```

```
    SLOT(processRemovePeer(QString)));
    engine = new IRISCollaborationEngine();
```

Adicionamos las acciones que son validas en la colaboración y el objeto que las ejecuta, en este caso el objeto grafito es de la clase *MyAppView*.

```
/* En Objective-C para Mac OS X */
  [engine addAction: @"addCircle::" performer: grafito];
  [engine addAction: @"addRect::" performer: grafito];
  [engine addAction: @"deleteFigure::" performer: grafito];

 /* En Qt para Familiar Linux */
  engine->addAction("addCircle");
  engine->addAction("addRect");
  engine->addAction("deleteFigure");
```

Y lo más importante es implementar el protocolo IRISPeer. Con el método `processNewPeer:` se define las acciones que se realizan cuando un nuevo participante se integra al grupo de colaboración.

Con el método `processPeerData:` se define las acciones que se realizan cuando un participante envía una mensaje al grupo de colaboración.

Con el método `processRemovePeer:` se define las acciones que realiza la aplicación cuando un participante se separa del grupo de colaboración.

- El objeto de la clase *MyAppView* es el encargado de desplegar las figuras en el área de dibujo, por lo tanto es necesario que implemente los métodos definidos como acciones validas para la colaboración al inicializar el objeto de la clase *MyAppController*

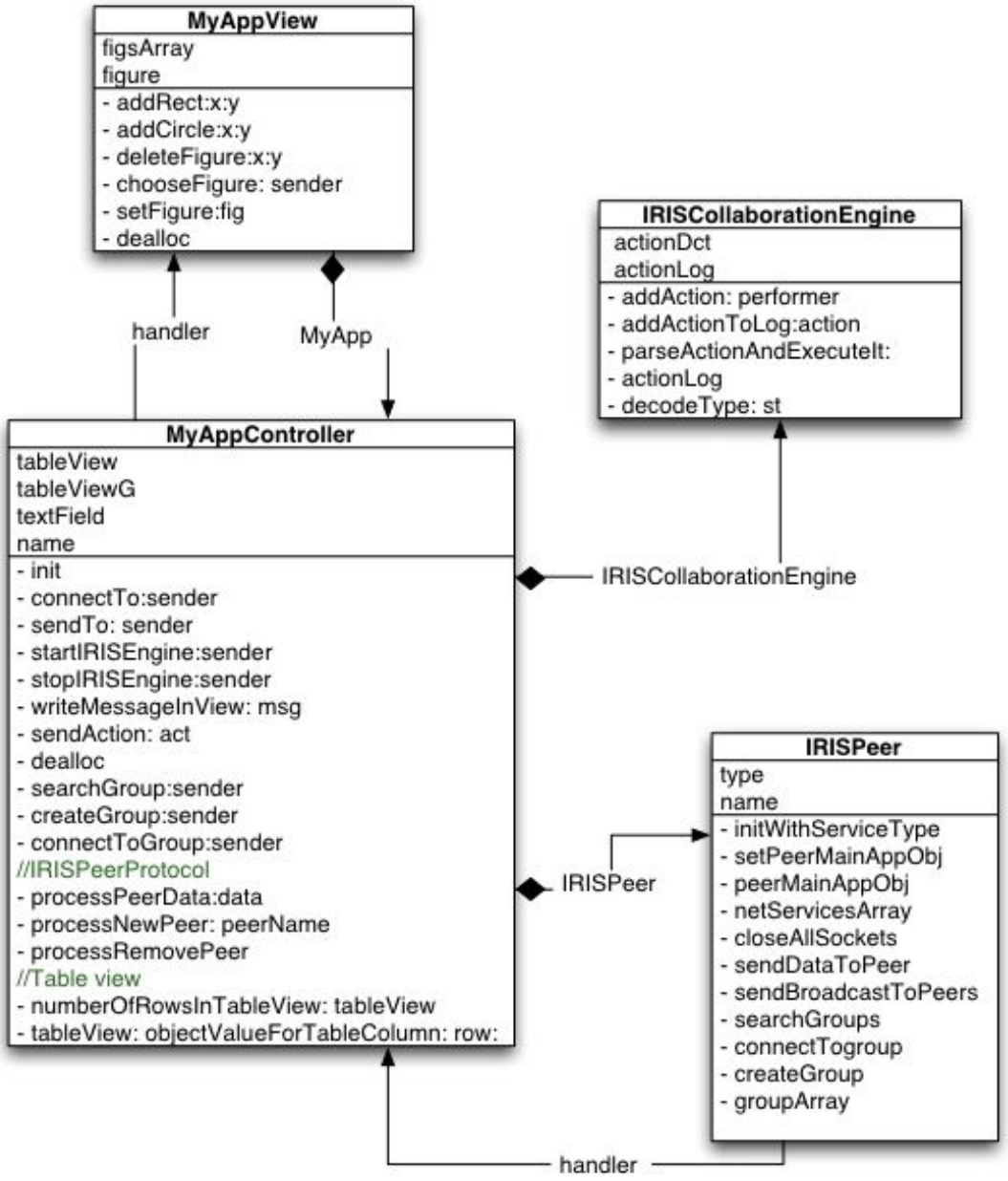

Figura 5.4: Diagrama UML de la aplicación

5.1.3. Características importantes de la aplicación

Los aspectos importantes que hay que considerar de la aplicación desarrollada:

- En ningún momento el usuario realiza alguna configuración para la formación de la red
- Los servicios no son centralizados
- Las aplicaciones en los diferentes dispositivos se comportan como servidores y clientes
- La comunicación es inalámbrica permitiendo al usuario tener más movilidad, aunque dentro de determinados límites

