ARQUITECTURA ORIENTADA A COMUNIDADES VIRTUALES COLABORTIVAS SOBRE DISPOSITIVOS MÓVILES: AYLLU

Autores:

Oskar Javier Cantor Flórez
Leonardo Enrique Mancilla Amaya

PROYECTO DE GRADO PRESENTADO PARA OPTAR AL TÍTULO DE INGENIERO DE SISTEMAS

Director:
Ingeniero Enrique González, Ph.D.

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
CARRERA DE INGENIERIA DE SISTEMAS
BOGOTA D.C.
JUNIO 2005

PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE INGENIERIA

CARRERA DE INGENIERIA DE SISTEMAS

Rector Magnífico:

Padre Gerardo Remolina Vargas S.J.

Decano Académico Facultad de Ingeniería:

Ingeniero Francisco Javier Rebolledo Muñoz

Decano del Medio Universitario Facultad de Ingeniería:

Padre José Sarmiento Nova S.J.

Director Carrera de Ingeniería de Sistemas:

Ingeniera Hilda Cristina Chaparro López

Director Departamento de Ingeniería de Sistemas:

Ingeniero Germán Alberto Chavarro Flórez

NOTA DE ACEPTACIÓN

_____

_____

_____

_____

_____

Ing. Enrique González Guerrero Ph.D.
Director del proyecto

_____

Ing. César Julio Bustacara Medina MSc
Jurado

_____

Ing. Julián Ernesto Rodríguez
Jurado

JUNIO 2005

Artículo 23 de la Resolución No. 1 de Junio de 1946

"La Universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado.

Sólo velará porque no se publique nada contrario al dogma y la moral católica y porque no contengan ataques o polémicas puramente personales. Antes bien, que se vean en ellos el anhelo de buscar la verdad y la Justicia"

Queremos dedicar este trabajo a nuestros padres, quienes hicieron un gran esfuerzo para que nosotros pudiéramos culminar esta etapa de nuestras vidas, alentándonos a ser mejores cada día y a trabajar por nuestros sueños.

También dedicamos este trabajo a Dios, que nos iluminó y nos dio la fortaleza para levantarnos cada día y perseguir esta meta en los buenos y malos momentos.

Oskar Javier Cantor Flórez
Leonardo Enrique Mancilla Amaya

AGRADECIMIENTOS

Queremos agradecer muy especialmente a nuestro director de tesis el Ingeniero Enrique González Ph.D., quien nos guió durante todo este proceso, fomentó en nosotros el espíritu investigativo y nos enseñó muchas cosas, las cuales estamos seguros que aplicaremos a lo largo de nuestra vida profesional. También queremos agradecer a la universidad y a todos los profesores que dejaron huella en nosotros. Agradecemos también muy especialmente a todas las personas que nos apoyaron y nos animaron con sus palabras a seguir trabajando por esta meta, que hoy terminamos con mucha satisfacción.

Gracias.

Oskar Javier Cantor Flórez
Leonardo Enrique Mancilla Amaya

CONTENIDO

TABLA DE FIGURAS

LISTA DE TABLAS

LISTA DE ANEXOS

Todos los anexos se encuentran en el CD del proyecto, y son los siguientes:

Anexo1:

Comparación de herramientas de desarrollo de aplicaciones para dispositivos móviles y agentes de software en /Anexo1/HerramientasDesarrollo.pdf.

Anexo 2:

Diagramas de clases y paquetes de la Arquitectura Ayllu en /Anexo2/Diagramas.

Manual del programador:

Manual del programador para la arquitectura Ayllu.

0. INTRODUCCIÓN

En cualquier tipo de negocio, la necesidad de cooperar es una constante, limitada por los avances tecnológicos. Las aplicaciones de groupware, CSCW (Computer Supported Cooperative Work), surgen como una solución a este inconveniente, proporcionando mecanismos que permiten comunicar personas entre sí, incrementar la productividad, reducir costos, crear grupos basados en intereses comunes, agilizar la labor de toma de decisiones, negociar o competir sin tener en cuenta el contexto individual de las personas como el tiempo o la ubicación geográfica.

Como consecuencia de toda la funcionalidad que prestan las aplicaciones CSCW, su diseño e implementación son mucho más complejos que los del software mono-usuario, debido a las consideraciones que se deben tener, en especial, la comprensión del comportamiento de las personas en un ambiente grupal.

Tradicionalmente, el desarrollo de las aplicaciones de groupware ha sido orientado a dispositivos tradicionales de cómputo pero, con la popularización de distintos dispositivos móviles y fijos como las PDA, los celulares, smartphones, computadoras portátiles, desktop y tecnologías como Bluetooth, o Wi-Fi que eliminaron los cables permitiendo la movilidad, se pueden extender las capacidades que actualmente brinda el groupware y aprovechar todas estas nuevas tecnologías.

Sin embargo, existen limitaciones para aplicar el groupware en estos dispositivos. Los mayores problemas se presentan en la intermitencia de la señal de las tecnologías inalámbricas y las limitaciones inherentes a los dispositivos móviles como la poca capacidad de procesamiento, poco espacio de almacenamiento y poca duración de la batería.

Actualmente existen varias plataformas para el desarrollo de aplicaciones móviles. Una de ellas es MAD (Mobility Supported by Agent-Based Devices) [BA2004]; MAD permite desarrollar aplicaciones móviles de una manera rápida y eficiente, basándose en el concepto de redes P2P, conviviendo con las limitaciones de los dispositivos móviles mencionadas anteriormente. MAD fue construida empleando agentes de software, proporcionados por la plataforma BESA (Behavior-oriented, Event-driven and Social-based Agent Framework) [GE2003], debido a la importancia que han adquirido los agentes de software para el desarrollo de nuevas tecnologías.

Los agentes de software son entidades virtuales, que poseen habilidades sociales, capacidad de cooperación, comunicación y un conjunto de objetivos bien definidos; al obtener una serie de percepciones del entorno en el que se encuentran, y teniendo en cuenta sus recursos y conocimientos, seleccionan la acción más adecuada para cumplir sus objetivos buscando alcanzar el mejor desempeño [FJ1999]. Los agentes de software poseen la flexibilidad necesaria para ser empleados en áreas tales como el entretenimiento (juegos de computador), la educación virtual, el manejo de información, e-business, robótica entre otras; su campo de acción es bastante amplio y pueden ser empleados prácticamente en cualquier campo de la informática.

Estas características fueron la inspiración para la visión de cooperación entre personas, la respuesta a la necesidad de simplificar el proceso de desarrollo de groupware y soportar la movilidad de la arquitectura Ayllu.

Ayllu hace un aporte importante en el área del desarrollo de groupware, orientándolo con el enfoque del *paradigma de las 5C,* que se define como un modelo de cooperación entre personas y soportando los servicios cooperativos en el concepto de *grupos volátiles* empleando agentes de software. El paradigma de las 5C consta de los siguientes elementos: *Cooperación, Colaboración, Coordinación, Resolución de Conflictos* y *Comunicación* y buscan integrar a las personas como parte de un SMA (Sistema Multi-Agente), puesto que las personas se pueden comportar como un agente más del sistema.

Bajo esta perspectiva, Ayllu no solo proporciona un framework de desarrollo, sino también una nueva visión, con la cual se puede abordar el tema de desarrollo de groupware ofreciendo nuevas características y ventajas.

Este documento se desarrolla de la siguiente manera. En la primera sección titulada *Estudio de arquitecturas para el entorno móvil*, se realiza una amplia investigación sobre arquitecturas orientadas al groupware, comunidades virtuales colaborativas y que sean mediadas por agentes de software para lograr la cooperación, con lo que se formará el marco teórico para diseñar la arquitectura Ayllu.

En la segunda sección titulada *Diseño conceptual de la arquitectura*, se introducirán formalmente los conceptos de groupware y el paradigma de las 5C, que es el modelo de cooperación entre personas que se definió para la arquitectura Ayllu y el mecanismo de los grupos volátiles, que soportarán los servicios cooperativos ofrecidos por la arquitectura y que en conjunto conforman la base para definir el modelo propuesto para la arquitectura Ayllu.

En la tercera sección, *Implementación de la arquitectura Ayllu,* se explicará el proceso que se realizó para desarrollar la arquitectura, donde se utilizó un proceso de ingeniería de software corto, con el fin de definir formalmente la arquitectura. En esta sección se podrán encontrar formalmente definidas las interacciones entre los agentes y componentes de Ayllu, identificadas en el modelo conceptual, la definición de los requerimientos funcionales y no funcionales para Ayllu y se explicará detalladamente cada uno de los servicios cooperativos y componentes que se implementaron para Ayllu.

Finalmente, la cuarta sección titulada *Caso de estudio*, se define la aplicación que utilizó la funcionalidad proporcionada por la arquitectura Ayllu y las pruebas realizadas para poder validar la arquitectura a partir del caso de estudio seleccionado. Por último se establecerá la *Discusión y conclusiones* acerca del trabajo realizado y el *Trabajo futuro* que puede surgir a partir de éste proyecto.

1. ESTUDIO DE ARQUITECTURAS PARA EL ENTORNO MÓVIL

Hoy en día existen diversas aproximaciones y soluciones para el tema de la colaboración móvil. Se han encontrado distintas propuestas y cada una tiene una aproximación diferente. En la mayoría de los casos las soluciones resuelven algunos problemas pero dan lugar a otros inconvenientes o interrogantes.

Para poder diseñar una arquitectura que cumpla con los objetivos planteados en éste proyecto, se identificarán varios factores deseables que permitirán determinar con exactitud hacia donde enfocarse.

Con el objetivo de lograr la flexibilidad y escalabilidad de la arquitectura planteada en el objetivo general del proyecto, una de las características deseables es que la arquitectura maneje P2P (*peer-to-peer*); esto significa que hosts móviles pueden cambiar su ubicación dentro de la red continuamente y establecer relaciones directamente entre ellos, basándose en la proximidad.

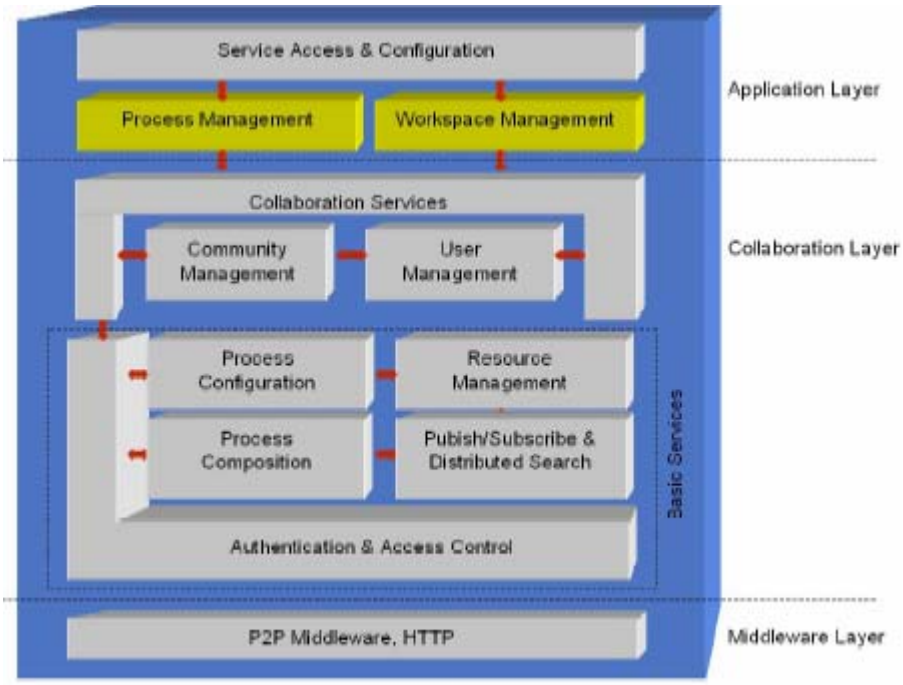
También, se debe manejar el tema de seguridad de la información residente en cada uno de los dispositivos que hagan parte de una comunidad virtual. Compartir e intercambiar información es de vital importancia si se desea obtener niveles altos de colaboración entre varias personas y cualquier incidente como por ejemplo, robo o alteración no autorizada de datos, puede debilitar las relaciones de confianza, no solo entre participantes sino hacia el sistema en general.

Se deben definir los servicios que se prestarán y tipos de comunicación que se manejarán (asincrónica y/o sincrónica); en consecuencia, la arquitectura podría presentar componentes y/o protocolos adicionales a los que se tendrían en cuenta regularmente, obteniendo una arquitectura N-Tier.

De acuerdo con [DS2002], en términos de un sistema Distribuido y Móvil Colaborativo (*DMC: Distributed and Mobile Collaborative*) las capas y componentes de una arquitectura pueden ser los que se muestran en la figura 1-1.

La capa de middleware proporcionará los mecanismos de comunicación entre los peers y los componentes de software; la capa colaborativa permitirá tener acceso a todas las funciones de trabajo en equipo y la capa de aplicación utilizará y configurará servicios específicos según las reglas de negocio.

FIGURA 1-1. Componentes de una DMC.



1.1 ANÁLISIS DE ARQUITECTURAS

1.1.1 Arquitectura QuickStep. QuickStep es una arquitectura diseñada por Jörg Roth y Claus Unger [RJ2001] de la Universidad de Hagen en Alemania, especialmente diseñada para aplicaciones de groupware sobre dispositivos móviles[1].

Su propósito es proporcionar una infraestructura que soporte interacción sincrónica entre varios miembros de una comunidad, algo bastante interesante por que generalmente las relaciones al interior de una comunidad virtual se dan de manera asincrónica por medio de mail, foros, message boards, etc.

Para su diseño se tuvieron en cuenta las limitaciones de los dispositivos móviles, además de consideraciones específicas respecto a la manera en que los dispositivos almacenan la información como se verá más adelante.

La arquitectura *QuickStep* proporciona servicios de comunicación y primitivas de colaboración que permiten al desarrollador concentrarse en detalles específicos de la aplicación que está desarrollando.

Se puede describir *QuickStep* de la siguiente manera [RJ2001]:

[1] Los diseñadores de la arquitectura han decidido soportar solamente dispositivos *handheld* (aquellos dispositivos de pequeño display, por ejemplo 160x160, y manejados con lápiz).

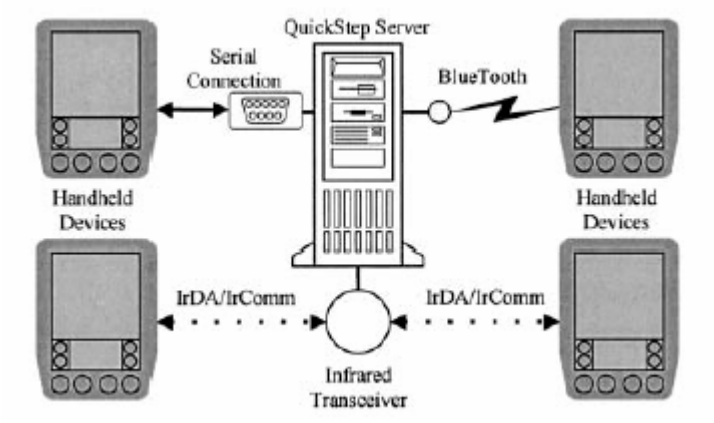
- Soporta aplicaciones que manejen información orientada a registros. No soporta multimedia, aplicaciones orientadas a gráficos o flujos continuos de información.
- Su objetivo principal es soportar colaboración sincrónica.
- Proporciona un servidor genérico que soporta cualquier aplicación cliente sin modificar o reconfigurar el servidor.
- La arquitectura asegura la privacidad de la información individual contenida en los dispositivos.
- *QuickStep* ofrece la posibilidad de incluir elementos gráficos de conocimiento para informar al usuario sobre el estado de colaboración y de contexto de trabajo.

✓ *Infraestructura de Comunicaciones.* Actualmente los dispositivos móviles (handhelds), son bastante limitados en lo que a capacidad de procesamiento se refiere. No permiten el manejo de procesos en background, algo importante cuando se manejan peticiones de comunicación. Es por esta razón, que *QuickStep* utiliza un servidor tradicional como puente para la interacción de los clientes, como se muestra en la figura 1-2.

Como se dijo anteriormente, el servidor de *QuickStep* es independiente de la aplicación que utilice sus servicios y actúa como punto de encuentro para todos los miembros de una comunidad sin importar cuales sean sus actividades; por ejemplo, podría ser una sala pública o un centro de reuniones para personal específico de una organización.

La aplicación del servidor no cuenta con interfaz gráfica, razón por la cual no necesita un operador y puede correr de forma relativamente transparente.

FIGURA 1-2. Comunicación en *QuickStep.*



✓ *Manejo de Grupos. QuickStep* no define de manera explicita los grupos en los cuales interactuarán los clientes. Estos grupos se definen de acuerdo a la aplicación que cada individuo esté utilizando; todas las personas que utilicen la misma aplicación son "miembros" de un mismo grupo de interacción.

Debido a esto, el manejo de la seguridad de la información individual se convierte en un factor crucial, puesto que cualquier persona puede unirse a un grupo sin tener la debida autorización e intentar obtener información de otras personas de manera ilegal. *QuickStep* integra un mecanismo que hace que la información publicada sea anónima, evitando así acceso no permitido a la misma.

La arquitectura presenta el concepto de *Relaxed Synchronous Collaboration* que se sitúa en medio de la colaboración sincrónica y asincrónica. El concepto se refiere a que durante las desconexiones de los usuarios, el sistema almacenará copias de la información en el handheld durante un período pre-establecido de tiempo para permitir su consulta mientras se produce la reconexión (situación bastante frecuente debido a la inestabilidad de las redes para estos dispositivos) y crear la sensación de estar conectado.

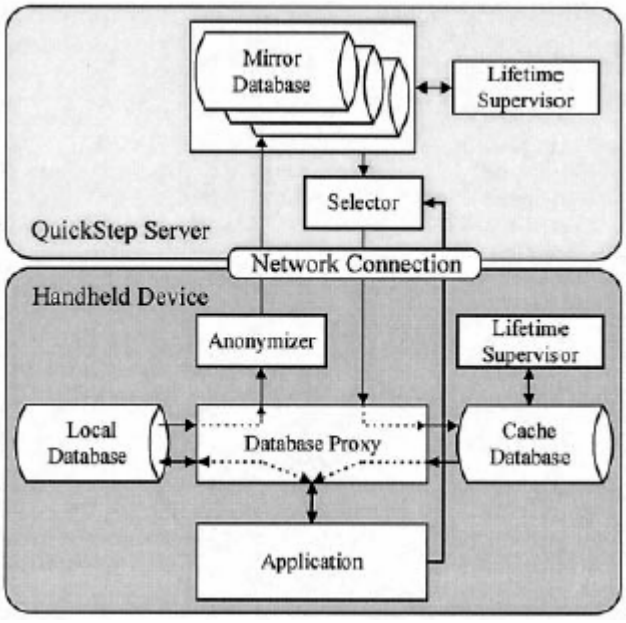
✓ *Manejo de Datos.* Los sistemas operativos de los dispositivos handheld utilizan datos estructurados y orientados a registros que conforman una entidad denominada *Bases de Datos*[2]. La arquitectura *QuickStep* proporciona un API para manejar los datos de manera similar desde las aplicaciones que se desarrollen. La estructura y la manera en que se interpretan los datos solo son conocidas por las aplicaciones y no por el servidor *QuickStep*.

Como en todo sistema distribuido, la concurrencia se presenta como un problema para manejar con cuidado. La aproximación de *QuickStep* es bastante sencilla: evitar manipulación concurrente de datos.

Para lograrlo, los datos solo pueden ser manipulados por el dispositivo que creó el registro originalmente. Las copias en otros dispositivos solamente pueden ser vistas y si se desea modificar los datos de otro miembro se debe hacer una copia por parte de quien desea cambiarlos, que será tratada como un nuevo registro.

Por otra parte, la arquitectura propone una manera de disminuir la carga de procesamiento en los dispositivos así como el tráfico en la red. Es un mecanismo de *Mirroring and Caching*, que se muestra en la figura 1-3.

FIGURA 1-3. Arquitectura de QuickStep: Mirroring and Caching.



[2] No se refiere a las bases de datos tradicionales [JC2001].

En la gráfica se aprecia que cada dispositivo tiene su propia *Base de Datos* con la información de la aplicación que esté utilizando. Cada vez que se ingresa al servidor, se hace una copia de los datos locales; si la copia ya existe, se actualiza con los nuevos datos que puedan existir.

Para permitir el manejo y consulta de los datos en caso de una desconexión se emplea el mecanismo descrito anteriormente, en la sección de manejo de grupos.

Una aplicación tiene que acceder a la información a la base local y de caché a través del Proxy. El *Anonymizer*, permite marcar la información como privada o no privada; si es privada, no se transferirá una copia al servidor bajo ninguna circunstancia.

Cada registro tiene un indicador de tiempo de vida tanto en el servidor como en los dispositivos (cuando es una copia de la información de otro usuario). El *Lifetime Supervisor* es el encargado de verificar que información mantener y cual eliminar. De esta forma se garantiza que las copias de la información de un usuario en otros handheld no permanezcan por siempre.

✓ *Análisis de la arquitectura QuickStep.* *QuickStep* presenta planteamientos interesantes. Por ejemplo, la seguridad y privacidad de la información residente en los dispositivos móviles es de vital importancia; no se debe permitir la perdida, difusión o alteración no autorizada de los datos (agendas personales, listas de pendientes por hacer, etc).

El sentido de pertenencia al grupo, conocido también como *collaboration awareness*, que proporcionan las aplicaciones de *QuickStep* puede ser más relevante y útil para el usuario de lo que es actualmente.

Podría presentarse información histórica de la participación de los usuarios, frecuencia de conexión, horarios regulares de conexión e intereses, entre otras, con el fin de facilitar las labores colaborativas (por ejemplo reuniones), y trascender más allá de proporcionar información sobre el estado de la conexión, proporcionando no solo un sentido de pertenencia más relevante sino también, haciendo sentir que hay más usuarios involucrados.

La aproximación de *QuickStep* hacia la concurrencia y el manejo de datos es bastante sencilla, aunque dentro de ambientes colaborativos no limitados, móviles y de condiciones cambiantes, podría resultar bastante compleja la labor de manejo de información, puesto que por naturaleza la arquitectura debería ser distribuida y no centralizada como *QuickStep.*

El hecho de ser centralizada, es un punto en contra de *QuickStep*; la arquitectura no parece ser lo suficientemente flexible y escalable como para soportar un gran número de usuarios y está diseñada para manejar colaboración tradicional, dependiente de los usuarios y no basada en agentes software.

Lo anterior supone que para poder implementar servicios colaborativos con agentes, y soportar ambientes móviles mucho más complejos, sería necesario utilizar otra

arquitectura puesto que *QuickStep* no podría satisfacer los nuevos requerimientos, a menos que fuera rediseñada para tal fin.

1.1.2 Arquitectura Proem. Proem es una plataforma de middleware para computación móvil P2P diseñada por Gerb Kortuem en la Universidad de Oregon. Presenta una solución para desarrollar e implementar aplicaciones peer-to-peer en redes móviles ad-hoc[3]. Es de utilidad cuando se requiere soportar reuniones esporádicas en empresas, control de pacientes en el área de la medicina y aplicaciones académicas de aula virtual.

Dentro de ésta arquitectura, las aplicaciones son denominadas *Peerlets* y son ejecutadas por un runtime denominado *Peerlet Engine*. Los *Peerlets* están basados en un modelo de programación orientado a eventos a partir de los cuales reaccionan y se comunican. El *Peerlet Engine* dispara eventos a los *Peerlets* cada vez que se presenta un cambio de estado interno o externo.

Proem define una serie de protocolos de comunicación que proporcionan la sintaxis y semántica de los mensajes que intercambian los peers para cumplir con sus objetivos y prestar servicios colaborativos [KG2002]:

- Protocolo de Transporte: Protocolo no confiable, no orientado a conexión y asincrónico que soporta la comunicación básica entre peers.
- Protocolo de Presencia: Permite a los usuarios anunciar su presencia y detectar la presencia de otros.
- Protocolo de Datos: Permite e intercambio de datos y sincronización de los mismos entre los peers.
- Protocolo de Comunidad: Permite a los peers formar relaciones de confianza a través de la creación de grupos de peers que confían entre ellos.

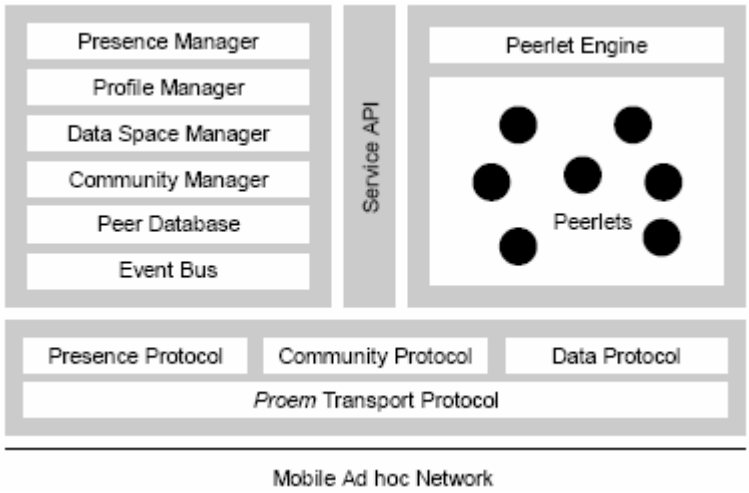
También se definen servicios que proporcionan funcionalidades comunes para todos los *Peerlets* [KG2002]:

- Administrador de Presencia: Responsable de anunciar la presencia de los usuarios y descubrir otros cercanos[4].
- Administrador de Perfiles: Mantiene la información sobre la identidad de los usuarios, relaciones, etc.
- Administrador de Espacio de Datos: Responsable de la administración de datos persistentes y control de acceso a los mismos.
- Administrador de Comunidad: Controla la pertenencia de los peers a los grupos.
- Base de Datos de Peers: Mantiene un log persistente de los encuentros entre peers, permitiendo generar información histórica sobre cuando y donde se ha encontrado un peer en particular. Esto presenta un valor agregado muy importante para las relaciones de comunidad y colaboración, facilitando encuentros futuros y coordinación de tareas.
- Bus de Eventos: Activa la comunicación basada en eventos entre los *Peerlets*.

[3] Redes cuya topología cambia constantemente debido a la ubicación de los dispositivos que a ella se conectan.
[4] Por cercano se entiende todos aquellos dispositivos que son alcanzables directa o indirectamente; depende de la topología de la red en ese momento [GK2002].