El hecho de que los servicios no sean centralizados es un concepto importante en el sentido de que la programación de las aplicaciones difiere de las aplicaciones tradicionales basadas en el modelo cliente/servidor. En el modelo cliente/servidor un determinado dispositivo de cómputo (servidor) es el encargado de proporcionar el servicio y de coordinar las comunicaciones, el cliente sólo es capaz de usar el servicio siempre y cuando el servidor esté disponible. El cliente, además, cuenta con mecanismos como un DNS (servicios proporcionados por otros servidores) para poder obtener información para localizar al servicio. Para este tipo de redes inalámbricas las aplicaciones tienen que ser distribuidas y definitivamente su comportamiento debe ser de servidor y cliente. De esta manera es más fácil tratar con el dinamismo de la red.

Las aplicaciones desarrolladas funcionan para que los dispositivos de forma automática y continua verifiquen la existencia de un servicio, y en caso de que exista se conecten a él.

5.2. Otras aplicaciones construidas con IRIS

5.2.1. Aplicación de intercambio de mensajes

La aplicación de intercambio de mensajes es muy sencilla, el objetivo principal es permitir el intercambio de mensajes de texto entre varios usuarios a través de sus dispositivos. La aplicación fue útil para probar la formación

de la red, la comunicación inalámbrica y el descubrimiento de servicios. La aplicación fue desarrollada en Mac OS X y Familiar. Los dispositivos deben estar equipados con tarjetas de red IEEE 802.11b.

Las aplicaciones tanto en Mac OS X y Familiar tienen una interfaz gráfica con los siguientes objetos:

- Un campo de texto para escribir los mensajes a enviar.
- Un campo de texto para mostrar los mensajes recibidos o enviados.
- Un botón que sirve para enviar el mensaje escrito en el campo de texto.
- Un botón para iniciar la búsqueda de grupo de colaboración
- Un botón para conectarse a un grupo de colaboración
- Un botón para crear un nuevo grupo de colaboración
- Un campo de texto para definir el nombre de un nuevo grupo de colaboración
- Un panel en el que se listan los grupos (las redes) de colaboración que existen en el área
- Un panel en el que se listan los diferentes dispositivos que se encuentran colaborando en el grupo al que se conecta el usuario y en donde el usuario pueda seleccionar el dispositivo al cual desea enviar el mensaje escrito en el campo de texto.