La figura 1-4 muestra la arquitectura general de *Proem.*

FIGURA 1-4. Plataforma Proem.



✓ *Análisis de Proem*. Proem es un middleware bastante flexible y novedoso que puede ser bastante útil si se desea implementar sistemas colaborativos para dispositivos móviles.

El core de la arquitectura, presenta protocolos novedosos que no se han encontrado explícitamente en otras arquitecturas; como por ejemplo el Protocolo de Presencia y el Protocolo de Comunidad.

En cuanto a servicios, todos los servicios que presta *Proem* son bastante interesantes y permiten enfocarse más en otros aspectos del desarrollo y delegar responsabilidades sobre el middleware, sin mencionar que el Administrador de Espacio de Datos y la Base de Datos de Peers no han sido encontrados en ninguna arquitectura o proyectos similares.

Esta es una propuesta que se perfila fuertemente como alternativa de desarrollo, pero nace la inquietud sobre su implementación de manera distribuida: ¿será posible?, ¿qué mecanismos de comunicación se deberían implementar para alcanzar la máxima eficiencia?

Como alternativas iniciales se puede pensar en mecanismos de Blackboard o MOM, cada uno con sus respectivas ventajas y desventajas; por ejemplo, en el caso del Blackboard se tendría que determinar con que frecuencia se realiza el escrutinio y en el MOM se debería establecer un mecanismo para determinar que mensajes son verdaderamente importantes.

La implementación de alguno de estos mecanismos dependerá en gran parte de las necesidades y requerimientos de aplicación que utilizará los servicios.

1.1.3 Arquitectura SALSA. SALSA es una arquitectura diseñada por Marcela Rodríguez y Jesús Favela del Departamento de Ciencias de la Computación, CICESE en Ensenada - México, y su objetivo es facilitar la implementación de sistemas ubicuos.

La problemática que motivó su desarrollo, radica en como hacer que los usuarios móviles sigan haciendo parte de un ambiente computacional ubicuo a pesar de estar desconectados y poder seguir realizando sus tareas para conseguir su objetivo; dentro de ésta aproximación, un ambiente computacional ubicuo es visto como un sistema multi-agentes en el cual los agentes poseen diferentes capacidades entre las que es importante destacar la asistencia al usuario dentro de sus actividades cotidianas, la realización de tareas a nombre del usuario aún cuando se encuentre desconectado y la interacción entre dispositivos y servicios vía Web.

Los investigadores definieron los requerimientos de la arquitectura con base en la observación de diferentes escenarios. Dos de los más destacados fueron un agente asistente en una biblioteca y un agente que maneja servicios en un ambiente casero; como consecuencia de estas observaciones, las características definidas para SALSA son [RM2003]:

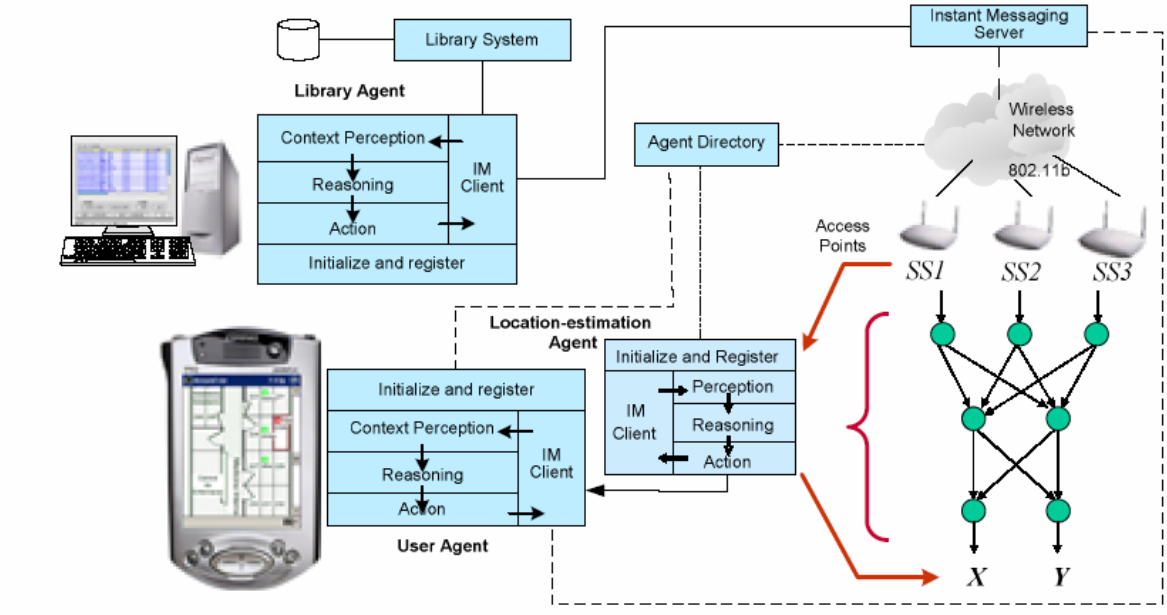
- SALSA proporciona una infraestructura de servicios que permite naming, registro, autenticación y localización de agentes.
- Un API sencillo para el desarrollo de agentes autónomos que posean las siguientes características:
    - Proactividad.
    - Ciclo de vida definido.
    - Detección del medio ambiente.
    - Comportamiento colaborativo.
    - Capacidad de comunicarse con los usuarios, dispositivos y otros agentes por medio de un lenguaje común.
- Soporte para modos desconectados de operación, lo que permite a un agente actuar en representación de un usuario móvil.
- Interoperación entre usuarios y dispositivos.
- Capacidad de lanzar los agentes de manera explicita o automática.
- El usuario puede especificar un conjunto de circunstancias o contextos que son necesarios para que los agentes lleven a cabo sus tareas. Para estos escenarios se definen 5 elementos relevantes:
    - Identificar las personas y servicios para ser ofrecidos.
    - Establecer el momento en el tiempo en el cual ocurre un evento o esperar que ocurra uno.
    - Encontrar la localización de los dispositivos de los usuarios.
    - Identificar el estado de los servicios, que son representados por agentes SALSA.
    - Recordar la información personal de los usuarios, que se encentra almacenada en un dispositivo PDA.

El framework para desarrollo de agentes que proporciona SALSA define un ciclo de vida para los agentes con siete estados básicos:

a) Activo: cuando el agente es creado.
b) Aprendizaje: en el agente adquiere conocimientos de su ambiente.
c) Pensante: momento en el cual el agente analiza el conocimiento adquirido y decide que acciones tomar.
d) En ejecución: cuando el agente ejecuta el plan de acción.
e) Comunicación: se refiere al momento en el que el agente se comunica con otros agentes.
f) Suspendido: cuando el agente no puede seguir realizando sus tareas debido a que necesita respuesta de otros agentes, se suspende hasta que pueda continuar.
g) Desactivado: una vez terminada su labor, el agente se desactiva.

SALSA también proporciona un lenguaje para la interacción entre agentes, usuario y servicios vía Web, basado en XML. Los componentes básicos de una aplicación diseñada en salsa, se muestran en la figura 1-5 (siguiente página).

FIGURA 1-5. Arquitectura de una aplicación creada con SALSA.



Los autores destacan varios aspectos de la arquitectura. En primer lugar, están los *Acces Points*, que permiten la conectividad inalámbrica a través del estándar IEEE 802.11b. En segundo lugar se encuentra el servidor de mensajería instantánea, que ha sido tomado de Jabber Software Foundation. Es un conjunto de protocolos basados en XML y tecnologías que permiten a dos entidades en Internet intercambiar mensajes, detectar presencias y otros tipos de información estructurada en tiempo real.

Luego, tenemos los *Agentes* que pueden representar usuarios, servicios o dispositivos. Un agente, a su vez, está constituido por varios componentes:

- Protocolo de registro, para poder registrarse ante el Directorio de Agentes.
- Un cliente Jabber para manejar las interacciones.
- Un subsistema que implementa la inteligencia del agente.

Finalmente, la librería SALSA, que es en esencia un conjunto de clases abstractas que proporcionan los mecanismos necesarios para la definición de objetivos, interacción y aprendizaje de los agentes.

✓ *Análisis de la arquitectura SALSA*. SALSA es una arquitectura que presenta una visión orientada a los agentes como entes representantes de un usuario más que a la colaboración y formación de comunidades. Sin embargo, aporta ideas interesantes que pueden ser útiles para el desarrollo de Groupware y la implementación de agentes como tal.

La capacidad del usuario para poder definir un contexto dentro del cual se desenvolverá el agente representante, aumenta la flexibilidad y la gama de servicios que puede prestar un sistema desarrollado con esta arquitectura. Sin embargo, así mismo aumenta la complejidad a momento de implementar los agentes a pesar del API que proporciona SALSA.

A pesar de esto, el hecho que el agente cumpla con una tarea y negocie con otros agentes no significa que exista una comunidad colaborativa, puesto que cada uno de los entes involucrados no posee afinidad ni objetivos en común. Si los agentes representantes pudieran formar comunidades y tener objetivos en común, se podría hablar de la creación de groupware dentro del cual cada agente actúa en nombre de un usuario dentro de un ambiente colaborativo.

Como se mencionó en las generalidades, los autores del artículo escogieron dos casos de estudio para determinar las características deseables de SALSA. En el primer caso (agente asistente de una biblioteca), podemos destacar el uso de sistemas de posicionamiento que reaccionan ante la proximidad de un dispositivo móvil y le envían un mapa con su ubicación actual y la ubicación del libro que desea obtener.

En el segundo caso (agente que maneja servicios en un ambiente casero), un agente se encarga de monitorear las terapias que un médico a domicilio enseña a su paciente todas las semanas; cuando el doctor ingresa a la casa, un agente realiza el reconocimiento de cara a través de una cámara y comienza la grabación de la sesión mientras avisa por medio del sistema de mensajería de SALSA a la persona interesada.

Estas dos aplicaciones pueden ser alternativas a tener en cuenta en el momento de escoger un caso de estudio.

1.1.4 Arquitectura DACIA. Creado por Radu Litiu y Atul Prakash del Departamento de Ingeniería Eléctrica y Ciencias de la Computación de la Universidad de Michigan, DACIA proporciona un framework para la construcción de aplicaciones distribuidas de una manera modular por medio de la composición de módulos de software [LR2000].

DACIA nace de la necesidad de crear aplicaciones de Groupware que se adapten a los recursos existentes, nuevos requerimientos y que brinden soporte para la movilidad.

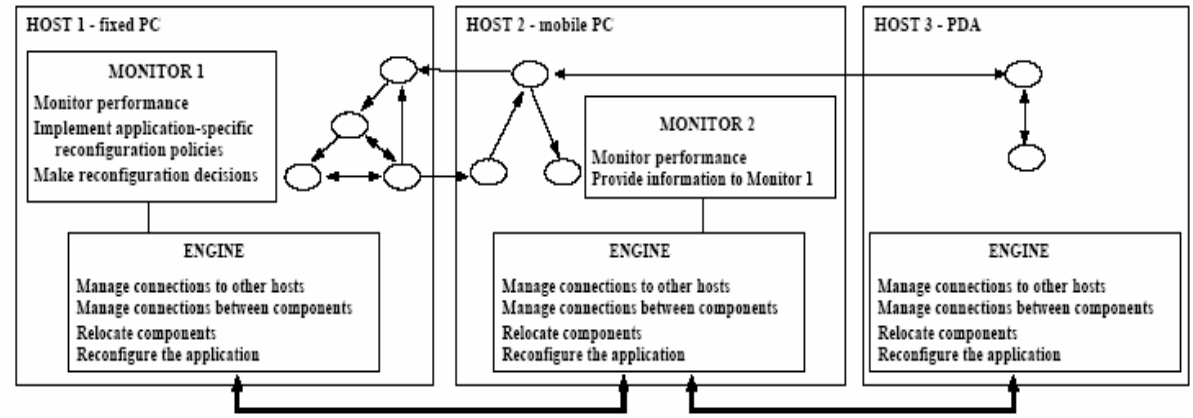
Dentro de DACIA, las aplicaciones se manejan como un grafo dirigido de componentes conectados, en donde cada arco determina el flujo de trabajo de la aplicación. El grafo podrá tener ciclos y múltiples caminos entre nodos.

Cada componente dentro de DACIA se denomina *Proc (Processing and Routing Component)*. Un *Proc* es la unidad fundamental para la construcción de las aplicaciones. Son entidades que pueden mantener un estado, ser reubicadas, interrumpidas, inicializadas y mantienen relaciones con otras entidades.

Los *Procs* se comunican por intercambio de mensajes a través de puertos de entrada y salida, soportando comunicación sincrónica y asincrónica.

La figura 1-6 (siguiente página) muestra la estructura general de una aplicación diseñada con DACIA:

FIGURA 1-6. Estructura de una aplicación DACIA.



En la figura se muestran dos componentes importantes: *Engine* y *Monitor*. El *Engine* desacopla la aplicación y la funcionalidad específica de las tareas administrativas tales como mantener la lista de *Procs* y sus conexiones, migrar *Procs* y mantener la comunicación. Se requiere un *Engine* en cada host donde se corre la aplicación. El *Engine* es reutilizable y se puede utilizar en cualquier aplicación. El *Monitor* trabaja de la mano con el *Engine*, y es la parte de la aplicación que monitorea su desempeño, realiza las decisiones de re-configuración que son comunicadas al motor principal.

La movilidad y la conectividad intermitente son factores que DACIA también pretende solucionar. Para ello, permite mover componentes entre diferentes hosts, a través de serialización Java. Cada vez que se mueve un componente, éste envía una notificación a todos los demás componentes que se encuentren conectados a él en ese momento; esto proporciona conexión persistente entre los *Procs*.

Si el usuario se desconecta, la aplicación entra en un estado de *parking*. Una aplicación en este estado, es capaz de seguir interactuando en representación del usuario con otros componentes, pero bajo ciertas limitaciones. En el momento en que el usuario se vuelva a conectar, así sea desde un host diferente, puede tomar el control de la aplicación inicialmente parqueada.

La reconfiguración dinámica de las aplicaciones es una característica bastante interesante de DACIA; esto permite que las aplicaciones de groupware sean más flexibles, adaptándose con mayor rapidez a los cambios en el entorno, los requerimientos y el número de usuarios.

Para DACIA, el proceso de reconfigurar una aplicación consiste en reordenar o reubicar algunos componentes o reemplazar un conjunto de estos con uno nuevo; como consecuencia, el grafo general de la aplicación cambiará adaptándose al nuevo entorno y cambiando el flujo de trabajo según sea necesario. Los cambios en la aplicación pueden realizarse en tiempo de ejecución.

✓ *Análisis de la arquitectura DACIA.* A diferencia de otras arquitecturas analizadas, DACIA está enfocada al desarrollo de componentes para groupware y no específicamente a ambientes móviles, sino que también soporta entornos no-móviles.

Hoy en día, las organizaciones presentan cambios constantes en varios aspectos como número creciente de usuarios para sus aplicaciones, requieren soporte a la movilidad y la infraestructura disponible cambia dependiendo de las necesidades específicas de cada una. Es por esto que DACIA puede ser una arquitectura bastante útil para el desarrollo de groupware porque presenta soluciones a cada uno de estos factores.

La relativa facilidad para poder reconfigurar el flujo de trabajo y la manera en que los componentes interactúan entre sí, trae como consecuencia mayor flexibilidad en el sistema, permitiendo adaptarse a los nuevos requerimientos del negocio.

De igual manera, la posibilidad de migrar componentes a diferentes tipos de dispositivo y el soporte a desconexión, hace más sencillo manejar las limitaciones inherentes a los ambientes móviles.

A pesar de todas estas bondades, DACIA no proporciona un marco de trabajo ni herramientas orientadas a colaboración, lo que es una desventaja porque no se explota todo su potencial. De igual manera, no proporciona librerías con componentes (*Procs)* estándar que puedan ser utilizados por cualquier aplicación, esto significa que se deben implementar todos los componentes básicos desde un principio, generando mayores costos y tiempo de desarrollo.

El hecho de ser “código móvil”, puede traer problemas de coherencia en los mensajes o de comunicación entre componentes asociados si no se tiene un estricto control sobre la movilidad de los componentes, pudiendo llegar a afectar el resultado del flujo de trabajo. Al respecto, el artículo [LR2000] es bastante pobre y no entra en detalles técnicos ni cuestionamientos sobre el tema.

Se concluye que DACIA proporciona buenas ideas para el desarrollo de groupware, pero no es del todo satisfactoria hasta que no logre proporcionar servicios colaborativos genéricos para ser utilizados por cualquier aplicación y que puedan, en un momento dado, ser la materia prima para la generación de verdaderos grupos de trabajo.

1.1.5 Arquitectura MOTION (Mobile Teamwork Infraestructure for Organizations Networking). MOTION [KE2002] es una arquitectura de servicios desarrollada para soportar el trabajo en equipo, teniendo en cuenta diferentes tipos de dispositivos y que usa XML meta data y XQL (XML Query Languaje) para consultas distribuidas y suscripciones.

Esta arquitectura proporciona los siguientes servicios:

- Localización de documentos y expertos a través de búsquedas distribuidas.
- Suscripción y notificación de información.
- Compartir recursos y acceder a información de manera móvil.

Lo que busca este proyecto es crear una arquitectura de TIC abierta, escalable y flexible para la colaboración móvil.

✓ *Descripción de la arquitectura de servicios MOTION.* MOTION es un sistema compuesto por peers [KE2002], que pueden ser clientes o hosts de servicios. Si un peer puede correr las librerías de MOTION podrá actuar como un host de servicios. Estos peers contemplan todas las formas de dispositivos (PDA, Laptop, Celulares, PC, etc).

La arquitectura se compone de 3 capas: el middleware de comunicación, la capa de servicios para el trabajo en equipo (Teamwork services layer TWS) y la capa de presentación.

El Middleware de comunicaciones: esta capa ofrece los servicios básicos de comunicación como por ejemplo el compartir archivos, búsquedas distribuidas, publicación y notificación de servicios; se basa en P2P.

La capa de servicios para el trabajo en equipo (TWS): esta capa está ubicada justo encima del middleware de comunicación. La capa integra los componentes del sistema básicos como el repositorio de datos y DUMAS (The Dynamic User Management and Access Control Component). También proporciona un API para los servicios de trabajo en equipo.

Los servicios ofrecidos por este API son los siguientes: guardar los artefactos y su perfil en el repositorio de datos, manejo de recursos (Artefactos, usuarios y comunidades), compartir los artefactos con otros miembros de la comunidad, suscripción a diferentes eventos en el sistema MOTION, envío y recepción de mensajes de otros usuarios o del sistema, manejo de acceso, permiso y búsqueda de recursos basado en la información de un perfil.

Esta arquitectura tiene como novedad, que utilizando el API proporcionado, se pueden especificar nuevos servicios colaborativos sobre los ya existentes para una organización o un negocio en particular.
La capa de presentación: esta capa proporciona una interfase para los servicios ofrecidos por el sistema MOTION. La capa es construida utilizando el API de TWS.

✓ *Componentes de los servicios para el trabajo en equipo TWS. DUMAS:* es un componente para el control de acceso. Su objetivo fue la creación de un componente genérico y parametrizable, que satisficiera los requerimientos de seguridad. Sus funciones son las de manejar usuarios y roles, por medio de la asignación de roles a usuarios.

La funcionalidad de DUMAS se divide en tres sub-componentes que son administración de usuarios, administración de comunidades y el componente de autorización.

Dumas sigue una arquitectura manejada por el modelo P2P de compartir archivos y los requerimientos para soportar la movilidad. Se utiliza un componente para el acceso al control de datos (ACD), éste se divide en dos tipos. En el primero se guardan los artefactos con ACD y luego se distribuyen por los peers. En el segundo se aplica ACD a usuarios y comunidades y éstas se guardan en servidores distribuidos.

DUMAS puede ser configurado para que corra en una gran cantidad de dispositivos móviles. El problema de la movilidad se maneja mediante la publicación de eventos XML; con una adecuada configuración del componente DUMAS se logrará hacer que responda a estas peticiones y ejecute las operaciones especificadas. Si hay latencia o carencia de la señal también se pone en cola un evento XML que será procesado lo más pronto posible mediante el servicio de publicación y suscripción.

✓ *Componente de mensajería MOTION:* la mensajería es un servicio que le permite a los usuarios comunicarse e intercambiar información. Las notificaciones basadas en suscripciones también son repartidas por este servicio.

Los mensajes enviados a los usuarios utilizan distintas tecnologías, como son ligthweigth push, e-mail (SMPT), GSM (SMS) y el wireless application protocol service indication (WAP SI). Para efectos de administración todos los mensajes son referenciados por los login names en vez de direcciones de correo convencionales. Los mensajes que se envían a teléfonos móviles se fraccionan en múltiples SMS o WAP SI, función que generalmente lo realiza un gateway que soporte estos protocolos.

El componente de mensajería MOTION se compone a su vez de 5 componentes que son los gateway (SMS, SMPT, Standard, WAP) y un quinto llamado componente front end, que hace una interfase entre los servicios específicos del negocio y el componente de mensajería, proporcionando transparencia para los servicios del negocio mediante el uso de simples primitivas para enviar mensajes.

Estos mensajes se transforman en eventos XML que son publicados por el componente de publicación y suscripción. Una vez enviado un mensaje, se toma dicho evento y se envía utilizando el protocolo adecuado. Si uno de los gateway no esta disponible, el mensaje se almacena en el repositorio del peer que corre el gateway hasta que se pueda enviar.

✓ *Componente de publicación y suscripción.* *E*ste componente proporciona a los servicios específicos del negocio la capacidad de suscripción de usuarios, artefactos y comunidades en el sistema. Éste componente se basa en el XML Query Lenguaje para realizar dichas funciones; la idea es que estas suscripciones reaccionen a eventos que sean iguales a un criterio.

✓ *El repositorio MOTION:* cada peer que corra servicios contiene un repositorio que es usado para guardar artefactos, información de perfiles sobre comunidades, usuarios y artefactos. El repositorio MOTION se integra de dos partes: un repositorio XML y un repositorio de artefactos. El primero de ellos es usado para guardar información de perfiles en XML, es segundo es utilizado para guardar artefactos del usuario.

✓ *Búsquedas distribuidas:* la arquitectura proporciona el servicio de búsquedas distribuidas similar al ofrecido por programas P2P como Gnuttella, Morpheus o Napster.

✓ *Análisis de la arquitectura MOTION.* La arquitectura MOTION es una arquitectura basada en el modelo P2P y busca ser flexible abierta y escalable. Los principales servicios que proporciona son los de localización de documentos y expertos mediante búsquedas distribuidas, la publicación y suscripción de información e información compartida en entornos móviles.

Sin embargo en cuanto a la colaboración, se ve que es limitada a las búsquedas distribuidas en el sistema MOTION y a los servicios de mensajería, donde tienen un punto a favor, pues manejan una amplia variedad de dispositivos. Aunque muchos de los servicios que ofrece la arquitectura si sean orientados al trabajo de equipo, se podría pensar en hacer más inteligente esa colaboración, mediante el uso de agentes de software que actuaran como representantes de cada peer.

Como un aspecto interesante, la arquitectura maneja un API que permite crear más servicios para un negocio en particular, cumpliendo así su cometido de ser abierta y flexible. El problema de la movilidad para los dispositivos móviles es solucionado de una manera interesante usando el componente de publicación y suscripción. Simplemente se publica un evento XML y el componente DUMAS se configura para que responda a esos eventos, así no hay necesidad de reconfigurar el dispositivo cuando se mueva de un lugar a otro. El mismo mecanismo es utilizado en el caso en que no hay conexión con algún host de servicios y se debía enviar algún mensaje.

En cuanto a los componentes de la arquitectura, es interesante la manera en que MOTION está estructurada, sobre todo por los componentes de Acceso y administración y el componente de mensajería. El primero de ellos tiene dos servicios interesantes y que deben tenerse en cuenta para el desarrollo de la arquitectura, estos son la administración de comunidades y la administración de usuarios. Aunque la arquitectura esta basada en P2P, para el manejo de las comunidades, los investigadores debieron colocarlas en servidores distribuidos, que muy seguramente deben tener una alta disponibilidad, pues en el esquema de P2P existe la posibilidad de un alto grado de desconexión o no disponibilidad para los peers. Por lo tanto la idea de integrar la arquitectura P2P con un modelo distribuido tradicional debe tenerse en cuenta como posible solución arquitectónica.

Adicionalmente el componente de mensajería se maneja de una manera interesante, pues la arquitectura utiliza métodos asincrónicos para manejar este tema. Sin embargo el verdadero potencial radica en que la arquitectura es capaz de colocar en cola los distintos mensajes e identificar cuando los peers se conectan para procesar los mensajes en cola; también es interesante en el concepto de la mensajería los tipos de mensajes que se manejan que son del sistema al usuario, sistema a la comunidad, usuario a usuario y usuario a comunidad, pues todos los mensajes son categorizados y enviados según su tipo y el estilo de configuración que tenga el peer.

1.1.6 eNcentive: A framework for intelligent Marketing in Mobile Peer–To–Peer Enviroments. Uno de las nuevas aplicaciones de los dispositivos y la computación móviles es el mercadeo, pues nuevos modelos de negocio se vienen creando para aprovechar el "boom" de estos dispositivos y para beneficiarse de todas las ventajas que estos nos ofrecen.

La arquitectura eNcentive es un claro ejemplo de una aplicación en el mercadeo. Esta arquitectura facilita el comercio electrónico basándose en P2P en ambientes móviles ad hoc [RO2003].

Básicamente eNcentive proporciona a los usuarios la capacidad de coleccionar información como cupones, promociones o descuentos electrónicos en sus dispositivos móviles. A su vez estos dispositivos actúan como propagadores de la publicidad hacia otros dispositivos que según el perfil del dueño acepten recibir la publicidad.

Los beneficios de este tipo de mercadeo se ven apreciados en que todas las partes del negocio ganan, pues el comerciante puede captar más clientes y los clientes que ayudan a propagar la publicidad van acumulando puntaje con un esquema de paso de identificadores de dispositivo a dispositivo, para que en el momento en que vayan a redimir el descuento, el sistema los premie por la colaboración.

Uno de los principales problemas en el mercadeo móvil es la privacidad. Ésta arquitectura permite que los usuarios guarden sus perfiles localmente en los dispositivos, eliminando la necesidad de enviar información a un servidor centralizado.

La arquitectura eNcentive esta basada en agentes de software, con un framework llamado Numi [RO2003]. Numi fue construida para administrar direcciones de datos, que soporte la movilidad de dispositivos móviles a través de regiones geográficas.

Cuando los dispositivos están rango de un access point, éstos pueden utilizar la infraestructura y la información; cuando los dispositivos están lejos del access point, éstos se soportan con los puntos vecinos para sus necesidades de datos. Numi proporciona la capa de comunicaciones para eNcentive con modelos basados en una infraestructura y modelos ad hoc, ofreciendo servicios como descubrimiento de dispositivos en infraestructura, manejo de lugares, comunicación de datos y mensajería, administración de aplicaciones y logging.

✓ *Configuración de eNcentive.* Hay dos configuraciones para eNcentive, la primera de ellas es *eNcentive Mobile Node Configuration* y la segunda es *eNcentive Advertiser Configuration.*
eNcentive Advertiser Configuration: esta configuración se establece en el negocio y consiste en dos agentes eNcentive: el eNcentive ad marker agent y el eNcentive marketing agent.

eNcentive ad marker agent: este agente es el encargado de proporcionar los anuncios. eNcentive marketing agent: es el agente encargado de agendar cuando se va a lanzar la promoción. En la figura 1-7 se puede observar el framework de estos dos agentes de software.

eNcentive Mobile Node Configuration: Cuando las promociones son enviadas por broadcast, éstas son atrapadas por los dispositivos móviles que tienen la plataforma de *eNcentive Mobile Node Configuration.* El framework consta de dos agentes de software el eNcentive marketing agent y el eNcentive Ad Manager agent, nótese que el primer agente se encuentra en ambos framework, pero en realidad realiza tareas diferentes.

Las funciones esenciales del eNcentive ad Manager agent son coleccionar, mantiener y organizar las promociones; la función de colección de promociones está ligada con el perfil del usuario, para asegurarse que solo las promociones que le interesen al consumidor sean aceptadas.

Otra funcionalidad de este agente es la de aceptar las peticiones de otros peers. Por otro lado la funcionalidad del eNcentive marketing agent es la de distribuir promociones que sean aprobadas por el negocio. En la figura 1-8 se puede apreciar con más detalle este framework.

FIGURA 1-7. eNcentive Advertiser Framework.

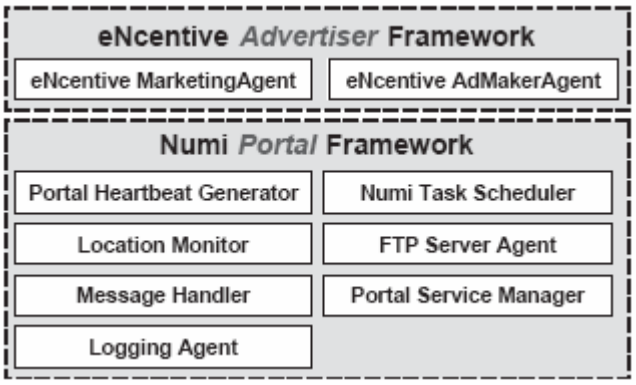
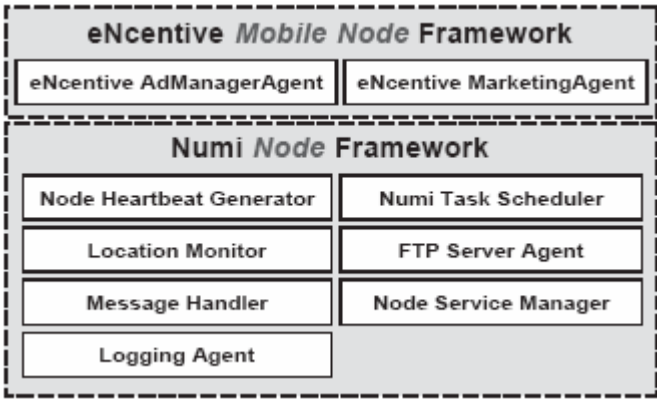


FIGURA 1-8. encentive MobileNode Framework.



✓ *Análisis de la arquitectura eNcentive.* eNcentive es un framework basado en la arquitectura P2P, su aproximación es la del uso de agentes de software para el mercadeo móvil. El framework tiene dos componentes uno móvil que se encarga de recibir las promociones y uno estático que se encarga de lanzar las promociones. Básicamente la idea de eNcentive es la de utilizar la replicación como medio de publicidad.

Existen algunos peligros en el framework como por ejemplo el spam, sin embargo los investigadores, proponen que al configurar los agentes que residen en los dispositivos móviles, éstos mantengan registro de los perfiles de los usuarios. Inherente a esto se encuentra el problema de la privacidad; la solución es la de mantener todos los datos en el dispositivo y no en servidores centralizados o distribuidos, donde se deba dejar información personal.

El esquema de ganancia, por medio de identificadores es interesante y presenta una gran oportunidad donde todas las partes reciben beneficios. Sin embargo el framework no aporta ideas en el campo colaborativo, pero éstas podrían fácilmente ser adaptas para dichos entornos, tratando de buscar colaboración.

El esquema que utiliza el framework podría ser modificado para que trabajara como motor de búsquedas distribuidas y aprovechando la flexibilidad de los agentes sería mucho más fácil realizar esta tarea. Utilizando el mismo esquema de paso de identificadores los agentes que acepten realizar colaboración, podrían mantener un record de citas y así saber quienes se conocen con quienes y en posteriores sesiones dar preferencia a agentes conocidos, también podría aplicarse este método de replicación para ubicar a otros colaboradores dentro de la red y montar sobre el framework un middleware que maneje mensajería asincrónica para poder dejar mensajes cortos de citas a una persona que este desconectada en el caso en que no lo encuentre.

Con esta idea del uso de agentes, se podrían solucionar de una manera novedosa algunos problemas colaborativos logrando una colaboración más inteligente.

1.1.7 A Collaborative Problem-Solving Framework for Mobile Devices. Esta arquitectura permite a los dispositivos móviles colaborar para desarrollar una tarea que es

de una alta complejidad computacional, basándose en el principio de la solución de problemas de software “divide y vencerás”. Básicamente lo que propone esta arquitectura es descomponer una tarea de alto nivel computacional en tareas más pequeñas y distribuirlas en varios dispositivos móviles, para que se procese más rápido. El diseño de esta arquitectura se basa en un modelo en grid (red), para resolver dichos problemas [KS2004].

✓ *Componentes de la arquitectura.* Cada uno de los dispositivos en el grid es un agente inteligente que facilita la resolución de un problema, sin que el usuario del dispositivo este conciente de cual es el método exacto que está usando el agente para resolver la tarea [KS2004].

Dependiendo de la naturaleza de la complejidad de la tarea, el agente decide buscar colaboración en el grid. Para ello se debe estar corriendo una aplicación cliente, para que intercambie información con el Brokering Service del grid. Cualquier dispositivo que corra el cliente se le llama subordinate y puede ser llamado a colaborar. Cualquier subordinate puede convertirse en un Initiator dependiendo si se ha mandado a realizar una tarea computacionalmente compleja.

El Brokering Service se compone de dos grandes repositorios, el Active Agent Repository (AAR) y la tabla de localización de tareas Task Allocation Table (TAT), que contiene información que describe como está localizada cada tarea distribuida a través del grid. Todos los aspectos de la comunicación en el grid, son manejados por el Keep-Alive Server que recibe mensajes de todos los dispositivos disponibles.

Si los dispositivos son bastante móviles y están entrando y saliendo del grid, podría ocurrir que se salgan de la celda y dejen tareas pendientes. En este caso el Brokering Service reasigna estas tareas a otros subordinados que se encuentren disponibles.

A continuación se proporciona más información acerca de los 3 puntos funcionales de la arquitectura.

Aplicación del cliente: ésta aplicación se instala en los dispositivos móviles y permite el acceso a través del grid.

Keep-Alive Server: es un servicio TCP/IP que es usado para ayudar al Brokering Service a mantener un record preciso del estado del grid. Todos los dispositivos móviles que se encuentran en el se comunican con el Keep-Alive Server a través del protocolo, con el objetivo de informar si continúan activos. Cualquier cambio en el estado del grid es reportado por los dispositivos o por el Brokering Service.

Brokering Service: es una suite de XML Web Services, que mantienen información acerca de la población del grid y de las tareas iniciadas por los dispositivos. Esta información se mantiene en el AAR y en el TAT respectivamente. En el AAR se mantiene información relacionada con varias características de cada dispositivo registrado en el grid y en el TAT se mantiene información acerca de las tareas iniciadas por cada dispositivo presente en la red.

✓ *Análisis de la arquitectura.* Este framework presentó nuevas ideas acerca de un posible servicio de procesamiento remoto de una tarea computacionalmente compleja. La aproximación de utilizar a los dispositivos móviles como agentes que tomen una fracción de una tarea mayor y la procesen es novedosa en cuanto a que lo que usualmente se piensa es en procesar dichas tareas en servidores remotos que tienen los recursos computacionales apropiados para acabar el trabajo más rápido.

Sin embargo con esta aproximación vemos que es factible utilizar a los dispositivos de una manera mucho más colaborativa que la expresada anteriormente. Aunque se utilizan agentes de software para realizar las tareas de procesamiento, se puede apreciar que hay retos mucho más grandes como por ejemplo el de la fragmentación en tareas. En el framework se especifica que se utiliza un agente que sabe como romper estas tareas y sabe además cómo repartirlas entre los dispositivos presentes en el grid.

Otro reto grande es la reasignación de tareas en el momento en que los dispositivos salen del rango de la red inalámbrica, pues si estos estaban procesando tareas el framework debe ser capaz de reasignársela a otro dispositivo que no esté procesando nada en ese momento.

Otro detalle que se debe tener en cuenta para una solución de este estilo es la capacidad de procesamiento de cada dispositivo, por que sería absurdo colocar un dispositivo de poca capacidad a que recibiera, procesara y enviara los datos al servidor, pues en este proceso se demoraría mucho y al enviar al servidor provocaría overhead en la red.

1.1.8 YCab: Decentralized Ad – Hoc Groupware API and Framework for Mobile Collaboration. Dominik Buszko, Wei – Hsing (Dan) Lee y Abdelsalam (Sumi) Helal, proponen un sistema móvil colaborativo diseñado para colaboración ad – hoc inalámbrica. Puesto que los dispositivos móviles cada vez aumentan sus capacidades para la comunicación, el siguiente paso lógico según estos investigadores es una “mayor interacción entre los usuarios móviles equipados con estos dispositivos” [BD2001].

Con base a esto se ha diseñado e implementado un ambiente colaborativo y un framework orientado a redes ad – hoc para pequeños dispositivos móviles, que proporcione tomar ventaja del control descentralizado, un ambiente colaborativo tolerante a fallas y un rápido desarrollo de espacios colaborativos para necesidades particulares.

Esta propuesta se desarrolla completamente para redes ad – hoc. La colaboración sobre este tipo de redes es muy distinta a la de las redes con topología fija, debido a su condición dinámica cuando los usuarios entran o salen de estas redes, esto proporciona pérdida de la señal o interferencia. Adicionalmente en las redes con topología fija, la calidad del servicio y la robustez son más altas en comparación con una red ad – hoc [BD2001]. Es por esta razón que este framework esta estructurado para ocultar la complejidad de implementar un servicio distribuido para el desarrollador de una aplicación y se concentra en proporcionar un ambiente tolerante a fallas que le permita a los colaboradores entrar o salir de la red ad –hoc sin causar interrupción en la sesión.

✓ *Diseño de la arquitectura.* Esta arquitectura llamada YCab, tiene como objetivo ser flexible y proporcionar un API para que se puedan desarrollar nuevos servicios; la arquitectura es totalmente configurable, pues tiene la posibilidad de personalizar los servicios y la interfaz gráfica del cliente.

En el diseño del sistema se pensó siempre en controlar la tolerancia a fallas. Hoy en la mayoría de las aplicaciones el cliente o colaborador que inicia una sesión, debe hacerlo a un servidor central, dejando en este todo el control. Éste no es un problema en las redes de topología fija, pero en el contexto de la red ad – hoc una solución de este estilo tendría un desempeño muy pobre, debido a que este sería el punto de falla. Sin embargo cada nodo en una red ad – hoc tiende a entrar o salir del rango de cobertura, las aplicaciones que corrieran en servidores serían inútiles si se salieran del rango de cobertura [BD2001].

Por esta razón, una aproximación más eficiente es la de distribuir el control sobre todos los nodos en la red, eliminando el punto de falla y proporcionando una aplicación más robusta y estable en este tipo de redes dinámicas. En la arquitectura YCab una de las principales tareas es la de distribuir el estado de la sesión sobre los participantes de la sesión, para así garantizar el soporte del control descentralizado.

El soporte para el ordenamiento de los mensajes se le delegó al protocolo 802.11, cuyas capacidades para soportar broadcast y multicast son bastante buenas y de mucho uso en el diseño de sistemas colaborativos, dado que el estándar está orientado a evitar colisiones (CSMA/CA) [BD2001]; esto da como resultado que los mensajes sean ordenados por igual para todos los clientes.

El API de la arquitectura tiene un gran uso; en un nivel simple un usuario puede usar la aplicación YCab que proporciona el API y en un nivel más complejo, se pueden usar este API para personalizar y desarrollar nuevos servicios para necesidades particulares. Según sus creadores “usando el API de servicios, los desarrolladores pueden crear poderosos módulos que podrán ser desplegados más adelante en cualquier aplicación personalizada” [BD2001].

YCab esta divido en dos partes el API del framework y el API de servicios; el API del framework tiene la responsabilidad de compartir la información de los servicios con los otros miembros en la sesión colaborativa sobre la red ad – hoc; también es responsable de implementar los protocolos de registro de la sesión, protocolos de elección de los líderes, y protocolos de recuperación de estados. El framework consiste de las siguientes partes:

- Administradores de comunicación y servicios.
- Sesión y servicios de elección.
- Administrador de recuperación de estados.
- Componentes GUI.

Los servicios implementan una interfaz que proporciona los métodos básicos por el framework y éstos deben ser implementados para cada nuevo servicio creado [BD2001].

Las dos características fundamentales de esta arquitectura son primero, que el sistema es completamente descentralizado y no requiere un cliente separado que de soporte a la sesión y segundo, la comunicación entre los clientes es hecha vía paquetes multicast. Aunque cada cliente reciba todos los mensajes que son enviados en la sesión, éste solo procesa los mensajes que están destinados para si mismo [BD2001].

Los siguientes son otros componentes importantes en la arquitectura YCab: el CoreMessage es la unidad de comunicación usada por el API, básicamente describe el encabezado y cuerpo de los mensajes; el Communication Manager es el responsable de proporcionar comunicación asincrónica entre los clientes. Todos los mensajes independientemente de su origen son enviados por el Communication Manager.

El Client Manager es el corazón del cliente. Este componente es el responsable de administrar los servicios, llevar un seguimiento de las propiedades de los clientes como son: el nombre, ranking, estatus de coordinador, etc. Adicionalmente es el responsable de instanciar los otros administradores, ruteo y filtro, también es reponsable de instanciar, inicializar, y mantener referencias de todos los componentes [BD2001].

Otro componente importante es el Message Router, quien es el responsable de enrutar los mensajes a los servicios apropiados, éste corre en continuo loop ejecutando los siguientes tres pasos:

1. Get Message: obtener mensajes por medio del Communication Manager y almacenarlos en una cola interna
2. Process Message: el Message Router extrae el nombre del servicio de la cabecera del mensaje y luego obtiene la referencia al servicio. Luego de esto el Message Router invoca el método processMessage() de ese servicio.
3. Actualizar el estado de la información: El Message Router determina si el servicio para el cual el mensaje es dirigido tiene activado el recovery. Si es así, el Message Router invoca el método updateState() del servicio, para que el mensaje pueda ser añadido al estado de servicios de acuerdo con las reglas proporcionadas por el servicio.

El Message Router puede correr en dos estados. El primero de ellos es el Normal State, donde todos los mensajes recibidos son pasados inmediatamente al servicio apropiado para que los procesen y el segundo es el State Recovery Mode, donde el cliente recupera el estado de sus servicios [BD2001].

El State Manager es otro componente importante en la arquitectura, éste debe mantener el estado de la sesión. Esta operación es transparente para el usuario, para salvar la información de la sesión, cada servicio debe registrarse ante el State Manager.

El Service y el Threaded Service, son otros dos componentes de la arquitectura; la clase Service es una clase abstracta que proporciona el framework básico para un servicio. Éste proporciona la funcionalidad básica del servicio y proporciona una guía de los métodos que se deben implementar para tener completa funcionalidad. El Threaded Service, es una extensión del objeto Service que ha sido incluida para soportar servicios en background.

En cuanto a los protocolos utilizados por la arquitectura se encuentra un protocolo para crear y entrar a una sesión, abandonar una sesión y State Recovery, adicionalmente uno para la elección de un líder y un protocolo de pinging.

Para crear o entrar a una sesión hay algunas disposiciones generales como por ejemplo, cuando un cliente va a entrar a una sesión, para ponerse al día con los otros colaboradores de la sesión, el coordinador de la sesión es el responsable de ponerlo al día. Si por el contrario este nuevo cliente trata de ingresar a una sesión pero no recibe respuesta, éste asume que es iniciador de la sesión y designado como coordinador de la sesión.

Cuando un cliente abandona la sesión envía un mensaje a todos los demás clientes de la sesión y estos lo remueven de su lista de clientes asociada. El mecanismo de State Recovery permite a los clientes colocarse al día y consistentemente con otros participantes de la sesión, para este proceso se requiere de un coordinador de sesión.

En el caso en el que el coordinador de la sesión se desconecte, se debe asegurar que las actividades van a continuar con los miembros restantes de la sesión, para lo cual existe el mecanismo de elección del líder que garantizará la elección de otro coordinador entre los miembros existentes de la sesión.

Por último el Pinging que es utilizado para garantizar la vista correcta de los miembros que están en la sesión, es utilizado para descubrir quién está conectado (vivo en la sesión) añadiéndolos o quitándolos de la lista clientes asociada de cada cliente.

Para terminar los servicios que presta la arquitectura son los siguientes: Leader Election Service, para elegir un coordinador en caso que este se desconecte, Session Manager Service, para administración de servicios en la sesión, Session Ping Service, para el descubrimiento de clientes en una sesión, Text Chat Service, Whiteboard Service, Image Viewer Service, GPS Service y un Client Info Service para el monitoreo de la sesión y estados del cliente.

✓ *Análisis de YCab.* YCab es una arquitectura orientada al trabajo en equipo y a la colaboración. Presenta unas ideas interesantes en cuanto a los servicios para un ambiente virtual colaborativo.

Una de ellas es la eliminación del punto de falla, típica de un sistema centralizado, por esta razón los creadores de YCab decidieron que el sistema debía ser descentralizado y además que el control de la sesión debería estar en cualquiera de los colaboradores conectados a una sesión. Para esto se aseguró la creación de un rol llamado coordinador, quien es el encargado de colocar al día a nuevos colaboradores que entren a la sesión, entre otras funciones. Adicionalmente en la red ad – hoc existe la posibilidad de que uno de los colaboradores se desconecte; y si este llegara a ser el coordinador, el sistema debe ser capaz de elegir a otro dentro de la sesión como coordinador, todo esto transparente para el usuario final.

La arquitectura cuenta con servicios interesantes como el de pinging para la ubicación de colaboradores y otros aún más orientados al Groupware como Whitheboard, Image Viewer Service y GPS Service.

Otra característica que la hace diferenciar de otras arquitecturas colaborativas analizadas, es que la comunicación no se basa utilizando P2P.; por el contrario, la arquitectura utiliza paquetes multicasting para enviar mensajes a los grupos. La utilización de este componente es basada en el IP-Multicasting de Java donde se crea una "canal exclusivo" para la comunicación en el grupo.

Ya que los servicios son manejados por mensajes, la arquitectura tiene un componente adicional al Communication Manager, que se llama Message Router y este es el encargado de entender que servicio es el que se desea ejecutar e invocar el método del servicio seleccionado, para obtener la respuesta deseada.

La implementación de los servicios es muy parecida al RMI de Java, pues utilizando el API de la arquitectura se podrían crear nuevos servicios. Para lograrlo se hace uso de una interfase que enuncia los métodos y otra clase que los implementa.

Por otra parte la arquitectura no contempla algunos de los servicios típicos de la colaboración como las búsquedas distribuidas ni tampoco el procesamiento de tareas, en donde podrían entrar los agentes de software a jugar un papel importante.

La arquitectura está más orientada a un soporte de actividades que tienen que ver con el Groupware, como el chat, pero trata de acercarse a lo colaborativo con servicios como el Whiteboard. De todas formas, las ideas arquitectónicas que aporta deben ser tenidas en cuenta así como también los servicios novedosos que propone.

1.1.9 Reconfigurable Context Sensitive Middleware for Pervasive Computing.

Pervasive Computing es una colección de dispositivos handheld conectados inalámbricamente a redes con topología fija como por ejemplo Internet. El mayor objetivo que persigue esta tendencia computacional es la transparencia [YS2002].

Las aplicaciones pervasive operando sobre redes ad – hoc tienen dos características importantes: sensibilidad del contexto y comunicación ad – hoc. El contexto se refiere a cualquier atributo relevante y detectable de un servicio de software o ambiente, y la sensibilidad del contexto es la habilidad de un software para sentir y analizar el contexto de distintas fuentes [YS2002].

El objetivo de este middleware es proporcionar desarrollo y soporte en tiempo de ejecución para garantizar un balance entre la sensibilidad del contexto y la transparencia de una aplicación [YS2002], en otras palabras, un middleware que analice el contexto y pueda llegar a tomar decisiones acerca de lo que siente en el ambiente, los autores nos presentan RCSM: Reconfigurable Context – Sensitive Middleware, para lograr este objetivo.

Los middleware para aplicaciones pervasive se pueden clasificar en dos grupos. La primera categoría es el software que se comunica con otros mediante lectura y escritura de uno o más espacios compartidos. La segunda categoría proporciona semánticas para la orientación de mensajes donde los objetos se vean unos a otros similarmente como un

mecanismo RPC [YS2002]. Con este middleware lo que se busca es la integración de la sensibilidad del contexto y la comunicación ad – hoc.

El middleware tiene como característica estar desarrollado y proporcionar un framework basado en objetos, para el soporte de aplicaciones con contexto sensitivo. También el uso de contenedores de objetos adaptables (ADC) y un Object Request Broker de contexto Sensitivo, como el mecanismo clave para proporcionar transparencia en la comunicación en el contexto sensitivo de la aplicación (R-ORB).

También proporcionar el descubrimiento de dispositivos y de servicios. Adicionalmente el middleware busca detección análisis y adquisición de una contexto específico, acciones dependiendo del contexto y soporte transparente para la comunicación ad - hoc [YS2002].

✓ *Análisis de la arquitectura RCSM.* RCSM es un middleware que en cierta forma hace que el software sea inteligente, mediante el análisis del contexto que lo rodea. La idea es que un dispositivo móvil con la plataforma de RCSM analiza el contexto en que se encuentre y dependiendo de eso realiza ciertas acciones programadas.

En cierta medida se podría llegar a pensar que son agentes de software primitivos, pero en realidad lo que hace posible el análisis del contexto son sensores que tiene el dispositivo móvil. Y el análisis del contexto lo realiza a sentencias programadas para reconocer esos cambios.

Uno de los problemas que se debe manejar es cada cuanto debe estar el dispositivo pendiente del contexto. Si es siempre, la carga computacional sería muy alta así fuera por medio de hardware o software y si fuera cada cierto tiempo, entra el problema de perderse algún evento que debía estar predestinado a reconocer.

Es entonces en este punto cuando deben entrar algunos servicios de Groupware como el blackboard, para captar mensajes de eventos que vayan ocurriendo en el contexto y así el dispositivo pueda disminuir la carga computacional y realizar las acciones que tiene programadas acerca del contexto que reconoce.

1.1.10 CoopScan. CoopScan es una arquitectura para el desarrollo de aplicaciones de groupware sincrónicas, creado por Roland Balter, Slim Ben Atallah y Rushed Kanawati.

El esquema empleado para ésta arquitectura no es específico de las aplicaciones de groupware, sino de todas las aplicaciones distribuidas en general; se basa en el principio WYSIWIS (What You See Is What I See) y tiene un esquema de arquitectura replicado.

Su objetivo principal es proporcionar un framework para el desarrollo de aplicaciones de groupware sincrónicas (CSCW), proporcionando mecanismos genéricos de control al para el acceso a datos, ingreso y salida de los grupos de trabajo. Para lograr esto, se utilizó el esquema de alto nivel de integración, en donde los mecanismos de cooperación se basan en eventos de la aplicación.

El esquema de alto nivel de integración solamente puede ser aplicado en aplicaciones abiertas[5], razón por la cual, los autores emplearon el paradigma de SAMD (Structured Active Multimedia Document) para superar este inconveniente.

SAMD ofrece un espacio estructurado de información y manejo activo de datos, factores esenciales para el desarrollo de aplicaciones CSCW; dentro de *CoopScan*, la interfaz de usuario de la aplicación será representada por un documento SAMD.

CoopScan se define a través de un modelo de clases abstractas, en donde se distinguen tres capas: aplicación, comunicación y conferencia.

La capa de aplicación es la aplicación del usuario como tal, la capa de comunicación encapsula los protocolos de transporte y la capa de conferencia proporciona los mecanismos para ingresar, salir y controlar las sesiones colaborativas. Esta última capa se define a través de un modelo abstracto llamado *Modelo de Agentes*, y tiene tres componentes principales [BR1995]:

- Agente Local (*Local Agent*): Representa a un usuario durante una sesión de trabajo. Maneja el acceso a la información, asignación de roles y negociación.
- Agente Distante (*Distant Agent*): Maneja todos los mensajes enviados por el Agente Local. Recibe la estructura de comunicación de la capa de comunicaciones, la decodifica y determina que función del API es necesario llamar.
- Agente de Sesión (*Session Agent*): Maneja la información de las sesiones como por ejempllo, la ubicación de los usuarios, su ubicación y aplicaciones. Es responsable del inicio de las sesiones de trabajo así como del ingreso y salida de las mismas.

CoopScan define espacios comunes de información que comparten los usuarios; estos espacios están formados por conjuntos de documentos que se han definido por parte de los usuarios.

También se tiene la aproximación orientada al usuario para iniciar las sesiones colaborativas; consiste en que los usuarios deben crear en primer lugar un contexto compartido, en el cual se incluirán posteriormente los documentos que quieran ser trabajados de manera grupal.