Un ejemplo del funcionamiento de la aplicación es como sigue: Una persona con una computadora portátil ejecuta la aplicación de intercambio de mensajes. Al momento de ejecutar la aplicación se lleva a cabo un proceso de inicialización. En el panel de grupos de colaboración no se despliega ninguno. Entonces el usuario define el nombre y aprieta el botón para crear un grupo de colaboración. Una persona con un asistente digital entra al recinto, ejecuta la aplicación y aparece en el panel de grupos el creado por el usuario de la computadora portátil. El usuario selecciona el grupo y activa el botón de conexión. La aplicación muestra en otro panel la existencia de la computadora portátil. El usuario selecciona la computadora portátil, inmediatamente se establece una conexión entre los diferentes dispositivos, las aplicaciones habilitan los campos de texto para la escritura de mensajes, y se actualiza

la lista de dispositivos que estén disponibles para el envío de mensajes. Los usuarios comienzan a enviar y a recibir mensajes.

Cabe destacar que es posible que el usuario de la computadora portátil se desplace dentro del recinto y ello no implica que se rompa la liga de comunicación, a menos que se salga del rango de comunicación.

5.2.2. Un control remoto para la aplicación iTunes

La flexibilidad de la arquitectura de IRIS, permite desarrollar aplicaciones que involucran controlar de manera remota aplicaciones de otro dispositivo que se encuentran dentro del rango de comunicación. El objetivo de esta aplicación era que un asistente digital pudiera controlar de forma remota la aplicación *iTunes*, la cual reproduce música con formato MP3. Con esta aplicación el usuario puede controlar de manera remota el volumén de la aplicación o decidir que canción se debe reproducir.

Un ejemplo del funcionamiento de la aplicación es como sigue: Al momento de ejecutar la aplicación se lleva a cabo un proceso de inicialización. En el panel de grupos de colaboración no se despliega ninguno. Entonces el usuario define el nombre y aprieta el botón para crear un grupo de colaboración. Una persona con un asistente digital entra al recinto, ejecuta la aplicación y aparece en el panel de grupos el creado por el usuario de la computadora portátil. Una vez hecho esto se presentará en un panel todos los dispositivos que actualmente tienen el servicio disponible. Selecciona alguno de la lista, se establece la conexión con el dispositivo y puede comenzar a enviar algún comando que seleccionará del menú que tiene en la parte superior del asistente digital. Finalmente si escoge *Start* o *Random* se escogerá alguna canción de manera aleatoria de la lista actual del dispositivo y se comenzará a reproducir.

6

Conclusiones y trabajo a futuro

En esta tesis se ha introducido el concepto de aplicaciones de colaboración digital espontánea y se ha presentado el diseño e implementación de una infraestructura de software que da soporte en el diseño, construcción, ejecución y evolución de las aplicaciones de colaboración espontánea. En este capítulo, resumimos nuestro trabajo y describimos brevemente las posibles extensiones a este proyecto que merecen atención como trabajo a futuro.

6.1. Resumen de la investigación

- Los objetivos planteados fueron alcanzados. Se cuenta con una infraestructura de software **IRIS** que trabaja sobre dispositivos móviles heterogéneos con tecnología inalámbrica IEEE 802.11b que hace posible el desarrollo de escenarios de colaboración digital espontánea.

- La infraestructura de software **IRIS** está formada por un conjunto de protocolos y mecanismos de colaboración para ambientes móviles que efectúan: la detección, asociación o creación de un grupo de trabajo, la entrada y salida de los participantes y la difusión de las acciones de cada participante al grupo de colaboración.

- La infraestructura de software **IRIS** utiliza el estándar IEEE 802.11b para redes locales inalámbricas y los protocolos de configuración nula para lograr la formación de una red sin infraestructura entre varios dispositivos móviles.

- El protocolo de colaboración permite la interoperación entre aplicaciones distribuidas en dispositivos heterogéneos. Está orientado al envío de acciones y ha mostrado ser un modelo flexible y poderoso.

- La implementación de la infraestructura de software **IRIS** en los sistemas operativos Mac OS X y Familiar Linux, cubre un gran rango de dispositivos, desde computadoras de escritorio hasta asistentes digitales.

- Las aplicaciones desarrolladas con las implementaciones de la infraestructura de software **IRIS** en los sistemas operativos Mac OS X y Familiar Linux muestran su funcionalidad para el desarrollo de aplicaciones de colaboración digital espontánea.