Para poder mantener control sobre las sesiones, *CoopScan* define tres roles:

- Chairman (*Administrador de Sesión*).
- Editor (*Operaciones de lectura y escritura*).
- Presenter (*Solo lectura*).

Cada uno de éstos roles posee ciertos permisos sobre los documentos lo que evita que alguien manipule un documento para sobre el cual no está autorizado.

[5] Aplicación abierta: Aquella que provee un mecanismo de API a través del cual puede ser manejada por módulos externos y no solamente por la interfaz de usuario. También ofrece mecanismos de callback para notificar las acciones del usuario sobre los objetos de la aplicación [RB1995].

✓ *Análisis de CoopScan.* Esta fue probablemente una de las primeras arquitecturas diseñadas específicamente para el desarrollo de aplicaciones CSCW. Las tres capas que define se pueden encontrar en arquitecturas más recientes y siguen constituyendo la base para el desarrollo de groupware en la actualidad.

CoopScan también presenta el concepto de *conocimiento* (awareness), a través del esquema de alta integración y el paradigma SAMD. Con la alta integración se garantiza atrapar todos los eventos generados por cada copia de la aplicación que se esté utilizando y poder replicarlos a los demás interesados; SAMD, por su parte, notifica al ambiente de la aplicación sobre cada una de las acciones que se ejecutan sobre la información compartida; estas son las bases del *conocimiento* por parte de los demás usuarios.

Dadas sus características y el contexto temporal en el cual fue diseñada la arquitectura, *CoopScan* presenta ciertos inconvenientes.
Se observa que la aproximación centrada en usuarios para la creación de sesiones colaborativas es una alternativa válida cuando no se consideran grupos ad-hoc ya que los colaboradores tendrían que reunirse en un sitio específico y no se podrían iniciar sesiones de manera espontánea.

De igual forma, los mecanismos de control de concurrencia, acceso a documentos y comunicación entre participantes han sido concebidos para medios de computación tradicional. Si se quisiera implementar una aplicación hecha en *CoopScan* dentro de un ambiente móvil, se tendrían que realizar varias modificaciones de fondo con el fin de soportar los nuevos requerimientos, sin mencionar que se tienen que implementar mecanismos más estrictos que los roles para poder garantizar la seguridad y privacidad de la información residente en los dispositivos.

En conclusión, esta arquitectura nos ofrece un panorama general de los requerimientos necesarios para el desarrollo de Groupware y muchas de sus características siguen siendo válidas hoy en día.

1.1.11 DISCIPLE. DISCIPLE (DIstributed System for Collaborative Information Processing and LEarning) es un framework de colaboración sincrónica diseñado por Weicong Wang, Bogdan Dorohonceanu e Ivan Marsic y pretende ser un medio que facilita compartir aplicaciones.

Está orientado a aplicaciones basadas en componentes Java, específicamente Java Beans y Applets, ya que los primeros soportan persistencia, manipulación visual, introspección, eventos y pueden ser acomodados a necesidades específicas; por otra parte, los Applets proporcionan una interfaz gráfica de usuario que corre dentro del contexto soportado por el applet.

DISCIPLE proporciona componentes de software para manejar el trabajo sincrónico en grupos, como por ejemplo control de concurrencia en actividades simultáneas.

✓ *Modelo de Colaboración.* El modelo de colaboración utilizado para *DISCIPLE* es sincrónico, con usuarios distribuidos geográficamente.

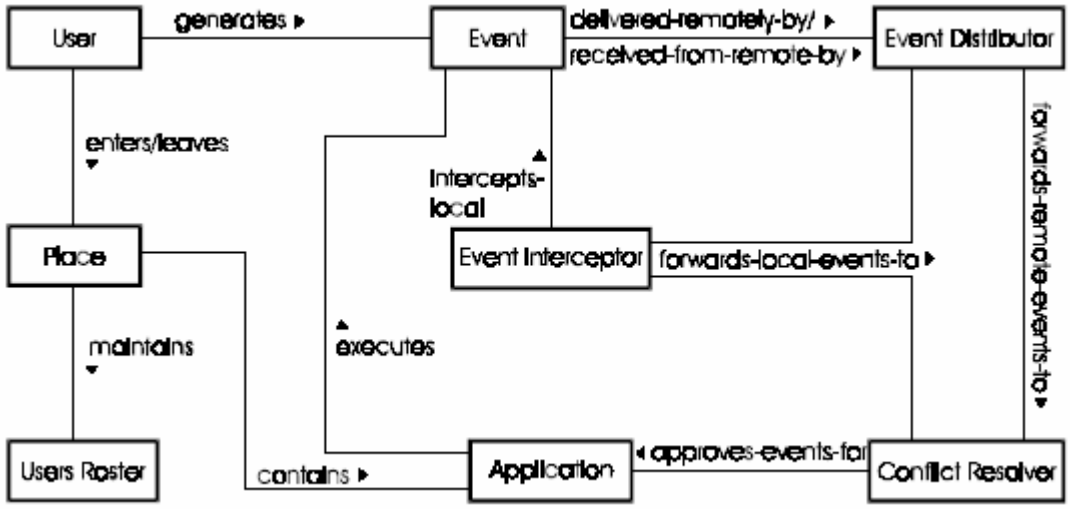
DISCIPLE utiliza el modelo de *places* (lugares de encuentro) en donde los usuarios pueden reunirse y trabajar en conjunto. Estos lugares de encuentro están relacionados con el tema de trabajo; son persistentes, lo cual significa que los usuarios pueden suspender una sesión y volver a conectarse tiempo después.

Los usuarios colaboran compartiendo artefactos, recursos y herramientas (componentes Java dentro del contexto de *DISCIPLE*), para lo cual, se definen dos elementos básicos:

- *Collaboration Aware Beans (CAB)*: Son beans que están diseñados conforme los estándares y reglas propuestas por *DISCIPLE*. Conocen otros peers y pueden enviar y recibir mensajes.
- *Collaboration Unaware Beans (CUAB)*: Son beans que se utilizan para capturar los eventos generados por la aplicación.

En la figura 1-9 se muestra el modelo conceptual para la colaboración sincrónica empleado por *DISCIPLE*.

FIGURA 1-9. Modelo conceptual de colaboración sincrónica.



✓ *Arquitectura del Sistema. DISCIPLE* es una mezcla entre cliente/servidor y P2P, y está basado en la arquitectura replicada para Groupware, y define tres capas principales:

- Presentación – GUI
- Lógica de la aplicación – Modelo conceptual del sistema.
- Almacenamiento – mecanismo de almacenamiento persistente.

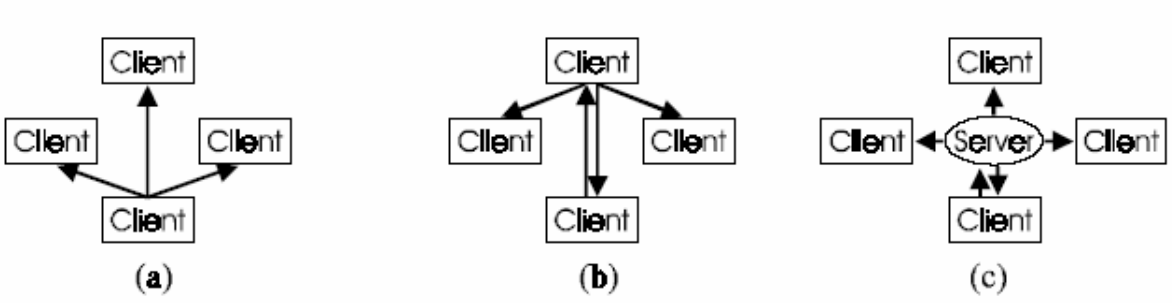
Existen canales de comunicación, entendiéndose por canal de comunicación un medio por el cual se puede intercambiar información. Cada canal posee dos roles: *Publisher* (publica los mensajes en el canal) y *Suscriber* (recibe los mensajes del canal).

Un usuario puede suscribirse a un canal de comunicación con cualquiera de los dos roles mencionados anteriormente o ambos si lo desea. Para estos canales de comunicación del estilo Publisher-Suscriber, la comunicación se lleva a cabo por multicast.

La arquitectura también proporciona canales P2P para el envío de eventos a usuarios específicos.

Uno de los factores claves dentro del Groupware sincrónico, es mantener la consistencia dentro de todas las copias de la aplicación compartida., para lo cual existen varios modelos, los cuales se muestran en la figura 1-10. *DISCIPLE* utiliza el modelo 10b, en donde un cliente asume el rol de punto de serialización con el fin de soportar grupos de trabajo móviles.

FIGURA 1-10. Modelos de distribución de eventos: a.) Cada cliente distribuye sus propios eventos; b.) un cliente actúa como punto de serialización y distribuye eventos a los demás; c.)un servidor centralizado actúa como punto de serialización.



DISCIPLE posee un *Collaboration Bus* (Bus de Colaboración), que corresponde a la capa lógica del modelo de tres capas y puede ser implementado sobre middlewares de comunicación como OMG CORBA. Su función básica es manejar los eventos generados por los usuarios para mantener el estado consistente de la colaboración.

Otro componente importante es el *Place Server*, que actúa como un directorio de todos los colaboradores, mantiene un registro histórico de sus reuniones a través del tiempo y proporciona reportes de fallas.

Los *Resource Servers*, son servidores en donde se encuentran recursos tales como imágenes, audio y archivos JAR que pueden ser usados en las labores colaborativas. Regularmente es un servidor tradicional al cual se accede simplemente con una URL. Los clientes podrían convertirse en *Resource Servers* si poseen recursos que necesiten compartir además de la aplicación.

✓ *Manejo de Eventos.* La colaboración dentro del modelo replicado de arquitecturas de Groupware, se basa principalmente en la intercepción, replicación y traducción de cambios sobre los componentes.

DISCIPLE define cuatro tipos principales de eventos: uno para los CAB, uno para los CUAB, uno para las acciones de los usuarios (p.ej: unirse a una sesión, abandonar una sesión, crear una sesión) y otro para la manipulación de los beans (agregación, distribución o remoción de un bean) y que son manejados a nivel del *Place Server.*

✓ *Análisis de la arquitectura DISCIPLE. DISCIPLE* es una arquitectura bastante robusta y completa. Proporciona todos los mecanismos necesarios para el utilizar de Groupware sincrónico.

Cabe destacar su relación con *CoopScan* en cuanto al modelo de tres capas se refiere y también al diseño de la arquitectura (replicada).

Esto demuestra que las tendencias en el desarrollo de aplicaciones colaborativas se han mantenido relativamente estables a través del tiempo, complementándose con el soporte a la movilidad. También se observa que a pesar de haber evolucionado, *DISCIPLE* todavía presenta problemas con el tema de la colaboración espontánea. Como se ha mencionado, el modelo de lugares (places) obliga a los usuarios a reunirse en una ubicación específica, independientemente de si está relacionada con la ubicación física o con el tema a tratar.

Otra de las razones para el soporte inadecuado de colaboración espontánea según los autores del artículo [WW1999], es que la maquina virtual de Java corre como un proceso en modo de usuario, sin afectar el manejador de ventanas (window manager) u otras aplicaciones, a diferencia de los sistemas de colaboración basados en sistemas de ventanas compartidas, en donde toda la interacción entre el usuario y el sistema operativo pasa a través del sistema de ventanas, permitiendo que cualquier aplicación sea colaborativa en cualquier momento.

La implementación de *DISCIPLE* sobre dispositivos móviles no parece factible debido a los requerimientos de memoria y almacenamiento que demandan los componentes Java utilizados para su diseño. Sin embargo, existe el framework *Manifold* [MI2002] diseñado por Ivan Marsic que soluciona este problema, permitiendo manejar la heterogeneidad en ambientes colaborativos sincrónicos.

1.1.12 DreamTeam. Roth Jörg & Unger Claus nos presentan su arquitectura DreamTeam, cuyo objetivo es el de desarrollar, evaluar y ejecutar aplicaciones colaborarivas sincrónicas en ambiente distribuido y heterogéneo [RJ2000]. Dreamteam tiene como característica adicional que tiene en cuenta problemas como inestabilidad de la red y conexiones de bajo anche de banda. El prototipo de la arquitectura se probado actualmente con dos aplicaciones: un web browser colaborativo y una herramienta para el diseño colaborativa.

Según [RJ2000], existen 3 tipos de aplicaciones colaborativas:

- Aplicaciones colaborativas inconcientes (Collaboration unaware applications): no ofrecen servicios colaborativos, generalmente son aplicaciones mono usuario y corren en ambientes compartidos.
- Aplicaciones colaborativas concientes (Collaboration aware applicactions): desarrolladas para ambientes cooperativos, pero sus servicios colaborativos son difíciles de desarrollar.

- Aplicaciones colaborativas transparentes (Collaboration transparent applications): proporcionan servicios colaborativos usando un nivel superior de servicios de una estándar de ambientes colaborativos.

DreamTeam se puede clasificar en la tercera clase, pues es una plataforma para desarrollar y ejecutar aplicaciones sincrónicas, colaborativas y transparentes.

✓ *Generalidades de la arquitectura.* El ambiente de DreamTeam permite desarrollar aplicaciones cooperativas como si fueran aplicaciones mono usuario. Este ambiente consiste de tres partes: el ambiente de desarrollo, el ambiente de ejecución y el ambiente de simulación.

Según [RJ2000], para lograr este funcionamiento se debe primero analizar varios aspectos importantes que son los que deciden como se va a construir la arquitectura:

La arquitectura: puede ser implementada de forma centralizada o descentralizada. Sin embargo ambas tienen sus ventajas y desventajas. Por ejemplo según [RJ2000], en las arquitecturas cooperativas centralizadas el manejo de mensajes es mucho más fácil, así como también los eventos, y el acceso a los datos, pues pueden ser sincronizados más fácilmente. Sin embargo el hecho de que sea centralizada, trae algunos problemas como son cuellos de botella y el punto único de falla.

En las arquitecturas descentralizadas, la sincronización y serialización deben ser manejadas por un protocolo más especializado, pero la disponibilidad y el ancho de banda pueden ser puntos a favor.

Manejo de sesiones: en una arquitectura descentralizada el manejo de las sesiones es otro problema que se debe contemplar. El originador de la sesión, debe definir el perfil de la sesión y generarla, para que así otros se puedan unir a ella. Si por ejemplo el originador de la se va, se debe mantener la sesión entre los demás participantes sin que nadie más se una a ella, o se utilicen otros mecanismos para elegir otro originador o coordinador como lo proponen [BD2001] con su arquitectura YCab.

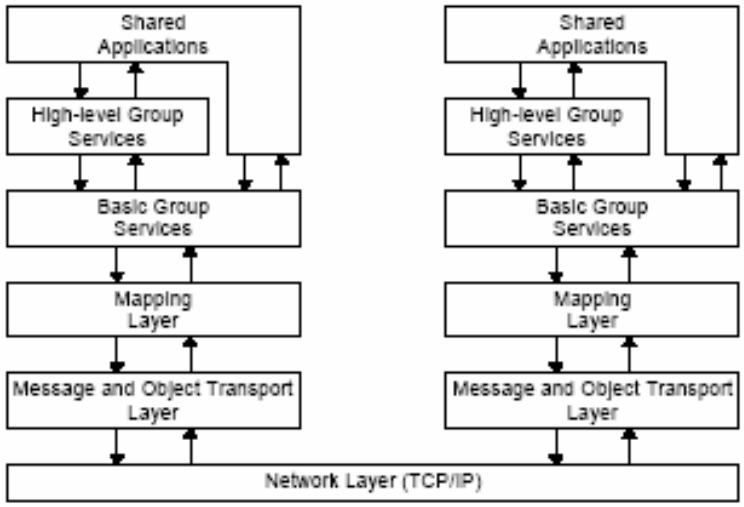
WYSIWIS (What You See Is What I See): este paradigma define que el espacio de trabajo debe ser el mismo para todos los participantes del sistema.

Modos de aplicación: Se debe permitir que las aplicaciones puedan correr tanto en ambiente compartidos como privados.

Distribución de la información: Garantizar una eficiente distribución de la información para todos los participantes en el sistema.

Sobre el protocolo de comunicaciones los autores encuentran que los problemas más comunes son: establecer conexiones, broadcast de eventos y sincronización de datos. En la arquitectura DreamTeam se propuso el siguiente protocolo de comunicaciones explicado en la figura 1-11.

FIGURA 1-11. Protocolo de comunicaciones de DreamTeam.



En la figura anterior se observa que la última capa es la encargada del nivel de comunicación y es basado en TCP/IP. La siguiente capa *Message and object transport layer,* proporciona los servicios de transporte elementales para el unicast y el multicast. La siguiente capa *Mapping layer,* se puede ver como un multiplexador/demultiplexador. El mecanismo de mapeo contiene la lista de todas las aplicaciones de una sesión y sus recursos. Luego se continua con *Basic group services layer,* que es la capa más baja a la que pueden acceder las aplicaciones compartidas; esta capa proporciona los servicios de multicast RPC, semáforos distribuidos y objetos compartidos. Por último, está el *High-level Group Services Layer,* que mapea los servicos básicos a los más complejos.

Como se había mencionado anteriormente, DreamTeam es una arquitectura con tres ambientes. El primer ambiente es el *ambiente de desarrollo,* que básicamente según [RJ2000], sirve para que se puedan desarrollar las aplicaciones compartidas en Java.

El segundo es el *ambiente de ejecución,* el cual se debe establecer para iniciar sesiones y las aplicaciones compartidas. Otros servicios que se inician en este ambiente son el *Log manager,* que recolecta información relevante y la guarda en un archivo; el *Connection manager,* que se activa durante la ejecución de una sesión y maneja la comunicación entre las aplicaciones compartidas. Otro servicio que se activa durante este ambiente es el *Session manager,* el cual maneja los perfiles de las sesiones, inicia, detiene sesiones, y permite la entrada y salida de las sesiones. El *Transfer manager,* quien es el encargado del trasporte de los datos, éste es usado en background y tiene baja prioridad; por último, el *Archive manager,* que puede ser visto como una pequeña base de datos.

El tercero es el *ambiente de simulación,* que sirve para probar el software con condiciones extremas, como por ejemplo cuellos de botella y conexiones de bajo ancho de banda. DreamTeam tiene este ambiente de simulación y se llama DNS (DreamTeam Network Simulator), que permite iniciar muchos ambientes de ejecución en una sola máquina, así como también las aplicaciones compartidas.

✓ *Análisis de la arquitectura DreamTeam.* DreamTeam es una plataforma para desarrollar aplicaciones colaborativas en PC de escritorio, y que tiene en cuenta distintos problemas como las conexiones de bajo ancho de banda, como las que podrían tener en entornos educativos distantes.

Más allá de los beneficios de la arquitectura que han sido probados con dos aplicaciones y que más adelante se estudiarán con más detalle, lo que nos enseña esta arquitectura es cómo atacar problemas estructurales, es decir cuales deben ser las principales consideraciones para realizar arquitecturas colaborativas concientes y transparentes como lo es DreamTeam.

Aunque ésta arquitectura no es orientada a dispositivos móviles como si lo es QuickStep [RJ2001], proporciona varias ideas para responder a los problemas colaborativos, como por ejemplo cuando se analizó el ambiente de ejecución de la arquitectura se puede observar una gran cantidad de servicios, que pueden ser llevados al entorno móvil.

Una de las grandes enseñanzas que nos deja DreamTeam es el proceso de selección del tipo de distribución de la arquitectura. A lo largo del análisis se pudieron observar los pro y contra de las arquitecturas centralizadas y descentralizadas.
Por lo tanto independientemente de si esta arquitectura es para entornos móviles o no, se puede observar que los servicios no son muy distintos a los de otras arquitecturas orientadas a dispositivos móviles; y DreamTeam así lo refleja. Perfectamente se podría hacer una implementación de esta en entornos móviles, con algunas variaciones y nuevos métodos de comunicación, pero la esencia de los servicios puede mantenerse intacta.

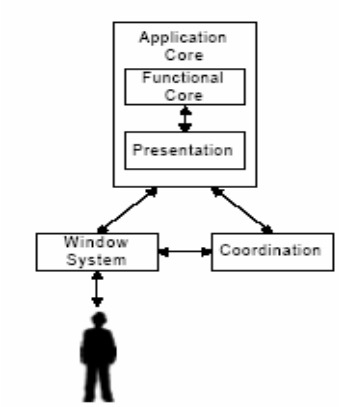
1.1.13 A Taxonomy for Synchronous Groupware Architectures. Según como explica [RJ2000A], hoy por hoy la distribución de las arquitecturas para groupware sincrónico, se pueden clasificar en centralizadas, replicadas o híbridas. Sin embargo una arquitectura que es perfecta para un escenario no lo es para otro. Por esta razón [RJ2000A], propone realizar una clasificación más extensa. El modelo propuesto se basa en 5 tipos de arquitecturas de donde se generan múltiples combinaciones. En especial, para las plataformas que soportan aplicaciones colaborativas concientes [RJ2000], se les puede clasificar en las arquitecturas híbridas.

Sin embargo, el problema yace en que este concepto es muy general, porque las arquitecturas híbridas tienen diferentes etapas de colaboración, que deben ser soportadas por la plataforma, por ejemplo preparar la sesión, entrar a una sesión, correr aplicaciones colaborativas, almacenamiento de artefactos, entre otras; y cada uno de estas etapas debería tener su propia distribución de arquitectura. Por ejemplo, los artefactos se almacenan en un servidor central, pero la aplicación colaborativa corre replicadamente [RJ2000A].

✓ *La taxonomía propuesta.* La taxonomía que presenta [RJ2000A], se debe estudiar solamente para describir la distribución de las arquitecturas. Ésta taxonomía consiste en dos partes: *el esquema de aplicación y el esquema de distribución*. El primero de ellos define los componentes de las aplicaciones de Groupware y el segundo describe cómo

estos componentes pueden ser distribuidos a través de muchos sitios. De acuerdo con la figura 1-12 (siguiente página), que describe el esquema de la aplicación, se puede ver que este se divide en tres componentes:

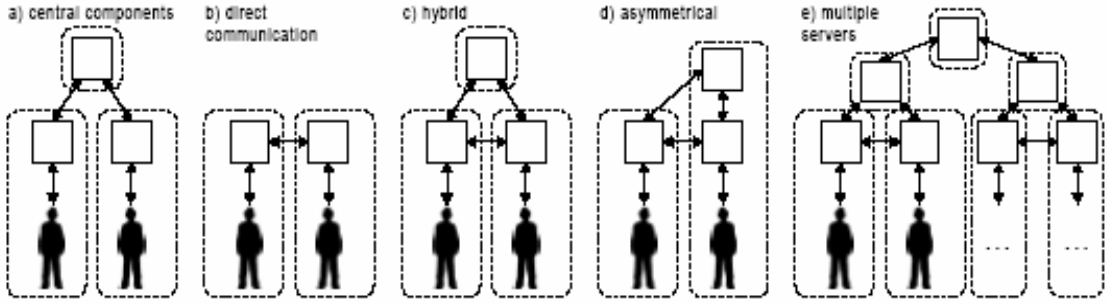
FIGURA 1-12. Esquema de la aplicación en la taxonomía de Jörg Roth.



El primer componente, *application core,* presenta la funcionalidad de la aplicación y se divide a su vez en los componentes *functional core* y *presentation.* El segundo componente es *window system,* que despliega las ventanas de la aplicación y recibe los eventos del usuario. Y el tercer componente es *coordination,* quien es el responsable de ejecutar la aplicación en un entono distribuido; tiene tareas como el control de la concurrencia y sincronización de las entradas del usuario.

Con base a este componente, nace el esquema de distribución. La figura 1-13 muestra las 5 distribuciones posibles:

FIGURA 1-13: Esquema de distribución en la taxonomía de Jörg Roth.



En la figura anterior los rectángulos representan uno o más de los componentes del esquema de la aplicación y los recuadros punteados, representan distintos sitios dentro de una red. Los sitios si son usuarios podrían llamarse *peers,* y si son computadores *servers.*

Según [RJ2000A], las 5 básicas taxonomías para las arquitecturas son las siguientes:

- Arquitecturas con componentes centralizados: estas arquitecturas por lo menos tienen un componente centralizado y los peers no se conectan unos con otros.
- Arquitecturas con comunicación directa: estas arquitecturas no tienen un componente central en ninguna parte. Todos los peers se interconectan entre si.
- Arquitecturas híbridas: estas arquitecturas tienen al menos un componente centralizado que permite la directa comunicación entre los peers.
- Estructuras asimétricas: no tienen un componente central, pero la distribución de los componentes entre los peers, no es simétrica.
- Múltiples servidores: estas arquitecturas usan más de un servidor, por ejemplo los componentes centralizados están distribuidos en más de un sitio.

Más allá de cual de las distribuciones es la mejor, la solución para escoger una distribución de la arquitectura nace del problema de Groupware que se quiera atacar como los explica [RJ2000A] "la escogencia de la distribución de la arquitectura, está fuertemente influenciada por el escenario en el cual va a trabajar. Una arquitectura puede funcionar muy bien en un escenario, pero tendrá muchas desventajas en otro". Para finalizar [RJ2000A], propone algunos criterios que se deben tener en cuenta en la elección de una distribución de arquitectura.

✓ *Criterios técnicos.* En escenarios donde operen peers a través de Internet, las arquitecturas de comunicación directa son más convenientes que una arquitectura con componentes centralizados o una arquitectura híbrida. Si por el contrario los sitios están en una LAN las demoras en la red pueden olvidarse un poco y las arquitecturas con componentes centralizados tienen más ventajas.

✓ *Criterios organizacionales.* Las arquitecturas con componentes centralizados e híbridos requieren de un servidor central, el cual debe ser instalado y administrado. Esto presupone un nuevo factor que es el económico, pues los servidores son costosos. Y si ésta es la perspectiva, las arquitecturas asimétricas y de comunicación directa son más económicas.

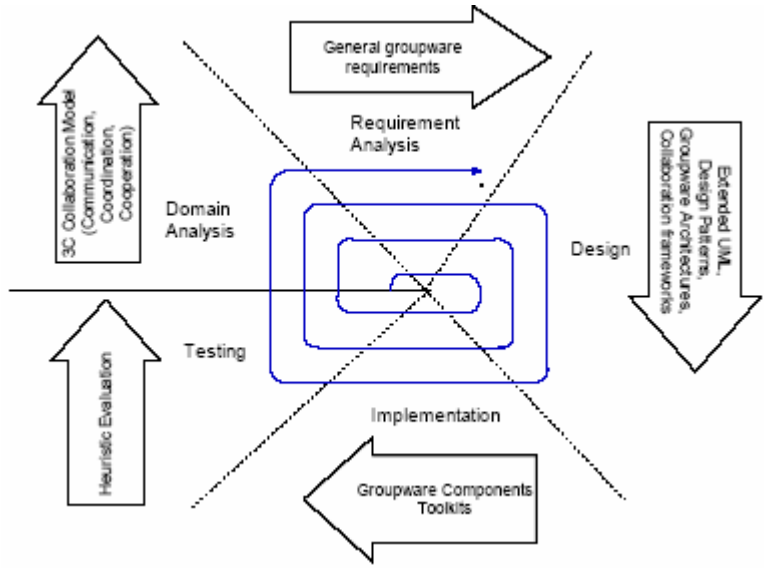
✓ *Criterios relacionados al desarrollo de Groupware.* La forma más fácil de integrar aplicaciones a escenario de grupo es utilizar aplicaciones of-the-shelf, es decir aplicaciones existentes, y así evitar costosos desarrollos. Pero si se va a realizar aplicaciones colaborativas concientes [RJ2000], este desarrollo no se puede obviar y esto afectaría los costos.

Las arquitecturas centralizadas permiten manejar la administración de datos mucho más fácil que las arquitecturas replicadas y otro ejemplo es que las arquitecturas con componentes centralizados usan algoritmos más fáciles que aquellas arquitecturas con comunicación directa, desde el punto de vista de la consistencia de los datos.

1.1.14 Towards an Engineering Approach for Groupware Development. Marco Aurelio Gerosa, presenta la idea de utilizar los principios de la ingeniería de software como una aproximación de ingeniería al Groupware, para formular aproximaciones sistemáticas y disciplinadas al desarrollo y mantenimiento del groupware.

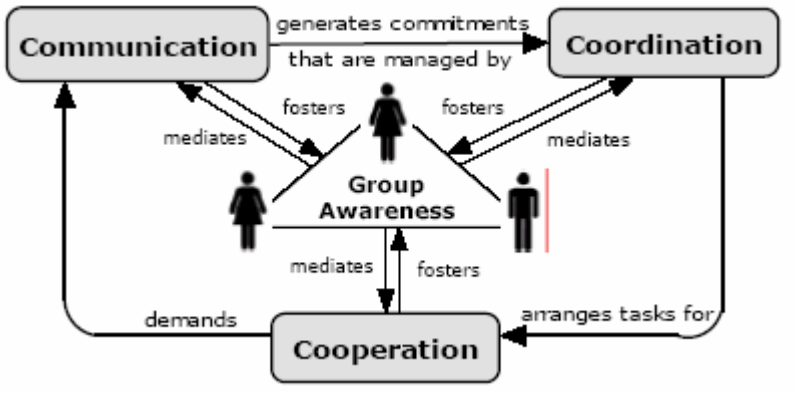
Este enfoque esta basado según [GM2003], en un modelo de colaboración llamado 3C, el cual analiza la colaboración en términos de *la comunicación, coordinación y la cooperación.* El enfoque se soporta en el modelo de espiral de la ingeniería de software, que se puede observar en la figura 1-14.

FIGURA 1-14. Ciclo de desarrollo del Groupware.



En la figura 1-15 se puede observar, como se interrelacionan estos tres conceptos en el Groupware.

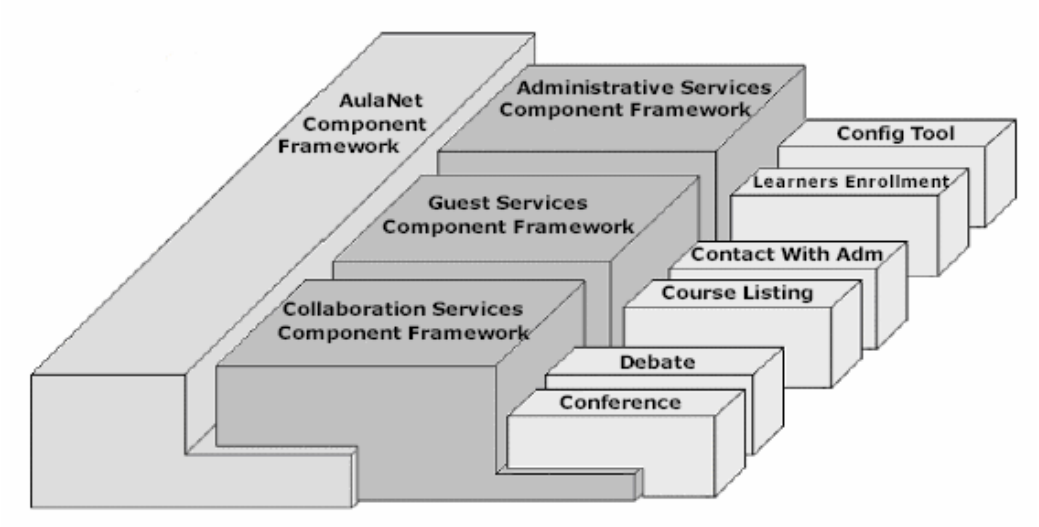
FIGURA 1-15. Modelo colaborativo 3C.



De acuerdo al modelo, la *comunicación* transmite y registra información, para que el grupo interprete los mensajes y actualice sus compromisos y conocimiento. Luego mediante la *coordinación,* el grupo garantiza la total realización de los compromisos. La *cooperación* organiza al grupo para evitar la pérdida de la comunicación y se encarga de los esfuerzos entre el grupo para realizar una tarea en el orden correcto [GM2003].

Tomando como base el modelo de espiral de ingeniería de software y combinándolo con el modelo de colaboración 3C, [GM2003] introduce una arquitectura basada en componentes para ambientes educativos AulaNet. La figura 1-16 expone la arquitectura AulaNet.

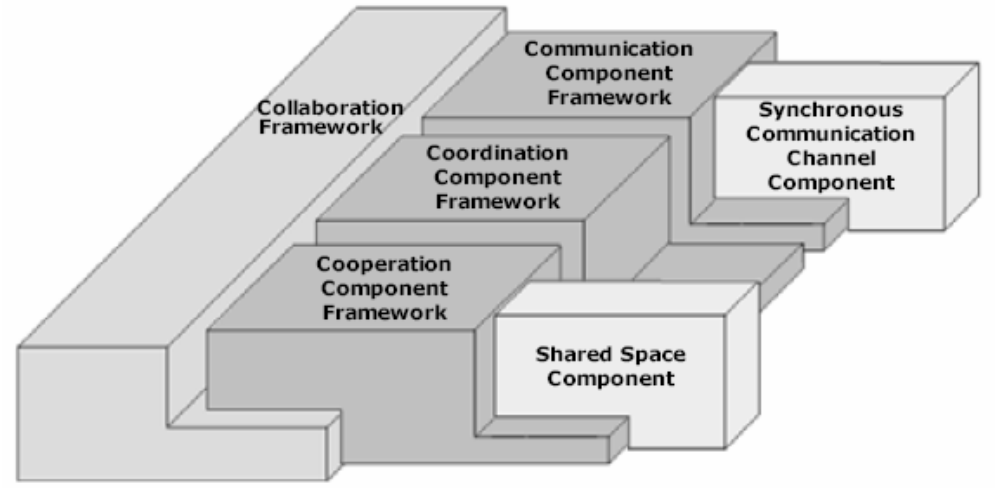
FIGURA 1-16. Arquitectura AulaNet.



AulaNet Framework define las funcionalidades generales a todos los servicios, como la administración y el manejo de datos compartidos. Actualmente esta se divide en tres grupos: Servicios colaborativos, servicios administrativos y servicios de invitados.

Con el fin de lograr la interacción con el modelo colaborativo 3C, los servicios son implementados basados en una arquitectura de componentes como la que se puede ver en la figura 1-17:

FIGURA 1-17. Arquitectura del servicio de debate.



En el servicio de debate se utilizan protocolos de comunicación sincrónicos y un espacio compartido que es utilizado por el framework de cooperación, con elementos que permitan el conocimiento (awareness). Y para el servicio de coordinación, con el fin de dar un mejor soporte a las actividades de integración, se implementaron el Floor Control, orden de participación y la agilidad de bloquear espacios compartidos.
Basados en el modelo 3C, con el fin de colaborar, los individuos deben debatir ideas (comunicación), para estar en sintonía con otros participantes del grupo (coordinación) y operar juntos en un espacio compartido (cooperación) [GM2003].

1.2 ANÁLISIS GENERAL DE LAS ARQUITECTURAS ESTUDIADAS

El crecimiento de la computación móvil durante los últimos años requiere de ambientes computacionales y de comunicación capaces de adaptarse a las nuevas necesidades de los usuarios tanto particulares como corporativos.

De igual manera, además de soportar la movilidad, las organizaciones modernas han dado mayor importancia a las labores grupales, y en consecuencia, a todas aquellas herramientas de software que les ayuden a desempeñar mejor este tipo de tareas.

Para cumplir con éste objetivo, muchas de las arquitecturas existentes tanto para dispositivos móviles como para groupware, poseen ciertas características en común que constituyen la base sobre la cual se implementan los diversos servicios que cada una presta.

Estas características son:

1. *P2P*: Las arquitecturas P2P facilitan la colaboración ad-hoc, y compartir información de manera distribuida sin la necesidad de tener un servidor específico, como sucede en arquitecturas de tipo Cliente/Servidor tradicionales. Sin embargo, en algunas oportunidades se pueden mezclar los dos paradigmas dependiendo de los servicios y ambientes computacionales que se desee soportar.

   Como se pudo apreciar anteriormente lo que decía [RJ2000A], una distribución de arquitectura podría estar bien para un escenario específico, pero para otro no sería tan buena. Por esta razón, se debe examinar bien el modelo de distribución de arquitecturas para implementar los servicios dado que algunos se desempeñan mejor cuando la arquitectura es centralizada, como por ejemplo el repositorio de datos, y otros podrían ser más fácil implementarlos con otro tipo de distribución de arquitectura.

2. *N-Tier (Multicapas)*: La mayoría de las arquitecturas definen como mínimo tres capas fundamentales, en las cuales se diferencia claramente la lógica, modelo de datos y presentación de las aplicaciones. En varios casos, esta división obedece específicamente a la implementación del patrón MVC. Dependiendo del grado de especialización de cada arquitectura, estas capas pueden subdividirse en otras dedicadas específicamente a soportar servicios colaborativos.

3. *Protocolos*: En las arquitecturas orientadas a comunidades virtuales, se definen protocolos orientados al manejo de grupos y usuarios. Dentro de este grupo de protocolos, podemos destacar en términos generales los siguientes: protocolos para detección de presencia, formación de grupos, mensajería, manejo de eventos y protocolos para el manejo de la información de manera concurrente.

4. *Servicios*: Con base en los protocolos mencionados anteriormente, también se definen servicios esenciales para la colaboración.

   Se identifican los siguientes:
   - Servicios de presencia.
   - Servicios de comunidad y de usuarios individuales.
   - Servicios de mensajería (asincrónica y sincrónica).
   - Servicios de publicación y suscripción de eventos o información para el grupo.
   - Servicios de consultas y búsquedas distribuidas.
   - Servicios de control de acceso y seguridad.

5. *Group Awareness (Conocimiento del grupo)*: El conocimiento del grupo se logra a través de los servicios de la arquitectura, pero en muchos casos, éste conocimiento es bastante limitado. Se debe proporcionar conocimiento más allá de la ubicación geográfica y el grupo de personas de confianza. Esto se puede lograr a través del manejo de historiales de encuentros entre miembros, consultas o trabajos y de frecuencia de conexión al grupo.

   La información histórica obtenida proporciona un valor agregado para la colaboración, y junto con los widgets[6], conforman un medio bastante completo de información al usuario.

6. *Arquitectura replicada de Groupware*: Las arquitecturas orientadas a Groupware tienen como factor común la implementación de una arquitectura replicada.

   Esto significa que cada usuario ejecuta una instancia de la aplicación compartida. Las entradas son distribuidas desde la ventana de cada usuario hacia las demás instancias. Las salidas se envían solamente al sistema local.

   Las arquitecturas replicadas permiten modificar la vista de la aplicación de manera relativamente fácil y soporta de mejor manera la heterogeneidad. Sin embargo, presenta problemas para poder mantener la consistencia en todas las instancias de la aplicación que se estén ejecutando.

7. *Modelo de colaboración*: Se debe definir un modelo de colaboración para las aplicaciones de Groupware. En términos generales, las arquitecturas de Groupware están basadas en el principio de las 3C: Comunicación, Coordinación y Cooperación. En este punto podríamos mencionar a los agentes de software cuyo propósito puede ser el de ayudar a la cooperación; se podría pensar en utilizar los

[6] Elemento de la Interfaz Gráfica de Usuario (GUI), que proporciona información o un medio específico para que el usuario interactúe con el sistema operativo y/o la aplicación en uso.

agentes para que apoyen algunos de los servicios mencionados anteriormente como por ejemplo, las búsquedas distribuidas.

8. *Manejo de sesiones*: este quizás es uno de los puntos más críticos en la arquitectura. Se debe garantizar que uno de los participantes inicie la sesión; una vez haya iniciado la sesión, otros participantes se pueden unir a la ella y colaborar. En el caso en que el coordinador la abandone, se debe garantizar de alguna forma o bien que los participantes continúen en dentro de la misma, pero que nadie más se les pueda unir, o que entre los participantes se vuelva a escoger a otro coordinador, que maneje la sesión.

Cada uno de los puntos mencionados anteriormente se relaciona directamente con las arquitecturas estudiadas a lo largo del capítulo. A continuación se presenta la tabla 1-1, en donde se asocian las características identificadas como esenciales para el proyecto, junto con las arquitecturas que las proporcionan:

TABLA 1-1. Características de las arquitecturas estudiadas.

	P2P	*N-Tier*	*Protocolos*	*Servicios*	*Group Awareness*	*Arq. Replicada de Groupware*	*Modelo de Colaboración*	*Sesiones*
DACIA	x	X				X		
DISCIPLE	X				X	X	X	X
DreamTeam	X	X	X		X	X	X	X
eNcentive	X	X			X			X
MOTION	X	X		X	X			X
Proem	X	X	X	X				
QuickStep		X		X	X			X
YCab		X	X		X		X	X
SALSA			X	X	X			
CoopScan				X		X	X	
RCSM					X		X	
A Taxonomy for Synchronous Groupware Architectures						X	X	
Towards an Engineering Approach for Groupware Development						X	X	

2. DISEÑO CONCEPTUAL DE LA ARQUITECTURA

En este capítulo se definirá principalmente el modelo conceptual de la arquitectura; para lograr este objetivo se presentarán varios conceptos que son la base de la arquitectura como son el paradigma de groupware soportado en el principio de las 5C, los servicios cooperativos soportados por los grupos volátiles y algunas definiciones que permitirán entender claramente el modelo propuesto.

Durante el desarrollo de todo el capítulo se integrará la arquitectura MAD [BA2004], que proporcionará la infraestructura necesaria para desarrollar la arquitectura propuesta, aportando los mecanismos para manejar las limitaciones inherentes a los dispositivos móviles. Adicionalmente MAD se enfocará hacia el manejo de los recursos de la arquitectura propuesta para así aproximar ésta a un contexto de groupware.

Finalmente, se explicará cada una de las capas que compone la arquitectura, definiendo sus características y funciones. El resultado final será una solución que permita crear aplicaciones de groupware con un enfoque más orientado hacia las personas y que aproveche las capacidades que ofrecen los agentes de software para el trabajo cooperativo.

2.1 GROUPWARE Y EL PARADIGMA DE LAS 5C

El groupware se define como el conjunto de hardware y software que ayudan a las personas a trabajar de manera colectiva sin importar su contexto individual (ubicación, medio, tiempo, etc). Se puede clasificar de dos maneras: el groupware sincrónico, que permite colaboración en tiempo real, con sus usuarios geográficamente distribuidos y el asincrónico que permite colaboración remota, pero no necesariamente al mismo tiempo.

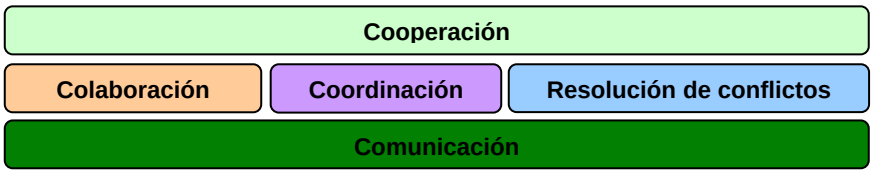
Con la popularización de distintos dispositivos móviles y fijos como las PDA, los celulares, smartphones, computadoras portátiles, desktop (en adelante se hará referencia a los dos tipos solo como *dispositivos*) y tecnologías como Bluetooth, o Wi-Fi que eliminaron los cables permitiendo la movilidad, podemos extender las capacidades que actualmente brinda el groupware y aprovechar todas estas nuevas tecnologías. Sin embargo, existen limitaciones para aplicar el groupware en estos dispositivos. Los mayores problemas se presentan en la intermitencia de la señal de las tecnologías inalámbricas y las limitaciones inherentes a los dispositivos móviles.

En cuanto a las personas, no hay mecanismos explícitos para la resolución de conflictos en el groupware tradicional, entre otras. Los usuarios deben estar pendientes en muchos casos del seguimiento de sus tareas. Para mitigar estas falencias se puede enfocar el groupware como un sistema multi-agente [FJ1999], basándose en el paradigma de la cooperación de Ferber que se fundamenta en la colaboración, la coordinación y la resolución de conflictos apoyada en todo momento por la comunicación.

Con esta visión se define el paradigma de las 5C, que es el modelo de cooperación entre personas definido para la arquitectura Ayllu, el cual es una extensión del paradigma de las 3C de Ferber, en donde la cooperación estará apoyada en todo momento en la comunicación que es la base principal del groupware orientado hacia las personas. Con este enfoque lo que se desea es representar a las personas, en adelante *usuarios*, como si fueran agentes humanos que realizan sus tareas cooperativas por medio de la ayuda de los agentes de software, dando como resultado una arquitectura para groupware orientada hacia las personas, capaz de tomar decisiones por el usuario sin necesidad de una constante interacción de éste y manejando a su vez los problemas de los dispositivos móviles.

En la figura 2-1 se muestra el modelo propuesto del paradigma de las 5C; a continuación se explicarán con más detalle cada uno de ellos:

FIGURA 2-1. Modelo del paradigma de las 5C.



La C*ooperación*, define las interacciones entre los diferentes usuarios de un sistema con base en las acciones que cada uno de ellos lleva a cabo para cumplir con sus objetivos. La cooperación se soporta en la colaboración, la coordinación y la resolución de conflictos y todas éstas se apoyan en la comunicación, ofreciendo las siguientes ventajas:

- Completar tareas imposibles de realizar de manera aislada dentro de un grupo de usuarios.
- Aumentar la productividad de cada usuario.
- Aumentar el número de tareas realizadas en un intervalo de tiempo para un usuario.
- Reducir el tiempo para realizar una tarea.
- Optimizar el uso de los recursos que puede utilizar un usuario.

La *Colaboración* se refiere a la forma en que se distribuyen las tareas entre los usuarios para conseguir un objetivo, puesto que existen situaciones en las cuales las labores no se pueden llevar a buen término solamente por un usuario, bien sea porque no posee el conocimiento, las capacidades o los recursos necesarios para su realización. El principio de colaboración permite que los usuarios establezcan relaciones con el fin de resolver una tarea de manera más eficiente.

La *Coordinación* es la manera en que se articulan las acciones individuales de cada usuario de tal manera que el todo final es coherente, aumentando el desempeño del sistema. Los planes y objetivos individuales de todo usuario se deben organizar de tal forma que no existan conflictos entre ellos, a través del uso de diferentes mecanismos como planeación, sincronización, regulación o por reacción.

La *Resolución de Conflictos*, es el conjunto de técnicas que permite manejar las diferencias que puedan tener los usuarios cada vez que quieren tener acceso a recursos compartidos o limitados. Los conflictos se pueden resolver por arbitramento o por negociación.

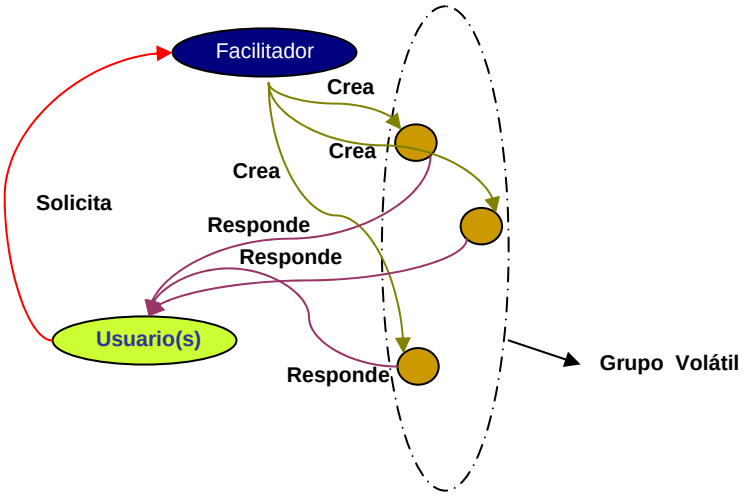
La *Comunicación* se define como el envío de un mensaje desde un *emisor* a un *receptor*; la información de este mensaje se encuentra codificada por medio de un lenguaje y es decodificada por el receptor al momento de su llegada. La información es enviada a través de un medio, y es válida dentro del contexto (situación) en la cual se ha iniciado el proceso de comunicación.

Basándose en este paradigma se propone la *Arquitectura Ayllu*, como solución a estos problemas y nuevo enfoque para groupware. *Ayllu*, que significa para la cultura indígena andina la base y el núcleo de la organización social. Tiene origen en la cultura Quechua y Aymará y significa *comunidad*, pero añade a este término la relación de *trabajo* de la tierra en forma *colectiva* y *solidariamente* [AP2000].

2.2 GRUPOS VOLÁTILES

Para resolver problemas no siempre se cuenta con las mismas personas. Los ambientes en este caso son cambiantes y en la formación de grupos pueden intervenir varias personas que pueden entrar o salir de estos grupos, por esta razón se ha definido el mecanismo de grupos volátiles. En la arquitectura Ayllu la base para la realización de tareas cooperativas y otras tareas no necesariamente de cooperación, es el mecanismo denominado *grupo volátil*; este permite conformar dinámicamente "un grupo de procesos" con el fin de llevar a cabo una labor conjunta; en la figura 2-2 se muestra el grupo volátil. Él mecanismo funciona de la siguiente manera:

FIGURA 2-2. Mecanismo de Grupo Volátil.



- Existe un *facilitador*, que se encarga de recibir las solicitudes del usuario.

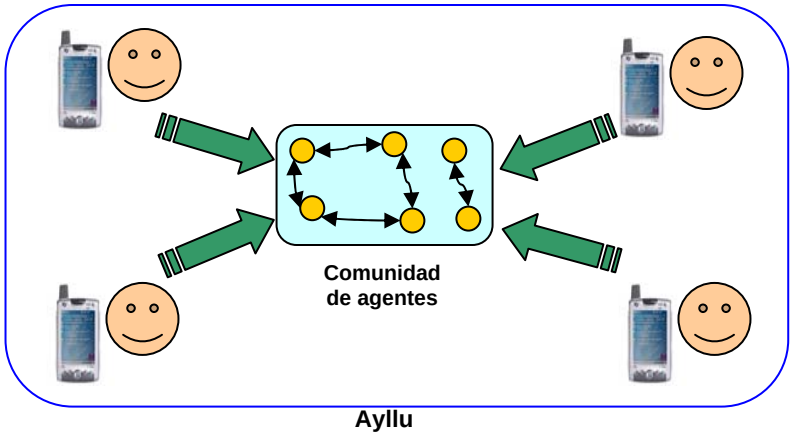
- El *facilitador* identifica otras entidades (procesos o usuarios) que estén en capacidad de resolver la solicitud del usuario.
- El *facilitador* crea los nuevos procesos y luego, se hace un bind para que cada uno de ellos sepa a que entidad pertenece.
- Finalmente los procesos se comunican, resuelven la solicitud y retornan la respuesta al usuario que la envió.

Dentro de Ayllu el mecanismo de grupos volátiles será utilizado para crear dinámicamente a los agentes que ejecutarán las tareas cooperativas que invoque un usuario por medio de un servicio cooperativo. El conjunto de éstos agentes se denominará *Comunidad de Agentes*, mientras que los usuarios que interactúan por medio de agentes representantes que apoyan la cooperación de la Comunidad de Agentes se denominará *el Ayllu*.

2.3 MODELO DE LA ARQUITECTURA AYLLU

El propósito de Ayllu es orientar el groupware con el enfoque del paradigma de las 5C, mediada por agentes de software y manejando a su vez los problemas de los dispositivos móviles, como se puede observar en la figura 2-3.

FIGURA 2-3. Visión de Ayllu



Teniendo en cuenta lo anterior, se ilustra en la figura 2-4 el modelo propuesto para la arquitectura.