6.2. Conclusiones

- La creciente evolución en Hardware que sin duda a rebasado por mucho al software, se debe a la integración a gran escala. En materia de software no se ha logrado crear componentes que sean tan fáciles de integrar en todos los proyectos que impulsen la evolución del software. Este tema merece un estudio minucioso, sin embargo una respuesta a crear una evolución es sin duda la creación de capas intermedias de software que sean completamente re-usables e intercambiables.

- Las aplicaciones en escenarios de colaboración digital espontánea necesitan una nueva abstracción y son muy diferentes a las tradicionales aplicaciones de red basadas en el modelo cliente/servidor. El cambio radica, principalmente, en que estas redes ya no pueden depender de servidores centrales, ya que se forman de manera dinámica y no se puede presuponer la existencia de algún servicio y ni siquiera de que éste va a estar disponible durante todo el tiempo de vida de la red. Además, deben de ser capaces de tratar con la entrada y salida de nodos, algo que no ocurre en las redes tradicionales alambradas.

- La tecnología de red inalámbrica no es tan sólo el reemplazo del cables por ondas de radio o luz infrarroja, es una oportunidad de crear nuevos esquemas que permiten a los usuarios colaborar a través de los dispositivos portátiles personales de manera espontánea.

- Es posible utilizar la tecnología IEEE 802.11b para la generación de redes espontáneas, con la gran ventaja de que son redes basadas en TCP/IP, con las cuáles ya se está familiarizado. Los protocolos de autoasignación de IP y descubrimiento de servicios para configuración nula efectivamente evitan que los usuarios se preocupen por la configuración de la red.

- El principal uso que se le da a los dispositivos IEEE 802.11b es en su modo de infraestructura BSS. Con la creación de aplicaciones de colaboración digital espontáneas se puede explotar la funcionalidad de su modo IBSS, ampliando la funcionalidad que los dispositivos WiFi le proporcionan al usuario.

- El desarrollo de escenarios de colaboración entre dispositivos heterogéneos presenta retos tecnológicos debido a la gran variedad de plataformas disponibles.

- El proyecto **IRIS** representan un punto de inicio para futuras investigaciones en el campo de computación móvil en México.

6.3. Perspectivas a Futuro

Los proyectos más ambiciosos no son sólo aquellos que comienzan con grandes metas, si no también aquellos que surgen de un proyecto que amplia la visión del equipo que lo realizó y se imponen como una perspectiva a futuro. Este proyecto no sólo nos proporcionó los resultados esperados y las respuestas a las preguntas que al inicio de éste nos hicimos, sino que también nos deja grandes retos que son:

- La integración de esquemas de seguridad en la infraestructura de software **IRIS**.

- La integración de algoritmos de reducción de acciones en la bitácora de colaboración.

- Implementación de la infraestructura de software **IRIS** en otros dispositivos móviles como Palm con sistema operativo Palm OS.

- Implementación de diferentes aplicaciones colaborativas, por ejemplo, aplicaciones de video y audio.

- La unión de las capas intermedias de software **IRIS** y **Lucrn** para la creación de aplicaciones de colaboración digital en redes que interoperen con tecnologías inalámbricas Blluetooth y WiFi.
- La definición de una metodología para integrar la capa **IRIS** con las antiguas aplicaciones de colaboración digital con el fin de re-usar estas aplicaciones para escenarios de colaboración espontánea.

Apéndice A

Rendezvous en Mac OS X

A.1. NSNetServiceBrowser

La clase *NSNetServiceBrowser* es una implementación del protocolo de descubrimiento de servicios Rendezvous. La clase *NSNetServiceBrowser* define una intefaz para encontrar servicios publicados sobre una red usando multicast DNS. Esta clase depende fuertemente de un delegado, el cual es el único medio de alertar a una aplicación del descubrimiento de un servicio. *NSNetServiceBrowser* sirve para dos propósitos: buscar por un dominio de red o buscar un servicio de red anunciado en un dominio dado. Cada objeto *NSNetServiceBrowser* ejecuta una búsqueda a un tiempo, así si se quiere ejecutar búsquedas múltiples simultáneas, se usan múltiples objetos *NSNetServiceBrowser*. Un objeto *NSNetServiceBrowser* ejecuta todas las búsquedas asíncronamente. El resultado de la búsqueda es regresado a través del objeto delegado asociado, el cual la aplicación cliente debe proporcionar. El proceso en forma oculta sigue buscando hasta que el método `stop` se ejecuta.