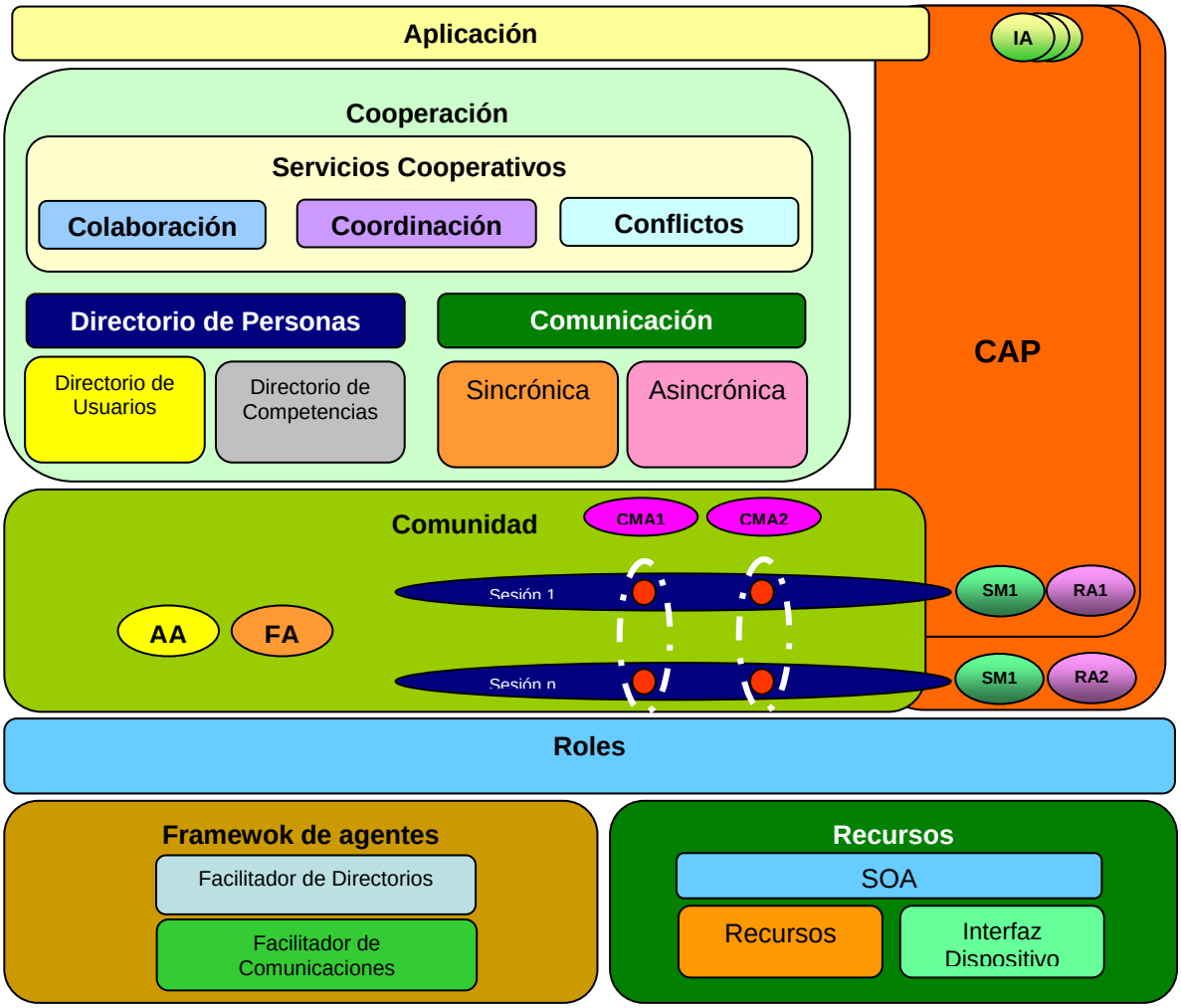
Ayllu consta de cinco niveles básicos y uno transversal; cada nivel se apoya en los niveles inferiores. Estos niveles son:

- Recursos & Framework de Agentes.
- Roles.
- Comunidad.
- Cooperación.
- Aplicación.
- CAP.

Capa de Recursos: en ella se concentrarán los artefactos que son paquetes de datos que pertenecen a los dispositivos móviles y que no son utilizados por éste debido a limitaciones de espacio para almacenamiento. Lo anterior representa un aporte en el campo de la computación móvil porque proporciona un mecanismo eficiente para manejar este inconveniente. En esta capa se encapsulan también otros tipos de recursos como son: archivos, bases de datos, entre otros.

La arquitectura MAD será la encargada de proporcionar el manejo de ésta capa, debido a que estos recursos son observados como un servicio que es accedido siempre a través de agentes que utilizan componentes de software tradicionales para realizar las operaciones *CRUD (Create, Read, Update, Delete),* proporcionando una visión de *SOA (Service Oriented Architecture)*. Esta visión aporta la solución para manejar las limitaciones de los dispositivos, debido a que este mecanismo permite el envío de datos que pueden ser procesados de manera remota, aliviando la carga de procesamiento en el dispositivo local.

FIGURA 2-4. Arquitectura Ayllu.



El Framework de Agentes: es el encargado de prestar todos los servicios necesarios para el manejo de agentes de software en las capas superiores. Entre estos servicios podemos encontrar localización, registro, comunicación entre agentes y contenedor de agentes (para el manejo del ciclo de vida).

La capa de roles: es en esencia un conjunto de abstracciones; un rol es una abstracción del comportamiento de un agente de software, que en consecuencia será una instancia del mismo. En esta capa es donde de definen las responsabilidades y capacidades que pueden tener los agentes dentro de Ayllu y cualquier tipo de rol que podrá ser empleado posteriormente dentro de la arquitectura. Se manejan particularmente estados, metas, entradas, actuadores y comportamientos para cada agente uno de los agentes que conforman la arquitectura.

La capa de comunidad: en esta capa es donde se manejan las sesiones de los usuarios y se da soporte a las tareas cooperativas de la capa inmediatamente superior; una *sesión* es el conjunto de agentes, que permiten a los usuarios realizar tareas cooperativas por intermedio de los grupos volátiles, junto con los agentes pertenecientes al *Punto de Acceso a la Comunidad CAP (Community Access Point)* de cada usuario. Los agentes residentes en la comunidad podrán interactuar entre sí soportados por la capa del framework de Agentes.

Punto de Acceso a la Comunidad CAP (Community Access Point): es el mecanismo de entrada al *Ayllu* y está conformado por un conjunto de agentes que conforman el medio a través del cual el usuario interactúa y se representa ante la comunidad de agentes proporcionado a su vez apoyo a la cooperación.

La capa de cooperación: es la encargada de permitir la creación de los servicios cooperativos, que son soportados por agentes en la capa de comunidad. La capa está orientada hacia la personas más que a los agentes de software y es donde se determina que agentes se han de crear para iniciar labores de trabajo cooperativo. En este nivel se proporcionan mecanismos de resolución de conflictos, colaboración y coordinación entre usuarios, representados mediante los servicios cooperativos, comunicaciones y directorios de personas.

La capa de aplicación: que simplemente representa cualquier construcción de software que emplee los servicios proporcionados por Ayllu para realizar groupware. A continuación se presentarán cada una de las capas anteriores explicando sus características y funciones con mayor detalle.

2.3.1 Framewok de agentes. En la arquitectura Ayllu, los agentes son los responsables de soportar los servicios de la cooperación, apoyándose en la capa de comunidad, que es donde se desplegarán los agentes. Con éste framewok se pretende poder crear a los agentes que manejarán los distintos servicios cooperativos de la arquitectura.

Esta capa trabajará soportando a las demás capas superiores. El framework que se utilice debe ser capaz de manejar un servicio de directorio de agentes, que podemos

descomponer en las páginas amarillas y las páginas blancas, donde los agentes se registran a si mismos y los servicios que saben ejecutar. También se debe garantizar la administración y control del ciclo de vida de los agentes.

Adicionalmente el framework debe proporcionar los protocolos de interacción para los agentes, es decir la comunicación que éstos utilizarán para realizar las tareas cooperativas, que generalmente es P2P asincrónica.

Para lograr una mayor abstracción de los agentes el framework debe proporcionar contenedores donde éstos puedan residir, ser buscados y encontrados; éstos no deben ser centralizados necesariamente, por el contrario, si son distribuidos se eliminaría el punto de falla y haría más robusta esta capa.

El framework debe estar implementado bajo los estándares FIPA con el fin de garantizar estandarización y posibilidades de interacción con otras plataformas de agentes.

2.3.2 Capa de recursos. Esta capa se ha inspirado en el modelo MAD [BA2004], debido a la aproximación que presenta: "emplear un proporcionador de servicios estático que realiza una actividad de coordinación encargándose de invocar a uno o varios servicios para poder recuperar la información" [BA2004], lo cual permite manejar las limitaciones de los *dispositivos* (móviles especialmente).

En la capa de recursos encontramos dos elementos principales: el repositorio de recursos y la interfaz física del dispositivo; ambos elementos son tratados como servicios y son vistos con una aproximación *SOA* para poder acceder a ellos y realizar las operaciones *CRUD* correspondientes.

El repositorio de recursos es donde se almacenan datos que no son utilizados por los dispositivos en cierto momento, con el fin de no sobrecargarlos cuando su capacidad de almacenamiento es reducida. Cada *usuario* tendrá un identificador único por cada artefacto que almacene en el repositorio para evitar manipulación de información por parte de usuarios no autorizados.

De igual forma, las interfaces físicas de cada dispositivo se encuentran ubicadas en esta capa con el fin de generar independencia y separar la lógica de aplicación de la presentación. Tanto el repositorio de recursos como las interfaces de los dispositivos se acceden a través de proporcionadores de servicios especializados para cada recurso. Esto quiere decir, que cada proporcionador invoca un(os) servicio(s) específico(s) con el fin de manipular información o de realizar algún tipo de procesamiento para aligerar la carga en el dispositivo que lo invoca.

2.3.3 Capa de roles. La capa de roles está enfocada principalmente hacia los agentes. Busca asociar responsabilidades a una clase genérica de agentes que saben cómo manejar un servicio en especial. En ella quedarán definidos todos los roles que se puedan utilizar en Ayllu; cada vez que se implemente un agente, se deberá instanciar uno de

estos roles, así el agente podrá adquirir las responsabilidades asociadas a un rol específico.

Esta forma de definir la capa de roles, permite que haya una mayor abstracción y que exista la posibilidad de crear nuevos roles relacionándolos con los servicios que se vayan implementado en Ayllu. Para el funcionamiento de las capas superiores, se han creado los siguientes roles predefinidos en Ayllu: el rol de administrador de comunidad y el rol representante, el rol administrador de sesión y el rol interfaz, rol de comunidad, rol administrador y el rol fabricador, los cuales serán introducidos a continuación y explicados en detalle en el contexto de las capas superiores.

✓ *Rol Administrador de Comunidad:* este rol define las responsabilidades y capacidades del agente manejador de comunidad CMA (Community Manager Agent), entre las que se encuentran gestionar la creación y destrucción (ciclo de vida) de los agentes de comunidad CA (Community Agents), que conforman las comunidades de agentes. Otra función es proporcionar un mecanismo de comunicación entre los agentes de comunidad CA de diferentes sesiones, que están colaborando para resolver una tarea cooperativa. Éste rol participa activamente en la conformación de los grupos volátiles actuando como el facilitador.

✓ *Rol Representante:* este rol define las responsabilidades y capacidades del agente representante RA (Representative Agent), entre las que se encuentran guardar información relacionada con el usuario, para no tener que preguntarle en todo momento sobre decisiones, acciones o eventos que se tengan que tomar en algunas situaciones a menos que sea estrictamente necesario, si éste agente no sabe cómo responder alguna petición, preguntará al usuario por medio del agente Interfaz IA (Interface Agent).

✓ *Rol Manejador de Sesión:* este rol define las responsabilidades y capacidades del agente manejador de sesión SM (Session Manager), entre las que se encuentran mantener una referencia de los agentes de comunidad CA creados en su sesión y actuar como un punto de entrada para las comunicaciones entre la capa de comunidad y el punto de acceso a la comunidad CAP. También es el encargado de manejar los protocolos para determinar los estados en los que pueden estar los CAPs de un usuario y en consecuencia localizar cual de ellos está activo, identificando el agente interfaz IA que lo maneja.

✓ *Rol Interfaz:* este rol define las responsabilidades y capacidades del agente interfaz IA, entre las que se encuentran separar la lógica de la presentación permitiendo soportar varios tipos de dispositivos y actuando como un proporcionador para acceder a los servicios de interfaz que se encuentran en la capa de recursos, adicionalmente se encarga del envío de mensajes de los agentes de la sesión hacia la aplicación y viceversa.

✓ *Rol de Comunidad:* este rol define las responsabilidades y capacidades del agente de comunidad CA, entre las que se encuentran participar activamente en los grupos volátiles como las entidades encargadas de realizar las tareas cooperativas. Avisar al agente manejador de comunidad CMA cuando acabe la tarea para que éste lo pueda destruir.

También puede realizar peticiones o enviar mensajes al agente manejador de sesión SM, para que éste retorne las respuestas adecuadas.

✓ *Rol Administrador:* este rol define las responsabilidades y capacidades del agente administrador AA (Administrator Agent), entre las que se encuentran realizar tareas como el logueo y autenticación de los usuarios al entrar al Ayllu. Manejar las operaciones CRUD de usuarios y competencias en la capa de cooperación donde se encuentran los directorios de personas.

✓ *Rol Fabricador:* este rol define las responsabilidades y capacidades del agente fábrica FA (Factory Agent), entre las que se encuentran autenticar que un usuario que desea invocar un servicio, tenga los permisos suficientes y crear dinámicamente los agentes manejadores de comunidad CMA de un servicio en particular.

2.3.4 Community Access Point – CAP. El punto de acceso a la comunidad CAP, permite el acceso de un usuario a los servicios cooperativos proporcionados por capa de comunidad. Posee tres elementos básicos: el agente interfaz IA, el agente representante RA y el agente manejador de sesión SM. Vale la pena destacar que dentro de la visión de sistema multi-agente empleada en éste proyecto, los usuarios poseen capacidades de decisión, autonomía y sensores que les permiten interactuar con su entorno, que es el objetivo del CAP.

El agente interfaz IA posee todas las características descritas por el rol i*nterfaz*, puesto que es una instancia del mismo. Su principal función es actuar como un sensor/actuador a través del cual los usuarios interactúan con el ambiente de cooperación proporcionado por Ayllu.

El agente representante RA, es una instancia del rol representante. Tiene otras responsabilidades como son gestionar las peticiones que le haga el agente manejador de sesión SM, para acceder a colaborar en una tarea cooperativa, cuando se necesita preguntar alguna información al usuario. Si el agente representante RA no sabe como responder, delegará la responsabilidad al agente interfaz IA.

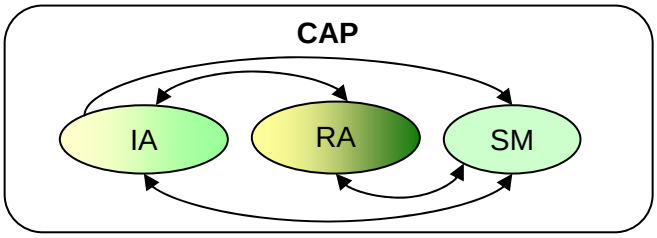
El agente manejador de sesión SM, es una instancia del rol manejador de sesión y es el encargado de atender cualquier solicitud que puedan hacer los agentes de comunidad CA que participan en una tarea cooperativa de un grupo volátil, así como autorizar finalmente al agente manejador de comunidad CMA sobre la participación del usuario en una tarea cooperativa.

En la figura 2-5 se ilustra la estructura del punto de acceso a la comunidad CAP mostrando las interacciones existentes entre cada uno de los agentes anteriormente explicados. Como se puede observar el agente interfaz IA se comunica con el agente representante RA y éste con el agente manejador de sesión SM cuando se está respondiendo a una petición hecha previamente por el agente manejador de sesión SM.

Otra interacción interesante es la que se puede observar del agente interfaz IA al agente manejador de sesión SM. En la interacción de una sola vía entre el agente interfaz IA y el

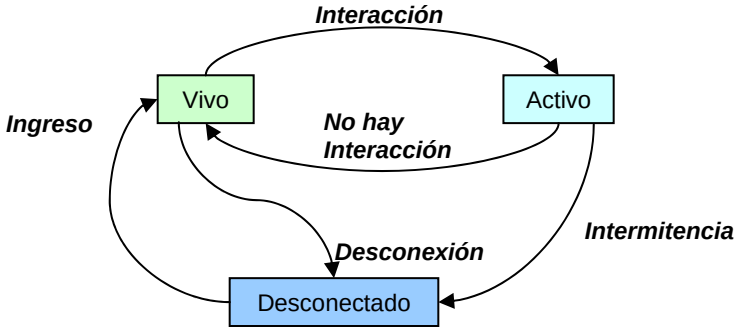
agente manejador de sesión SM se representa el reporte del estado del punto de acceso a la comunidad CAP. En la interacción de doble vía entre los mismos agentes se utiliza para el envío de mensajes cuando no hay necesidad de pasar por el agente representante RA.

FIGURA 2-5. Estructura del CAP (Community Acces Point).



El punto de acceso a la comunidad CAP es uno de los puntos más propensos a caídas inesperadas o desconexiones temporales, cuando se habla de dispositivos móviles, por esta razón Ayllu debe manejar algunos mecanismos para diferenciar los tipos de caídas temporales de las definitivas. Teniendo en cuenta lo anterior se maneja tres estados que dan soporte a los mecanismos para identificar el tipo de caída; en la figura 2-6 se pueden apreciar los eventos que los activan.

FIGURA 2-6. Estados del CAP.

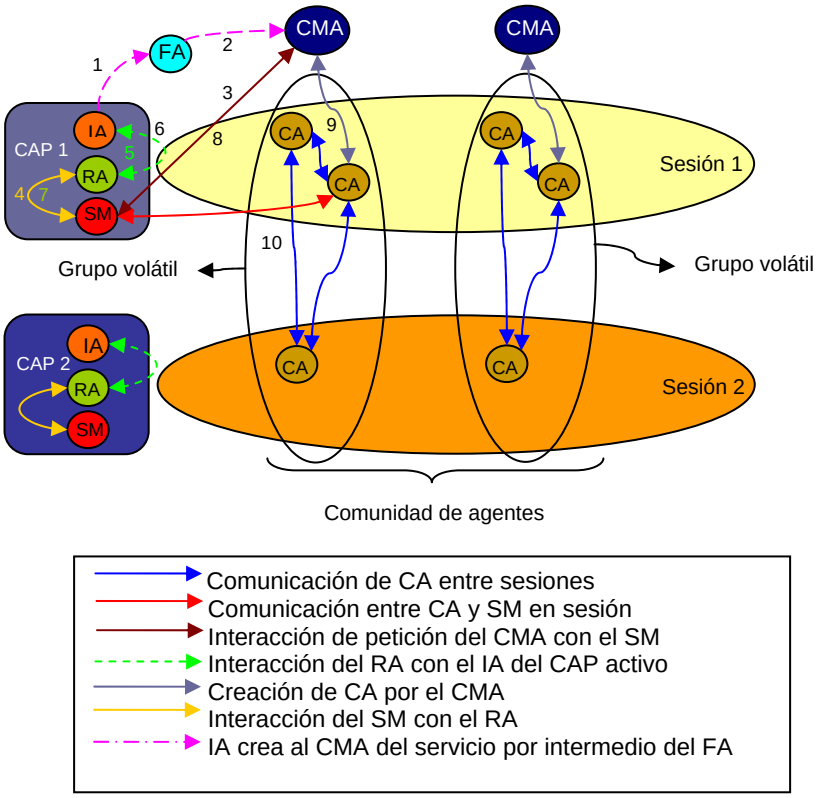


- *Vivo:* es un estado que indica que el punto de acceso a la comunidad CAP está presente en el Ayllu; en este estado el CAP busca a su agente manejador de sesión SM para empezar a reportarse. Solo se llega a este punto desde el estado *Desconectado* y por un evento de *Ingreso.*
- *Activo:* cuando se presenta una interacción del usuario con el punto de acceso a la comunidad CAP, éste pasa al estado *Activo*. Este estado permite saber que hay un usuario interactuando en el Ayllu de manera frecuente.
- *Desconectado:* el punto de acceso a la comunidad CAP queda en este estado cuando ocurre una caída del dispositivo causada por intermitencia en la señal o desconexión (que a su vez puede ser causada por falta de batería, problemas de red, entre otras razones) y se desconecta del Ayllu, esto quiere decir que se puede pasar del estado *Vivo* a *Desconectado* o de *Activo* a *Desconectado.*

2.3.5 Capa de Comunidad. La capa de comunidad es la encargada de contener las sesiones de los usuarios y conformar los grupos volátiles. En consecuencia da soporte a los servicios que ofrece la capa de cooperación.

En esta capa es donde se crea el agente manejador de comunidad CMA, quien es el encargado de crear los agentes de comunidad CA necesarios que colaboran para realizar una tarea cooperativa dentro de cada una de las *sesiones* de los *usuarios*. En la figura 2-7, se puede apreciar la creación de los grupos volátiles y los pasos que realizan los agentes para realizar una tarea cooperativa.

FIGURA 2-7. Representación de servicios cooperativos en Ayllu.



(1) Cuando se invoca algún servicio cooperativo el agente interfaz IA crea el agente manejador de comunidad CMA de ese servicio por intermedio del agente fábrica FA.

(2) Éste realiza la validación del usuario que invoca el servicio para saber si tiene los permisos necesarios para ejecutarlo. La creación toma lugar en la capa de comunidad y se inicia el proceso del grupo volátil.

(3) El agente manejador de comunidad CMA es el encargado de buscar a otros *usuarios*, en el directorio de personas de la capa de cooperación, que puedan colaborar en el servicio invocado para resolver una tarea cooperativa. Al haber detectado los posibles candidatos, el agente manejador de comunidad CMA realiza una petición al agente

manejador de sesión SM de cada una de las *sesiones* y les solicita permiso para colaborar en la realización de una tarea cooperativa.

(4) El agente manejador de sesión SM pregunta al agente representante RA si sabe sí el *usuario* desea participar en la tarea cooperativa.

(5) Si el agente representante RA no conoce la respuesta, se comunica con el agente interfaz IA, que está asociado al punto de acceso a la comunidad CAP activo, quien se encargará de preguntarle al *usuario* si desea aceptar la petición de participación.

(6) Luego el agente interfaz IA correspondiente retornará la respuesta al agente representante RA.

(7) El agente representante RA retornará la respuesta al agente manejador de sesión SM.

(8) Finalmente el agente manejador de sesión SM comunicará la respuesta al agente manejador de comunidad CMA.

(9) Si el agente manejador de comunidad CMA recibe una respuesta afirmativa a su petición por parte del agente manejador de sesión SM, inmediatamente creará por demanda los agentes de comunidad CA necesarios para cumplir con el servicio dentro de la(s) sesión(es) correspondiente(s), referenciando a su vez cada uno de estos al agente manejador de sesión SM respectivo. Éste proceso es repetido con cada uno de los candidatos que el agente manejador de comunidad CMA encontró en su búsqueda.

(10) Por último de esta manera queda conformado el grupo volátil cuando los agentes de comunidad CA empiezan a cooperar entre ellos para resolver la tarea cooperativa. Al finalizar la tarea el agente de comunidad CA asociado a la sesión del usuario que invocó el servicio cooperativo, retornará la respuesta a su agente manejador de sesión SM, quien la enviará directamente al agente interfaz IA activo; posteriormente los agentes de comunidad CA, notificarán al agente manejador de comunidad CMA que han finalizado la tarea y que deben ser destruidos, quién destruirá el grupo volátil.

2.3.6 Capa de cooperación. La capa de Cooperación es la encargada de proporcionar los servicios cooperativos básicos de Ayllu. Todos éstos están soportados por una serie de agentes de comunidad y por el agente manejador de comunidad CMA en la capa de comunidad, que es creado dependiendo del servicio. Sin embargo, el diseño es tan flexible que se permite adicionar más servicios de manera sencilla.

Los servicios de cooperación ofrecidos por esta capa son la abstracción de la colaboración, coordinación y la resolución de conflictos, proporcionándolos de manera explícita al implementar protocolos de interacción cooperativos entre los usuarios mediados por los agentes que conforman una comunidad. Uno de los agentes del punto de acceso a la comunidad que se puede colocar paralelamente en esta capa es el agente interfaz IA, quien es el encargado de actuar como un puente con la capa de aplicación, al permitir la invocación y participación de cualquier servicio cooperativo por parte de los usuarios.

La capa de cooperación está orientada hacia las personas. Este aspecto es consecuente con el paradigma de las 5C y visión de sistema multi-agente que se ha tomado como base para Ayllu. Además justifica plenamente la definición dentro del modelo de los conceptos de Colaboración, Coordinación y Resolución de Conflictos, los cuales no deben ser confundidos con los tradicionales orientados a agentes de software; estos están orientados hacia agentes humanos, lo cual puede aumentar la complejidad de los servicios cooperativos. A continuación se analizarán con más detalle los servicios básicos de esta capa.

Esta capa cuenta con un grupo de servicios cooperativos, que para facilidad de conceptualización fueron organizados en tres categorías que son las siguientes:

- Servicio de Directorios de Personas
- Servicios de Comunicación
- Servicios Cooperativos.

✓ *Servicio de directorios de personas:* el servicio de directorios de personas de la capa de cooperación, incluye servicios de directorio de usuarios y directorio de competencias que serán introducidos a continuación.

El directorio de usuarios proporciona el mecanismo para registrar y localizar a los distintos usuarios que se encuentren el Ayllu. En el proceso de registro se almacena la información del usuario y el agente manejador de sesión SM, quien conoce a los demás agentes del punto de acceso a la comunidad CAP.

El directorio de competencias ofrece los mecanismos necesarios para registrar y ubicar los servicios que prestan los usuarios de manera individual, como por ejemplo archivos para compartir, habilidades, conocimientos o saberes acerca de la solución de un problema.

El agente manejador de comunidad CMA de un servicio cooperativo específico es quien tiene la capacidad de acceder a estos directorios para buscar los usuarios potenciales con el fin de llevar a cabo una tarea cooperativa y luego poder crear a los agentes de comunidad CA que reunidos formarán el grupo volátil para resolver dicha tarea cooperativa.

Otro agente que tiene la capacidad de realizar operaciones en los directorios de personas es el agente administrador, quien por intermedio de una fachada que es controlada por éste último, puede realizar todas las operaciones administrativas en los directorios, como logueo y autenticación de usuarios y operaciones CRUD de usuarios y competencias.

Dentro del servicio de directorios se pueden encontrar otros servicios especiales como son el Grupo de Interés y Detección de Presencia y las Búsquedas Distribuidas que se explicarán a continuación.

Servicio de Grupo de Interés y Detección de Presencia: este servicio permitirá que los usuarios por medio de su punto de acceso a la comunidad CAP, creen organizaciones

donde se compartan intereses comunes y se busque resolver problemas o alcanzar un objetivo particular.

Este servicio permitirá manejar una lista de los contactos con la posibilidad de cambiar estados de disponibilidad de los usuarios (por ejemplo conectado, ausente, no disponible, ocupado, etc.) y detectar la presencia de usuarios cuando estos se encuentren dentro del Ayllu, notificando este suceso a los demás miembros de su grupo de interés.

Este servicio tiene como fin proporcionar el sentido de awareness y poder mantener el sentido de pertenencia dentro del Ayllu y el grupo de interés. Cabe anotar que el awareness no depende solamente de éste servicio; todos los demás servicios de la capa de cooperación pueden proporcionar otro tipo de información, mucho más especializada para cumplir con éste objetivo.

Servicio de Búsquedas Distribuidas: este servicio permitirá encontrar cualquier tipo de información o recurso dentro del Ayllu. La búsqueda se podrá realizar de manera parametrizable es decir, seleccionando los criterios que más se acomoden al tipo de recurso a buscar y a las necesidades de quien realiza la búsqueda.

✓ *Servicio de comunicación:* la capa de cooperación prestará el servicio de comunicación sincrónica y asincrónica. Para la comunicación sincrónica, Ayllu podría emplear ordenamiento total, con el fin de garantizar que todos los eventos y mensajes dentro del sistema lleguen en el orden estricto en que sucedieron o fueron enviados; todo lo anterior con el objetivo de mantener la integridad del flujo de trabajo dentro de las aplicaciones de groupware y evitar las inconsistencias.

Los algoritmos más comunes para conseguir ordenamiento total de los mensajes son los de contadores centralizados, timestamps y contadores distribuidos [CG2001]. Los dos primeros son mucho más eficientes en el aspecto de consumo de recursos de red, pero tienen la desventaja de ser centralizados, lo cual representa un cuello de botella y poca tolerancia a fallos.

En los algoritmos de contadores centralizados, cada miembro del grupo envía el mensaje de multicast, junto con un *id*. El secuenciador, se encarga de aumentar el *id*, anexarlo al mensaje y transmitirlo a todos los destinatarios. Los mensajes serán retransmitidos hasta que no se obtenga una respuesta de *ackknowledgement* por parte de los receptores. Luego de un tiempo sin respuesta, el mensaje se da por perdido.

Los algoritmos de timestamp son similares, pero el *id* es una estampilla de tiempo que es sincronizada por un proceso central con base en su hora local. Los algoritmos de contadores distribuidos son mucho más costosos porque presentan mayor latencia. Todos los procesos que reciben los mensajes, proponen números de secuencia según les van llegando y los devuelven al emisor, quien los usa para determinar el número de secuencia acordado. Esto nos lleva a concluir que se requieren tres mensajes entre emisor y receptor para acordar el *id*.

Existe otro tipo de comunicación sincrónica para grupos, que es sin ordenación de paquetes, que dará soporte a servicios como chat; de igual manera el servicio de comunicación asincrónica dará soporte a servicios como un foro, por ejemplo.

Todos estos servicios crean un cierto número de agentes en la capa de comunidad, pero en la capa de cooperación se muestran como la abstracción, es decir que en esta capa es donde realmente se ve el funcionamiento hacia las personas, mostrándolos por medio de la interfaz y permitiendo la interacción a través del punto de acceso a la comunidad CAP.
En Ayllu, la comunicación por defecto no contempla ordenamiento de paquetes con el fin de agilizar las comunicaciones entre los distintos dispositivos y demás elementos involucrados en la arquitectura. Sin embargo, si es necesario se puede implementar el servicio de ordenamiento según lo requieran los servicios cooperativos que se implementen.

A continuación se explicarán otros servicios especializados que se prestan para la comunicación.

Servicio de Suscripción y Notificación de Eventos: este servicio permitirá a los usuarios registrarse en listas de temas de interés para recibir y publicar mensajes según lo deseen permitiendo la cooperación entre las personas.

Servicio de Edición de Documentos Compartidos: este servicio permitirá que varios usuarios trabajen cooperativamente sobre un mismo recurso resolviendo conflictos en tiempo real (de manera sincrónica), como por ejemplo la escritura simultánea.

Servicio de Chat: este servicio permitirá que dos o más usuarios dialoguen en tiempo real, para compartir ideas. Este servicio podrá soportar a cualquier otro servicio cooperativo que pueda necesitarlo. No se limita a los miembros de un mismo grupo de interés, si no que es totalmente abierto a personas que no usen este servicio.

Servicio de Mensajería: este servicio permitirá el intercambio de mensajes entre varios usuarios de manera asincrónica, a diferencia del servicio de chat. Este servicio podrá soportar cualquier otro servicio cooperativo.

✓ *Servicios Cooperativos:* Ayllu proporciona una serie de servicios cooperativos que pueden ser utilizados por las aplicaciones que corran sobre ella. Estos servicios llevan a otro nivel la experiencia cooperativa del groupware mediante la utilización de agentes de software para apoyar ciertas tareas al interior de un equipo de trabajo.

Todos los servicios cooperativos de Ayllu se pueden apoyar en los servicios de directorios y servicios de comunicación, dado que estos últimos facilitan la realización de tareas cooperativas a través de los mecanismos estudiados anteriormente.

Mecanismos de Colaboración, Coordinación y Resolución de Conflictos: como se mencionó anteriormente los servicios cooperativos son una abstracción del paradigma de las 5C y se proporcionarán de manera implícita al momento de ser implementados. A continuación se mencionarán de manera general algunos aspectos sobre estos mecanismos, inspirados en los sistemas multi-agentes. Para más detalles referirse a al libro de Ferber, “Multiagent Systems: An Introduction to Distributed Artificial Intelligence” [FJ1999].

- Colaboración: La colaboración se refiere a la manera en que se distribuyen las tareas entre los agentes del sistema para poder conseguir un objetivo.

Bajo esta perspectiva, y teniendo en cuenta que los usuarios son el objeto principal de la colaboración dentro de Ayllu, se presentan las siguientes alternativas para manejar la colaboración entre personas:

- *Distribución centralizada de tareas por un negociador:* es el método más simple, en donde existe un negociador que posee una tabla de usuarios conocidos y sus respectivas capacidades para realizar una tarea.

  Cada vez que un usuario A_i necesita realizar una tarea cualquiera, pero no quiere o no puede hacerlo por sí mismo, le pide al negociador que encuentre a alguien más para realizar la labor.

  El negociador utiliza el *Servicio de Directorios* de la capa de cooperación para buscar a los usuarios que poseen las habilidades para cumplir con el requerimiento y, si algún usuario A_j responde bien sea de manera afirmativa o negativa, le informa al usuario A_i la respuesta.

- *Asignación mediante red de usuarios conocidos*: este mecanismo distribuido asume que cada usuario está al tanto de las habilidades, a través del directorio de competencias, de los usuarios que conoce. Existen dos variaciones de éste tipo de asignación: directa y por delegación.

  En la asignación directa, una tarea *T* solo puede ser ejecutada por un usuario que sea conocido de manera directa por otro que la inicia; en la asignación por delegación, es posible encadenar varios usuarios que no se conocen directamente. De esta manera, un usuario *A* conoce solo al usuario *B* que a su vez, conoce al usuario *C* quien puede realizar una tarea de *A*, contrario a lo que sucede en la asignación directa, en donde *B* sería el único capaz de realizar las tareas que ofrezca *A.*

- *Distribución de tareas por Contract Net:* es un mecanismo distribuido de asignación de tareas basado en un protocolo de oferta y demanda.

  Usa un protocolo para poder establecer contratos en mercados públicos. Existen dos roles básicos definidos en la capa de roles: *Manager* (responsable de monitorear la ejecución de una o varias tareas y procesar los resultados) y Contractor (responsable directo de realizar las tareas).

  Para establecer un contrato se tienen que seguir cuatro pasos básicos:

  1. Un *manager* avisa sobre la existencia de una tarea a través de broadcast general, limitado o comunicación P2P.
  2. Los posibles contratadotes reciben el anuncio, y deciden si pueden realizarla; luego envían una oferta.
  3. El *manager* evalúa las ofertas recibidas y envía un mensaje de “recompensa” a uno o más oferentes.
  4. El proceso de negociación puede ser recurrente.

- Coordinación: Con el objetivo de completar una tarea, los agentes, de un sistema multi-agente, deben realizar una serie de acciones, que aunque parezcan

suplementarias, son necesarias para que al final la tarea sea completada de una manera satisfactoria. Éste es el concepto que presenta Ferber [FJ1999], al hablar sobre la coordinación de acciones.

Otra aproximación interesante es la que propone Farias Damian y Pérez Adith [FD2002], cuando explica la coordinación: “la coordinación permite detectar los conflictos para evitarlos, así como las sinergias para aprovecharlas y de esta forma mejorar el desempeño del sistema”.
Ahora, puesto que se ha definido que las personas sean vistas como agentes dentro del contexto de este proyecto, se podría aplicar la misma filosofía que proponen los sistemas multi-agente, para manejar la coordinación de acciones entre las personas.

Sean agentes o personas vistas como agentes, la coordinación de acciones es necesaria, porque estos necesitan información y resultados que otros agentes pueden proporcionar, por ejemplo en los workflows. Otra razón importante son los recursos que están en juego y que pueden conllevar a situaciones de conflictos, que deberán ser solucionadas a través de la coordinación de acciones y mecanismos de resolución de conflictos.

También encontramos la optimización de costos, eliminado acciones innecesarias que demanden trabajo improductivo y por último, buscar que los agentes tengan objetivos separados, pero a la vez interdependientes, los cuales podrían ser compatibles o incompatibles, provocando situaciones de conflicto o sinergia que deben ser coordinadas. Ferber [FJ1999], identifica cuatro mecanismos básicos para la coordinación de acciones; a continuación se explicarán brevemente cada uno de ellos:

- *Coordinación por Sincronización:* esta es la forma más elemental de coordinación, tiene sus orígenes en los sistemas distribuidos; el método describe que la secuencia de acciones, conllevará a una necesaria sincronización en la ejecución, donde se debe manejar la concurrencia de muchas acciones y verificar que el resultado de las operaciones sea coherente; lo anterior se evidencia en los protocolos de commit como el 2PC o 3PC [CG2001].

- *Coordinación por Planeación: e*sta técnica es la más usada en inteligencia artificial. Consiste en realizar las acciones en dos pasos importantes. En el primero, se llevan a cabo un conjunto de acciones que tienen un objetivo y producen planes de acción. En el segundo, se selecciona alguno de los planes y se ejecuta.

  Los planes deben ser coordinados, puesto que los agentes pueden caer en situaciones de conflicto. En resumen, estos planes son buenos para realizar la coordinación, pero tienden a ser ineficientes en situaciones extremamente complicadas.

- *Coordinación Reactiva: e*sta técnica dice, que en vez de planear un conjunto de acciones y sus interacciones antes de actuar, es mejor implementar

mecanismos de coordinación basados en agentes reactivos, donde se usa prioritariamente la percepción-acción del agente. En resumen, es actuar cuando algo sucede, basándose en las capacidades de percepción del agente.

- *Coordinación por Regulación:* Ésta técnica generalmente es más usada en la práctica donde no se requiera una coordinación tan rigurosa. Consiste simplemente en crear un conjunto de reglas de comportamiento, que permitan eliminar posibles conflictos.

- Resolución de conflictos: En muchas ocasiones, los recursos que necesitan los usuarios no son ilimitados. Esto conllevará a que surjan situaciones en donde se presenten conflictos por esos mismos recursos.

  Estas situaciones de conflictos son generadas por la interacción de los usuarios entre si y se dan generalmente por la necesidad de obtener algún recurso al mismo tiempo que otro. Por esta razón se deben manejar estas situaciones, mediante técnicas de resolución de conflictos.

  Los métodos más conocidos son por negociación y por arbitramento. El método por arbitramento [FJ1999]; mediante el arbitramento y la negociación se consigue que los usuarios no entren en conflictos que pueden deteriorar las prestaciones del sistema completo. En el arbitramento existen un conjunto de leyes para todos los agentes a las que deben atenerse y suele existir un árbitro que se encarga de resolver los conflictos. En la negociación, cuando dos usuarios entran en conflicto, tratan de resolverlo mediante un acuerdo bilateral.

2.3.7 Capa de Aplicación. En la capa de aplicación se encuentran los aplicativos que harán uso de los servicios cooperativos proporcionados por Ayllu y que tienen como objetivo lograr una meta por medio de trabajo en grupo.

Estos aplicativos deberían ser soportados por dispositivos móviles, porque estos son el objetivo principal de la arquitectura, pero es posible extender hacia aplicaciones que estén dirigidas a equipos tradicionales de cómputo. Para lograr esto Ayllu se vale del agente interfaz IA, que es el agente capaz de desplegar las interfaces dependiendo del tipo de dispositivo al que se esté dirigiendo y mostrar mensajes de las capas inferiores a la aplicación que manejan los usuarios, separando la lógica de la presentación.

Dado que los servicios cooperativos en Ayllu son genéricos, es decir, no existe un lenguaje específico predeterminado, se posee una mayor flexibilidad al permitir que cada aplicación defina los elementos/componentes requeridos en sus propios términos.

3. IMPLEMENTACIÓN DE LA ARQUITECTURA AYLLU

Luego de haber realizado un análisis de distintas arquitecturas y con base en ello haber definido un modelo conceptual para la arquitectura Ayllu, el siguiente paso es entrar a definir el proceso de ingeniería de software, que en este proyecto hace parte de la metodología de investigación, con el cual se precisarán las etapas más importantes contempladas en el modelo de cascada las cuales son análisis, diseño, implementación y pruebas para la arquitectura, puesto que en este momento se entra a realizar el software como tal. Dado que éste no es un proyecto de ingeniería de software, la rigurosidad de cada uno de los entregables de las etapas de la metodología de cascada, no es la misma que se esperaría en un proyecto puramente de ingeniería de software.

Se ha decidido trabajar con el modelo de cascada porque el desarrollo de este software se ha basado en el diseño conceptual de la arquitectura que si ha tenido varias iteraciones en donde se ha revisado, depurado y se han definido requerimientos que en éste capítulo serán formalizados. Por esta razón la metodología en cascada es la que más se ajusta al proceso de ingeniería de software contemplado para la metodología de investigación.

Este capítulo tiene como fin mostrar los resultados de cada una de las etapas del modelo de desarrollo mencionado anteriormente. De igual manera se explicará el criterio para la selección de herramientas de desarrollo, el diseño general de la arquitectura, basado en los requerimientos definidos en el capítulo de diseño conceptual, y se realizará una descripción detallada de la implementación a través de los diagramas de interacción para cada uno de los agentes y componentes principales de la arquitectura.

3.1 DESCRIPCIÓN GENERAL DE LAS ETAPAS DE IMPLEMENTACIÓN

Para el desarrollo de la arquitectura Ayllu se definieron cuatro etapas siguiendo el modelo en cascada. Las actividades específicas para cada etapa son:

- *Análisis*: en esta etapa se precisan con detalle los requerimientos tanto funcionales como no funcionales que debe tener la arquitectura Ayllu, teniendo en cuenta el *Estudio de arquitecturas para el entorno móvil* y *Diseño Conceptual de la arquitectura Ayllu* donde se llevó a cabo el análisis previo a la implementación de Ayllu.

  Previamente se estudiaron las diferentes arquitecturas para dispositivos móviles, groupware, comunidades virtuales y agentes de software existentes destacando las características más importantes de cada una; se propuso el modelo de la arquitectura Ayllu con el fin de poder definir sus características generales y los requerimientos funcionales y no funcionales de la misma. Así mismo, se realizó la selección de las herramientas de desarrollo, las cuales se sustentan posteriormente dentro de éste capítulo.

Como parte del proceso de ingeniería de software en la etapa de análisis se presentarán formalmente los requerimientos funcionales y no funcionales de la arquitectura Ayllu, destacándolos por medio de diagramas de caso de uso y un explicación concisa del los requerimientos para cada una de las capas del modelo de la arquitectura Ayllu.

- *Diseño*: en esta capa se explicará formalmente las interacciones de cada uno de los agentes involucrados en la arquitectura Ayllu, la explicación estará soportada por los diagramas de interacción de los agentes, que muestran claramente los comportamientos de los agentes en el ambiente.

- *Implementación*: luego de haber establecido los requerimientos con base en el modelo conceptual, se procede a traducirlos a código empleando las herramientas descritas más adelante en éste capítulo, para más información sobre las herramientas estudiadas ver el Anexo1 (*Comparación de herramientas de desarrollo de aplicaciones para dispositivos móviles y agentes de software*).

  En esta sección se explicarán la estructura de la arquitectura Ayllu, detallando la forma en que se implementaron los servicios y componentes adicionales de software que se utilizan. También se mostrará la organización de la implementación de la arquitectura mediante los diagramas de paquetes.

- *Pruebas: e*n esta sección se mencionarán las pruebas de funcionalidad mínima que la arquitectura Ayllu satisfizo correctamente. Posteriormente en el capítulo *Caso de Estudio,* se mostrarán las pruebas realizadas con el caso de estudio seleccionado y se presentará formalmente un estudio de las pruebas realizadas.

3.2 SELECCIÓN DE HERRAMIENTAS DE DESARROLLO

Después de haber analizado varias herramientas disponibles para el desarrollo de aplicaciones para dispositivos móviles y agentes de software, se optó por emplear la plataforma BESA [GE2003] para el desarrollo de agentes de software y la arquitectura MAD [BA2004] para desarrollar todo lo concerniente a dispositivos móviles. A continuación se explicará cada una de ellas.

3.2.1 BESA (Behavior-oriented, Event-driven and Social-based Agent Framework).

BESA es un framework que soporta el diseño e implementación de sistemas multiagente (*MAS*) propuesto por Enrique González, Jamir Ávila y César Bustacara de la Pontificia Universidad Javeriana [GE2003]. El modelo abstracto de *BESA* se basa en tres conceptos fundamentales: una arquitectura modular de agentes orientada a comportamientos, una técnica de control orientada a eventos que implementa un mecanismo de selección tomado de la programación concurrente y un soporte basado en mecanismos sociales para lograr la cooperación entre agentes. La arquitectura está soportada sobre tres niveles importantes el nivel de los agentes, el nivel social y el nivel del sistema.

BESA ha sido desarrollado completamente en java, para aprovechar las ventajas de portabilidad que ofrece entre diferentes plataformas. Su entorno es orientado a objetos, soporta multi-threading y tiene gran facilidad en las comunicaciones.

Para el segundo semestre del año 2004 se terminó el desarrollo de la versión de *BESA* para dispositivos móviles, llamada *BESA-CE (Behavior-oriented, Event-driven and Social-based Agent Framework Compact Edition),* que proporciona la misma funcionalidad, aunque con la diferencia de que los directorios no están replicados en cada contenedor debido a las limitaciones de los dispositivos móviles.

Adicionalmente la fuente de información es mucho más directa, pues se cuenta con la asesoría presencial de las personas que están desarrollando el framework, otro punto a favor, es que la plataforma ya es funcional y se utilizó en el desarrollo de la arquitectura MAD [BA2004], que hace uso de esta extensión de *BESA* para proponer una arquitectura basada en componentes distribuidos para dispositivos móviles. Con MAD se puede dar solución a otros requerimientos de Ayllu, con lo que se tiene una mayor ventaja en el momento de su desarrollo.

✓ *Conceptos fundamentales del modelo abstracto de BESA.* En esta sección se explicará más a fondo lo referente a Behavior-oriented, Event-driven y Social-based, que son los conceptos fundamentales del Framework *BESA* [GE2003].

- *Behavior-oriented:* cuando se construyen los agentes la complejidad es un aspecto crítico, puesto que los agentes deben ser autónomos y racionales. El comportamiento se entiende como una entidad que encapsula lo necesario para garantizar el cumplimiento de un propósito bien definido. Esto conlleva a que se puedan crear relaciones semánticas entre los agentes, para que se formen sociedades cooperativas, donde se obtiene una conducta racional, dando como resultado que en *BESA*, un agente esté compuesto de un conjunto de comportamientos concurrentes.

- *Event-driven:* los agentes se encuentran inmersos en ambientes llenos de eventos. El evento se puede interpretar como una señal que le informa al agente que algo ha sucedido en el ambiente y que debe reaccionar ante esa situación de acuerdo a lo que conoce.

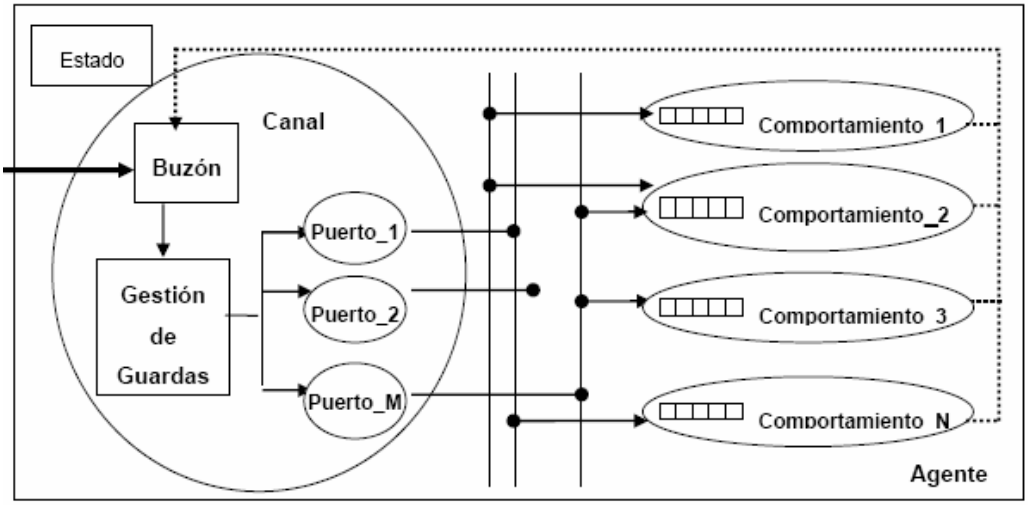
- *Social-based:* este enfoque permite estudiar al sistema multiagente como una organización de entidades que interactúan. Ésta organización a su vez puede ser simplificada varias veces hasta llegar a encontrar que el agente tiene bien definida una meta en la organización y una interfaz con los otros agentes del ambiente.

✓ *Capas de la arquitectura BESA*. Como se mencionó anteriormente BESA se compone de tres niveles principales, que son los siguientes: el nivel de agentes, el nivel social y el nivel de sistema. A continuación se explicará cada uno de ellos [GE2003].

- *Nivel de agentes:* básicamente este nivel trata los comportamientos y eventos a los cuales reacciona un agente y explica cómo es el flujo de acciones cuando sucede un evento y el comportamiento que el agente sigue debido a este. Ver figura 3-1.

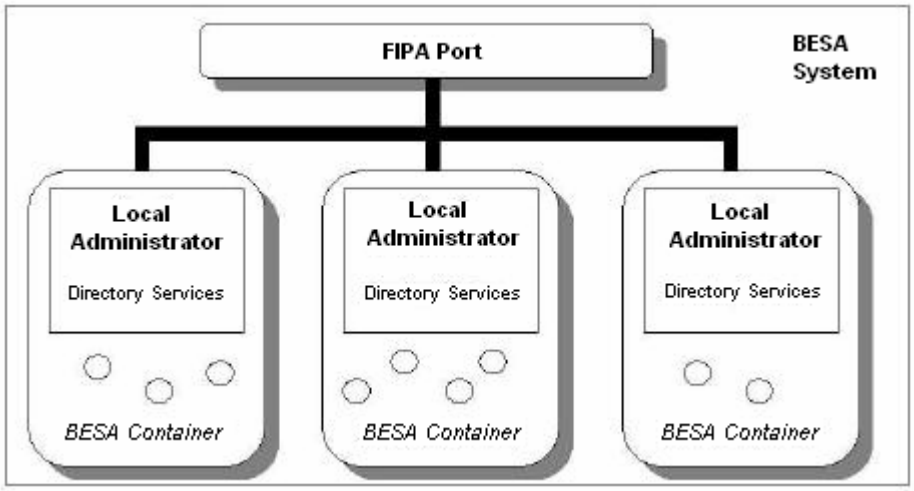
- *Nivel social*: para incorporar abstracción en las capas intermedias, BESA propone una organización jerárquica, en la cual un conjunto de agentes interactúan con un agente mediador que se encuentra un nivel más arriba en la jerarquía. El mediador facilita el trabajo cooperativo entre los agentes de la organización y también actúa como un receptor y lanzador de eventos a los agentes concernientes. Otra función del mediador es ser un puente para un conjunto de agentes capaces de representar un servicio.

FIGURA 3-1. Arquitectura interna de los agentes en BESA.



- *Nivel de sistema:* en este nivel es donde se definen los ciclos de vida y la administración de los agentes. Un sistema de agentes implementado con el framework de *BESA*, puede ser visto como un sistema distribuido compuesto por uno o varios contenedores que comparten un único puerto FIPA, el cualque recibe comunicación de otros sistemas que utilicen un sistema de comunicación de agentes *ACL (Agent Communication System)*. Ver figura 3-2.

FIGURA 3-2. Nivel de sistema en BESA.



✓ *Ventajas de BESA*

- Portabilidad al haber sido implementado sobre el lenguaje Java. Adicionalmente tiene los beneficios de contar con un enfoque orientado a objetos.
- Es una plataforma que sigue la especificación propuesta por FIPA.
- La plataforma es eficiente para aplicaciones concurrentes y distribuidas.
- Cuando hay un cambio en los ciclos de vida del agente o hay un cambio en cualquiera de los directorios, se activa un mecanismo de replicación a los administradores locales, para garantizar que se maneje la misma información y que esta sea consistente.
- Se garantiza que los eventos y comportamientos sean manejados de manera óptima por el agente, porque se maneja un mecanismo de guardas que decide según el evento que comportamiento debe realizar el agente.
- Se maneja una única entrada de comunicación, que hace más fácil la administración de los mensajes.

✓ *Desventajas de BESA*

- En los sistemas donde los agentes son continuamente creados y removidos, el mecanismo de replicación puede traer un significativo aumento de overhead en la red.
- No se encuentra mucha documentación, aparte de la que se pueda hallar en la Universidad Javeriana.

3.2.2 Arquitectura MAD (Mobility Supported by Agent-Based Devices). La arquitectura *MAD* es el producto de un proyecto de investigación realizado en la Universidad Javeriana, por los estudiantes Alejandro Botero, Hugo Giraldo y Alexandra Moyano y dirigido por el Ingeniero Enrique González Phd. El proyecto fue finalizado en el año 2004.

MAD, fue concebida como una arquitectura de software híbrida P2P y P2MP [BA2004] para dispositivos móviles, que debería tener presente las principales limitaciones sobre estos y también para dispositivos estáticos; adicionalmente esta diseñada para trabajar sobre ambientes cerrados, entendiendo lo anterior como un lugar dentro de unas fronteras físicas.

La arquitectura se desarrolló, pensando en ser una nueva capa que hereda del framework *BESA* [GE2003] y *BESA-CE*, para la funcionalidad de los agentes de software que realizan todas las tareas que la arquitectura involucra.

A continuación se explicarán los principales conceptos de la arquitectura, dando un especial énfasis en los modelos propuestos.