Métodos para el manejo del delegado

- `delegate`

  Regresa un apuntador al objeto delegado.

- `setDelegate:`

  Asigna al objeto como delegado.

Métodos de solicitud de búsquedas

- `searchForAllDomains`

  Inicia una búsqueda por todos los dominio que son visibles al nodo.

- `searchForRegistrationDomains`

  Inicia una búsqueda por los dominios en los cuáles el host puede registrar servicios.

- `searchForServicesOfType: type inDomain: domain`

  Inicia una búsqueda por servicios del tipo `type` y dentro de un especifico dominio.

- `stop`

  Detiene la actual búsqueda y manda el mensaje `netServiceBrowserDid StopSearch:` al delegado, además descarta cualquier búsqueda pendiente.

Métodos que un delegado tiene que implementar

Búsqueda

- `netServiceBrowser:didNotSearch:`

  Es enviado siempre cuando un error impide que ocurra una búsqueda. Se pueden usar los parámetros para saber cual fue el error.

- `netServiceBrowserDidStopSearch:`

  Con éste se informa al delegado que la búsqueda fue detenida con el mensaje stop.

- `netServiceBrowserWillSearch:`

  Indica que una búsqueda está comenzando. El delegado puede usar esta notificación para preparar sus estructuras de datos para recibir datos.

Respuestas a búsquedas de dominios

- `netServiceBrowser:didFindDomain:moreComing:`

  Este método es enviado cada vez que se encuentra un dominio. Se puede generar con éste una lista de dominios encontrado, cuando `moreComing` es igual a NO, indica que no existen por el momento más dominios.

- `netServiceBrowser:didRemoveDomain:moreComing:`

  Este método es enviado cada vez que un dominio desaparece o no estará disponible. Se puede remover en éste de la lista de dominios encontrado, con el parámetro `moreComing` en NO, indica que no existen por el momento más dominios a remover.

Respuestas a búsquedas de servicios

- `netServiceBrowser:didFindService:moreComing:`

  Este método se invoca cada vez que un servicio es encontrado. Y el objeto del servicio se encuentra en el argumento de entrada que se puede usar para conectarse y hacer uso del servicio, además se puede generar con éste la lista de servicios encontrados. Con el parámetro `moreComing` en NO se indica que no existen por el momento más servicios encontrados.

- `netServiceBrowser:didRemoveService:moreComing:`

  Este método se invoca cada vez que un servicio desaparece o no estarán disponible. Y el objeto del servicio se encuentra en el argumento de entrada que se puede usar para desconectarse del servicio que desapareció, además se puede usar para remover estos servicios de la lista de servicios que están disponible en la red. Con el parámetro `moreComing` en NO se indica que no existen por el momento más servicios para remover.

A.2. NSNetService

La clase *NSNetService* representa un servicio de red que las aplicaciones publican o usan como un cliente. Para configurar un *NSNetService*, se debe hacer lo siguiente:

1. Configurar un socket que escuche por peticiones de servicio, al cual los clientes se van a conectar.

2. Iniciar un objeto *NSNetService* con el dominio, el tipo, el nombre y el puerto del servicio.

3. Asignar un objeto delegado a esta instancia.

4. Manejar cualquier mensaje recibido por el delegado.

Antes de publicar el servicio, tu debes primero crear un socket por el cual los clientes puedan conectarse para acceder al servicio. Además se debe proporcionar un objeto delegado para responder a los mensajes y manejar apropiadamente los errores.

Métodos

- `initWithDomain:type:name:port:`

  Inicia un servicio. El argumento `domain` es el dominio en el cual el servicio está registrado. En Mac OS X sólo se tiene registro en el dominio local es .local. El argumento `type` especifica el tipo de servicio y el protocolo de transporte -*_servicetype._tcp*. La autoridad que asigna los números de internet (IANA Intermodal Association of North America) tiene un catálogo de nombre de servicios y una lista de puertos donde las aplicaciones pueden encontrar estos servicios. El tercer argumento `name` es el nombre del servicio y como es el que lo identifica en la red debe ser único. El último argumento `port` es el puerto del servicio.

Manejo del delegado

- `delegate`

  Regresa un apuntador al objeto delegado

- `setDelegate:`

  Asigna al objeto como delegado.

Obteniendo información del servicio.

- `addresses`

  Regresa un arreglo de estructuras sockaddr que contienen la información para conectarte a el socket, pueden ser varias si el servicio es *multihoming*.

- `domain`

  Regresa el nombre del dominio del servicio en un objeto *NSString*.

- `name`

  Regresa el nombre del servicio en un objeto *NSString*.

- `type`

  Regresa el tipo del servicio en un objeto *NSString*.

Manejo del servicio

- `publish`

  Este método difunde la información del servicio en la red.

- `resolve`

  Con éste método se verifica que el servicio está disponible.

- `stop` Detiene la publicación o resolución de un servicio en proceso. Este método mando un mensaje a su delegado que es `netServiceDidStop`.

Métodos que un delegado tiene que implementar

Publicación

- `netService:didNotPublish:`

  Notifica al delegado que el servicio no puede ser publicado.

- `netServiceWillPublish:`

  Notifica al delegado que la red esta lista para publicar el servicio.

Resolviendo servicio

- `netService:didNotResolve:`

  Informa al delegado que un error ocurrió durante la resolución de un servicio.

- `netServiceDidResolveAddress:`

  Informa al delegado que la dirección fue resuelta. El delegado puede usar el método `addresses` para obtenerla.

- `netServiceWillResolve:`

  Notifica al delegado que la red esta lista para resolver el servicio.

Deteniendo servicio

- `netServiceDidStop:`

  Informa al delegado que una publicación o resolución del servicio se detuvo.

Constantes de Error

NSNetServicesUnknownError

Un error desconocido ocurrió.

NSNetServicesCollisionError

El servicio no puede ser publicado porque el nombre ya está siendo usado.

NSNetServicesNotFoundError

El servicio no puede ser encontrado en la red.

NSNetServicesActivityInProgress

El servicio de red no puede procesar la solicitud por el momento.

NSNetServicesBadArgumentError

Un argumento invalido fue usado cuando se creo el objeto *NSNetService*.

NSNetServicesCancelledError

El cliente canceló la acción.

NSNetServicesInvalidError

El servicio de red fue configurado inapropiadamente.

Bibliografía

[1] M. Weiser. The computer for the 21st century. *Scientific American*, September 1991.

[2] Henri ter Hofte. *Working Apart Together*. Telematica Instituut, 1998.

[3] Bengt Ahlgren Laura Marie Feeney and Assar Westerlund. Spontaneous networking: An application-oriented approach to ad-hoc networking. *IEEE Communications Magazine*, June 2001.

[4] Laura Marie Feeney. A taxonomy for routing protocols in mobile ad-hoc networks. *SICS tech rep T99:07*, 2001.

[5] Aidan Williams Kwan-Wu Chin, John Judge and Roger Kermode. Implementation experience with manet routing protocols. *ACM SIGCOMM Computer Communications Review*, November 2002.

[6] Tao Lin. A framework for mobile ad hoc routing protocols. *Proc. IEEE 2003 Wireless Comm. and Networking Conference*, 2003.

[7] E.Royer and C.-K Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Pers. Commun.*, April 1999.

[8] J. Chicharo M. Abolhasan, J. Lipman. A routing strategy for heterogeneous mobile ad hoc networks. *IEEE 6th CAS Symposium on Emerging Technologies: Frontiers of Mobile and Wireless Communications (MWC)*, 2004.

[9] Panagiotis Papadimitratos and Zygmunt J. Haas. Secure routing for mobile ad hoc networks. *Proceedings of SCS Communication Networks and Distributed Systems a Modeling and Simulation Conference*, 2002.

[10] Stephan Preuß and Clemens Cap. *Overview of Spontaneous Networking - Evolving Concepts and Technologies*, volume 24. Fachbereich Informatik der Universität Rostock, 2000.

[11] Sumi Helal. Standards for service discovery and delivery. *IEEE Pervasive computing*, September 2002.

[12] Jorge Alfonso Briones García. *Lucrn: Un middleware para el desarrollo de aplicaciones en redes espontáneas de dispositivos móviles Bluetooth.* CINVESTAV, 2004.

[13] Erik Guttman. *Autoconfiguration for IP Networking: Enabling Local Communication.* IEEE Intenet Computing, May 2001.

[14] E.Gregori R.Bruno, M.Conti. *WLAN technologies for Mobile ad hoc Networks.* Proceedings of the 34th Hawaii International Conference on System Sciences, Septembe 2001.

[15] Steve J.Vaughan-Nichols. *Bull Market for IEEE 802.11 WLAN Chipsets.* IEEE Computer Magazine, November 2002.

[16] Matthew S Gast. *802.11 Wireless Networks The Definitive Guide.* O'REILLY, 2002.

[17] Jennifer J.-N. Liu Imrich Chlamtac, Marco Conti. Mobile ad hoc networking: imperatives and challenges. *Elsevier Ad Hoc Networks*, 2003.

[18] Secretaría de Comunicaciones y Transportes. Proyecto de norma oficial mexicana proy-nom-121-sct1-2001, telecomunicaciones-radiocomunicacion-sistemas de radiocomunicacion que emplean la tecnica de espectro disperso. *Diario Oficial de la Federación. Viernes 8 de Febrero de 2002.*

[19] S. F. Midkiff T. Lin and J. S. Park. Approximation algorithms for minimal connected dominating sets and application with routing protocol in wireless ad hoc network. *Proc. of IEEE International Performance Computing and Communications Conference (IPCCC)*, 2003.

[20] S.C.M. Woo and S. Singh. *Longest life routing protocol (LLRP) for ad hoc networks with highly mobile nodes.* In Proceedings of IEEE WCNC, 2000.

[21] E. M. Royer and C. Perkins. *An implementation study of the AODV routing protocol*. Proceedings of the IEEE Wireless Communications and Networking Conference, 2000.

[22] Dimitri Bertsekas and Robert Gallager. *Data Networks*. Prentice Hall, 1992.

[23] IETF Zero Configuration Networking Working Group. *http://www.zeroconf.org/*.

[24] Gerard J. Holzmann. *Design and Validation of computer protocols*. Prentice Hall, 1991.

[25] Bernad Aboba Stuart Cheshire and Erik Guttman. *Dynamic Configuration of IPv4 Link-Local Addresses (draft 17)*. http://www.ietf.org/ietf/1id-abstracts.txt, July 2004.

[26] Stuart Cheshire. *Performing DNS queries via IP Multicast*. http://www.ietf.org/ietf/1id-abstracts.txt, August 2004.

[27] Craig Hunt. *TCP/IP Network Administration*. O'REILLY, 2001.

[28] Michel Beam and James Duncan Davidson. *Cocoa in a nutshell*. O'REILLY, 2003.

[29] Karim Yaghmour. *Building Embedded Linux Systems*. O'REILLY, 2003.

Los abajo firmantes, integrantes del jurado para el examen de grado que sustentará la Lic. **Silvana Bravo Hernández**, declaramos que hemos revisado la tesis titulada:

IRIS: Una infraestructura de software para el desarrollo de escenarios de colaboración digital espontánea sobre dispositivos móviles IEEE 802.11b

Y consideramos que cumple con los requisitos para obtener el Grado de Maestría en Ciencias en la especialidad de Ingeniería Electrica opción Computación.
Atentamente,

Doctor Guillermo Benito Morales Luna

Doctor Francisco Rodríguez Henríquez

Doctor José Oscar Olmedo Aguirre