✓ *Generalidades de la Arquitectura.* MAD, es una arquitectura, enfocada hacia los dispositivos móviles, teniendo en cuenta las limitaciones inherentes de estos aparatos como son: limitada capacidad de procesamiento, poca capacidad de almacenamiento, reducida duración de la batería y la intermitencia de la comunicación [BA2004]. La

arquitectura esta orientada hacia los ambientes cerrados, es decir lugares encerrados por unas fronteras físicas y limitados por el alcance de la señal inalámbrica, como por ejemplo un aeropuerto, un estadio, un centro comercial, un hospital o un museo.

La arquitectura presenta dos mundos, entendiéndose como mundo a la reunión de varios dispositivos de características similares, ubicados en algún lugar. El primer mundo es el estático, en el cual los dispositivos permanecen fijos en un sitio geográfico determinado. Éstos pueden ser servidores que alojan recursos como bases de datos, archivos o web services.

El segundo mundo es el dinámico, donde los dispositivos son móviles, éstos pueden cambiar de posición continuamente; para la arquitectura es transparente el cambio de sitio, puesto que utilizan un roaming, que proporciona esta característica [BA2004].

Otras características adicionales son el uso de una estructura de comunicaciones peer to peer y la utilización de un repositorio de información en el cual se almacenan paquetes de datos llamados artefactos pertenecientes a los dispositivos móviles que no son utilizados en ese momento [BA2004].

Se maneja el concepto de aplicación, que es un conjunto de procesos que colaboran entre si para cumplir una meta en común. Está repartida mediante agentes en los mundos estático y dinámico. También, usando el concepto anterior, un proceso del mundo dinámico puede realizar un procesamiento pesado hacia el mundo estático, creando una imagen suya, llamada representante, quien es el encargado de realizar el procesamiento y enviar la respuesta a la aplicación móvil del mundo dinámico que requirió el procesamiento.

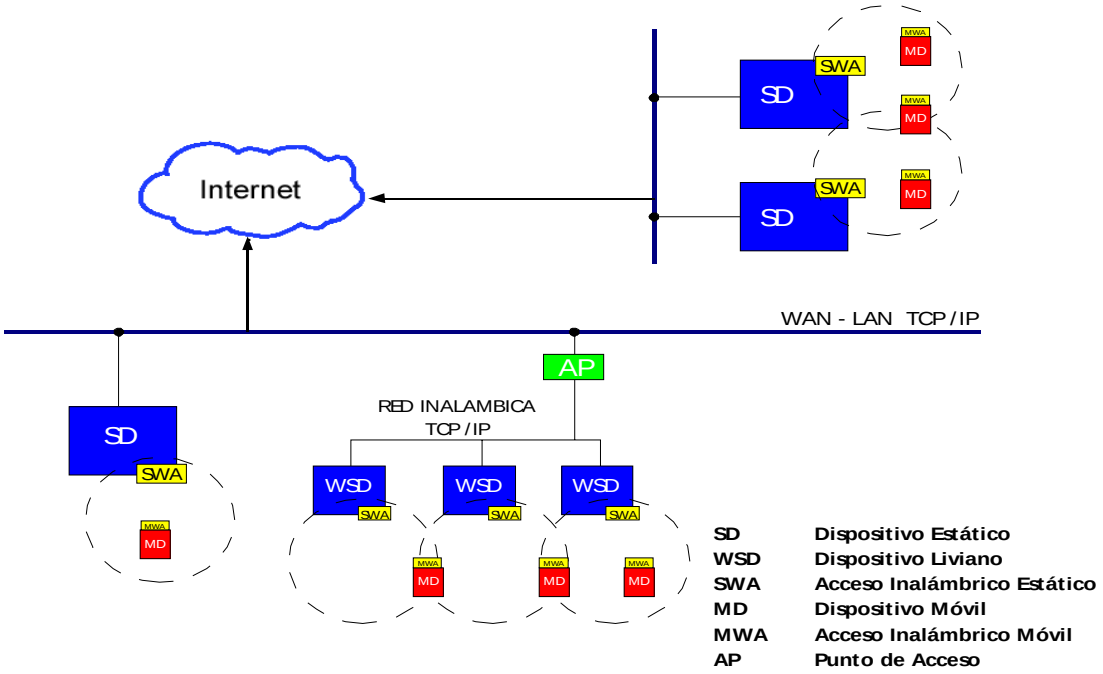
Para manejar el problema de la reducida capacidad de almacenamiento, MAD proporciona un repositorio para los artefactos que no utilice en un momento dado algún dispositivo móvil; este repositorio se encuentra distribuido en el mundo estático. Para administrar los artefactos, se utiliza un comprobante que contiene un identificador único, un identificador del dispositivo móvil propietario del artefacto, fecha y tiempo máximo de almacenamiento. Estos comprobantes se almacenan en directorios para una mayor facilidad de acceso [BA2004].

Para manejar los problemas de la reducida duración de la energía y la intermitencia, MAD propone utilizar un componente que funciona como un Proxy. Este tiene las tareas de verificación del canal de comunicación entre los dos mundos, manejar un “repositorio” de los mensajes cuando estos no se puedan enviar de una lado a otro, notificar a los interesados cuando se reestablezca la comunicación del canal y manejar un time out para los mensajes, para evitar que se quede infinitamente entregando un mensaje por fallas en la comunicación [BA2004].

✓ *Modelo Físico*. La arquitectura da la flexibilidad para que el mundo estático pueda estar compuesto por uno o más tipos de redes, que se comunican entre sí a través de un punto de acceso. De esta manera, se forma una red LAN o WAN utilizando el protocolo TCP/IP. Ambos mundos tienen un acceso inalámbrico, encargados de manejar los problemas de comunicación y el roaming de los dispositivos móviles. En la figura 3-3, se puede apreciar la distribución del modelo físico de la arquitectura MAD [BA2004].

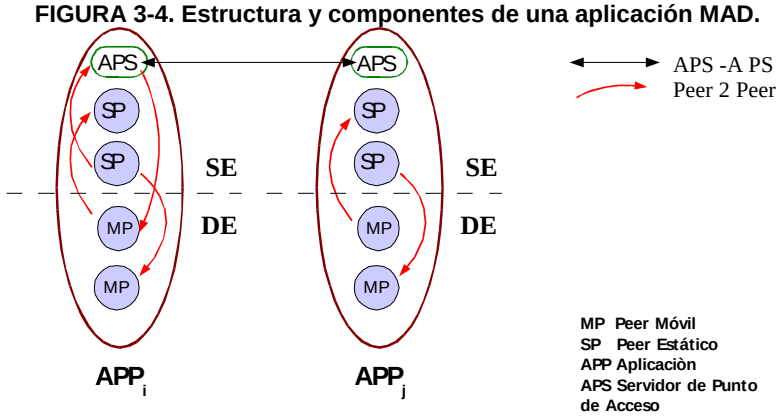
✓ *Modelo Lógico*. Aquellas entidades que representan la arquitectura a nivel de software, es a lo que en MAD se llama el modelo lógico. Éste esta compuesto por otros dos modelos que se explicarán a continuación: el modelo de la aplicación, que describe todas las entidades que hacen parte de una aplicación y el modelo de de servicios, que muestra todas la entidades externas a la aplicación que permiten acceder a recursos, como las bases de datos, archivos, entre otras.

FIGURA 3-3. Modelo físico de la arquitectura MAD.



✓ *Modelo de la Aplicación.* Desde la perspectiva de MAD, una aplicación es un conjunto de procesos (Peers) que trabajan en forma conjunta para conseguir una meta. Los peers de una aplicación pueden encontrarse tanto en el mundo estático como en el mundo dinámico, y su ubicación es determinada cuando se realiza el despliegue de la aplicación [BA2004]. La figura 3-4 ilustra los componentes que hacen parte de una aplicación:

FIGURA 3-4. Estructura y componentes de una aplicación MAD.

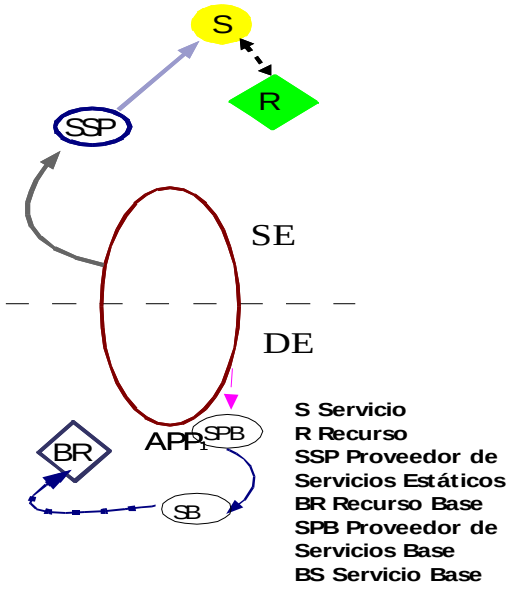


✓ *Modelo de Servicios.* El modelo de servicios define los componentes externos a una aplicación y que le permiten acceder a recursos como Bases de Datos, Servicios Web, entre otros. El acceso a estos recursos garantiza el buen funcionamiento de la aplicación. En la figura 3-5 se ilustra el modelo de servicios [BA2004].

Como es característico en MAD, las entidades se encuentran distribuidas entre el mundo estático y el mundo dinámico. En el mundo estático se halla el recurso R (WebService, Sensor, BD, etc), S que es un servicio que define métodos CRUD para ser aplicados sobre R y el SSP que es un proporcionador de servicios que se encarga de invocar uno o más servicios S para realizar las tareas que garantizan el funcionamiento de la aplicación.

En el mundo dinámico se encuentran el BR, BS, y SPB. El BR es el conjunto de elementos que proporciona un dispositivo móvil para interactuar con el cliente (teclado, pantalla, etc); el BS representa el conjunto de métodos para acceder a los servicios del BR; el SPB procesa las solicitudes hechas por el peer móvil, pero para lograrlo debe acceder a los BR a través del BS.

FIGURA 3-5. Modelo de Servicios MAD.



3.3 ANÁLISIS DE REQUERIMIENTOS DE LA ARQUITECTURA

En esta etapa de recopiló la información necesaria para poder realizar el diseño de Ayllu y se definió la funcionalidad que debe prestar. Adicionalmente se establecieron los requerimientos no funcionales, los cuales han sido esbozados de manera muy general en la propuesta del proyecto (Objetivo general y objetivos específicos) y en durante primer capítulo.

A continuación se explicarán los detalles de la etapa de análisis de requerimientos para la arquitectura Ayllu.

3.3.1 Requerimientos Funcionales. Los requerimientos funcionales de Ayllu se basan en las necesidades de cada una de las capas del modelo conceptual; por esta razón se han dividido de igual forma.

✓ *Recursos & Framework de Agentes.* Como soporte para la capa de recursos se ha decidido emplear el modelo MAD [BA2004], pues éste será el encargado del manejo del repositorio mediante la técnica de "emplear un proporcionador de servicios estático que realiza una actividad de coordinación encargándose de invocar a uno o varios servicios para poder recuperar la información" [BA2004], y manejará las limitaciones de los dispositivos y la comunicación especializada entre los mismos. Por las razones expuestas anteriormente, los requerimientos funcionales de la capa de *Recursos & Framework de Agentes* serán soportados por MAD, los cuales se introducen a continuación:

Framework de Agentes: para el framework de agentes se identificaron dos casos de uso que se pueden observar en la figura 3-6.

FIGURA 3-6. Casos de Uso del Framework de Agentes.

- *Registrar Agentes en los directorios*: El Framework de Agentes de debe proporcionar un mecanismo para registrar los agentes y permitir a su vez el registro de los servicios que estos prestan.

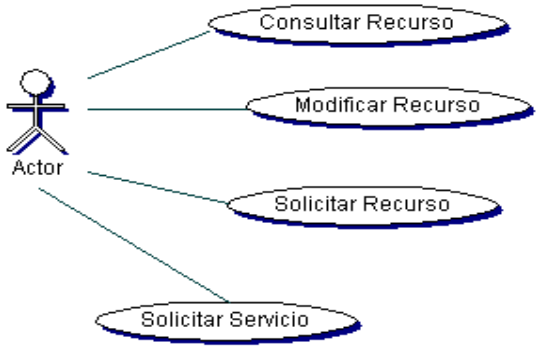
- *Comunicar Agentes*: El Framework debe proporcionar los mecanismos necesarios para permitir que los agentes se comuniquen entre si y puedan ante los eventos en su ambiente.

Recursos: para los recursos se identificaron cuatro casos de uso, que se pueden observar en la figura 3-7.

- *Consultar recurso*: un servicio realiza una consulta a un recurso según la petición realizada por el proporcionador de servicios. Corresponde al Read de las operaciones CRUD.

- *Modificar recurso*: un servicio actualiza, ingresa o modifica información de un recurso según la petición realizada por el proporcionador de servicios. Este requerimiento corresponde al Create, Update y Delete de las operaciones CRUD.

FIGURA 3-7. Casos de uso para Recursos.

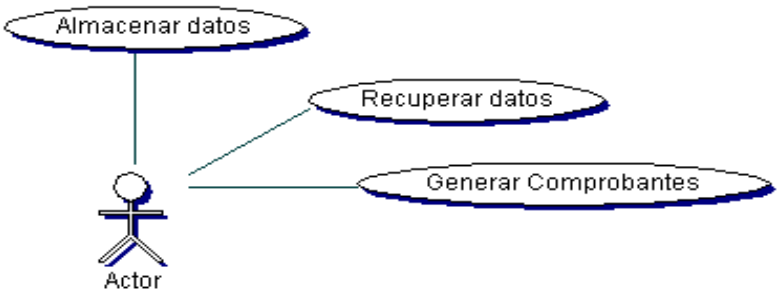


- *Solicitar servicio*: los proporcionadores de servicios solicitan un servicio para llevar a cabo alguna tarea de la aplicación o acceder a un recurso.

- *Solicitar servicio al proporcionador de servicios*: un peer estático o móvil, le solicita a un proporcionador de servicios estático o móvil, un servicio para que la aplicación cumpla su meta.

Repositorio para el repositorio se identificaron tres casos de uso, que se pueden observar en la figura 3-8.

FIGURA 3-8. Casos de Uso para Repositorio.



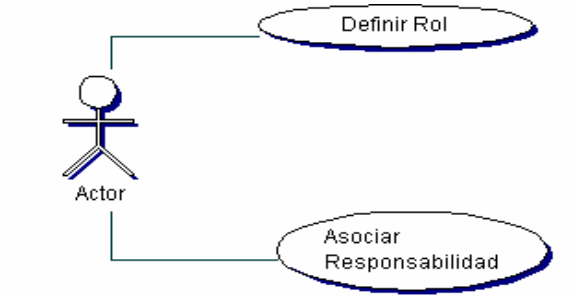
- *Almacenar datos*: MAD soporta el manejo de un repositorio de datos en el mundo estático en el cual los dispositivos móviles del mundo dinámico almacenan artefactos que no necesitan en ese momento.

- *Recuperar datos*: el usuario del dispositivo puede recuperar en cualquier momento la información almacenada en el repositorio de datos para realizar alguna operación sobre los datos.

- *Generar comprobantes*: para identificar que artefacto le pertenece a cada dispositivo se generarán unos comprobantes que contienen el identificador del propietario, del artefacto y el tiempo que debe ser almacenado.

✓ *Capa de Roles*. Se han identificado los casos de uso que se muestran en la figura 3-9 para la capa de roles:

FIGURA 3-9. Casos de Uso para la capa de Roles.

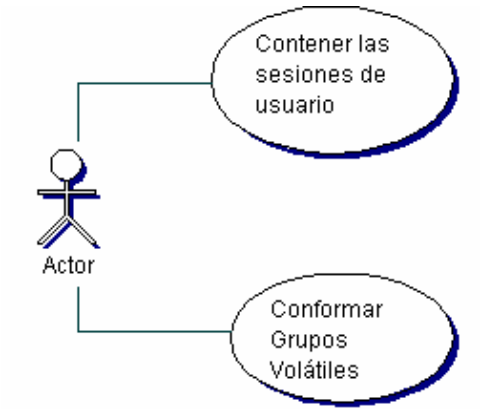


- *Definir Rol*: Ayllu permite definir los roles necesarios para realizar tareas cooperativas. Cada rol es en principio es descrito genéricamente y sabe como manejar un servicio especial.

- *Asociar Responsabilidad*: Cada agente existente dentro de Ayllu debe estar asociado a uno de los roles que se han definido para poder adquirir las responsabilidades y conocimiento sobre como manejar los servicios.

✓ *Capa de Comunidad*. Se han identificado dos casos de uso básicos para la capa de comunidad: Contener las sesiones de Usuario y conformar los grupos volátiles, tal como se presenta en la figura 3-10:

FIGURA 3-10. Casos de Uso para la capa de Comunidad.

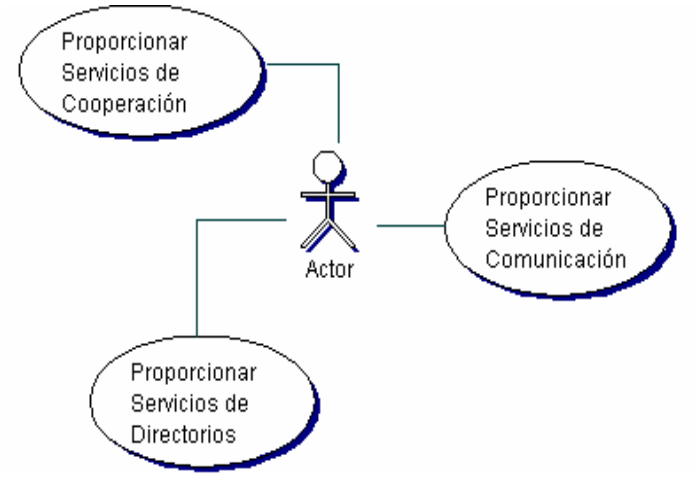


- *Contener las sesiones de usuario*: La capa de comunidad debe mantener todos los *agentes manejadores de sesión SM* y manejar su ciclo de vida dentro de la comunidad. De igual manera debe contener todos los agentes asociados a tareas cooperativas.

- *Conformar Grupos Volátiles*: La base para la ejecución de tareas cooperativas dentro de Ayllu; la capa de comunidad es la encargada de crearlos y manejar su ciclo de vida (un grupo volátil solo existe mientras se realiza una tarea cooperativa).

✓ *Capa de Cooperación.* Como se muestra en la figura 3-11, para la capa de cooperación se han definido tres casos de uso, que son:

FIGURA 3-11. Casos de Uso para la capa de Cooperación.



- *Proporcionar Servicios de Directorios*: Ayllu proporciona el servicio de Directorios de personas para manejar la información personal, las listas de contactos, información sobre recursos y conocimientos de los usuarios.

- *Proporcionar Servicios de Comunicación*: Ayllu proporciona servicios de comunicación para soportar la cooperación entre usuarios.

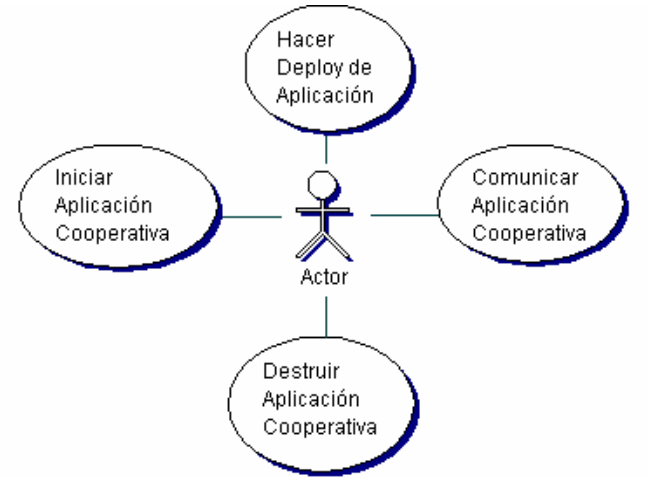
- *Proporcionar Servicios de Cooperación*: Ayllu proporciona de manera explicita la abstracción de la colaboración, coordinación y resolución de conflictos al implementar protocolos de interacción cooperativos entre los usuarios mediados por los agentes que conforman una comunidad de agentes.

✓ *Capa de Aplicación.* Para la capa de aplicación se han definido cuatro casos de uso, basándose en los casos de uso para una aplicación MAD [BA2004], dado que el modelo de comunicaciones empleado en Ayllu está soportado por esta arquitectura. En la figura 3-12 se ilustran estos casos de uso:

- *Iniciar Aplicación Cooperativa*: el usuario inicia la aplicación cooperativa que desee utilizar en su dispositivo. En ese momento se crean los peers que sean necesarios en ambos mundos MAD según lo diseñe el desarrollador de la aplicación.

- *Deploy Aplicación Cooperativa*: para que el usuario pueda utilizar la aplicación cooperativa, se debe realizar previamente el deploy en el dispositivo y en el mundo estático.

FIGURA 3-12. Casos de uso para la capa de Aplicación.



- *Comunicar Aplicación Cooperativa*: las aplicaciones cooperativas se pueden comunicar a través de los servidores de punto de acceso para utilizar las funciones de otra aplicación cooperativa.

- *Destruir Aplicación Cooperativa:* cuando el usuario no desee seguir utilizando la aplicación cooperativa, se destruyen los peers que se habían creado en ambos mundos.

✓ *CAP.* Como se muestra en la figura 3-13, para el CAP se han definido dos casos de uso, que son:

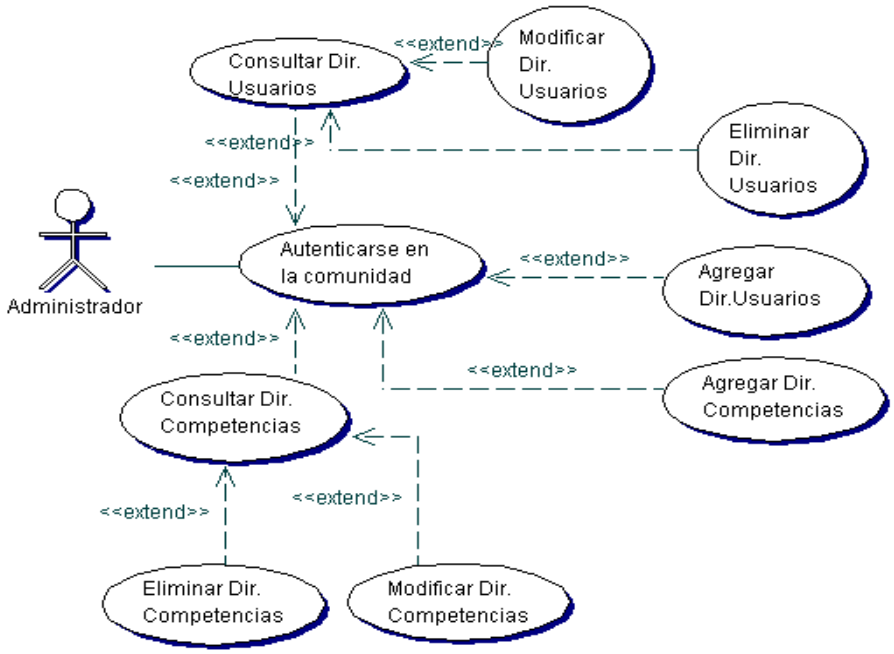
FIGURA 3-13. Casos de uso para el CAP.

- *Permitir Acceso a la Comunidad:* El acceso en Ayllu solo se logra a través del CAP, que permite a un usuario interactuar con los demás miembros del Ayllu de manera cooperativa.

- *Reportar Estado de Conexión:* El CAP reporta periódicamente el estado de conexión de los diferentes IA, permitiendo manejar la intermitencia en la señal y las ausencias temporales del usuario dentro de Ayllu.

✓ *Casos de uso del administrador.* Como muestra la figura 3-14, para el actor administrador se han definido los siguientes casos de uso:

FIGURA 3-14. Casos de uso administrador.



- *Autenticarse en la comunidad:* el administrador proporciona el login y clave que lo identifica como tal y accede al Ayllu.

- *Consultar directorio de usuarios:* el administrador obtiene una vista del directorio de personas, donde se encuentran todos los usuarios del Ayllu.

- *Modificar directorio usuarios:* el administrador puede modificar algún registro del directorio de usuarios.

- *Eliminar directorio usuarios:* el administrador puede eliminar algún registro del directorio de usuarios.

- *Agregar directorio usuarios:* el administrador puede agregar un registro al directorio de usuarios.

- *Consultar directorio competencias:* el administrador obtiene una vista del directorio de competencias, donde se encuentran todas las competencias que proporciona Ayllu.

- *Modificar directorio competencias:* el administrador puede modificar algún registro del directorio de competencias.

- *Eliminar directorio competencias:* el administrador puede eliminar algún registro del directorio de competencias.

- *Agregar directorio competencias:* el administrador puede agregar un registro al directorio de competencias

3.3.2 Requerimientos No Funcionales. La arquitectura Ayllu cumple con los siguientes requerimientos no funcionales:

- Flexible y escalable: permite adicionar nuevos servicios cooperativos de acuerdo a las necesidades de cada aplicación, y a su vez proporciona servicios base que pueden ser reutilizados.
- Robusta: es adaptable a las limitaciones de los dispositivos móviles.
- Segura: La arquitectura incorpora los elementos de seguridad proveídos por MAD para el acceso a los recursos. También contempla los siguientes aspectos de seguridad:
    - Los peers colaborativos solo se pueden comunicar entre sí siempre y cuando pertenezcan al mismo grupo volátil.
    - Un usuario solo puede iniciar una tarea cooperativa siempre y cuando, tenga las competencias requeridas registradas apropiadamente en el servicio de directorio de personas Ayllu.
    - Un usuario no puede acceder al Ayllu si no se encuentra debidamente registrado en el servicio de directorios de personas de Ayllu.
- Transparencia: los detalles de la arquitectura y su manejo son transparentes para el desarrollador.

3.4 DISEÑO E IMPLEMENTACIÓN DE LA ARQUITECTURA AYLLU

Una vez identificados los requerimientos para la arquitectura Ayllu, se inicia con el diseño e implementación. Puesto que Ayllu es una arquitectura basada en agentes de software, en el diseño se explicarán las interacciones, metas y comportamientos de cada uno de los agentes, entre ellos y con los componentes de software que existen dentro de la arquitectura; adicionalmente en la sección de anexos se encontrarán los diagramas de clase que muestran la estructura estática de los elementos desarrollados para la arquitectura.

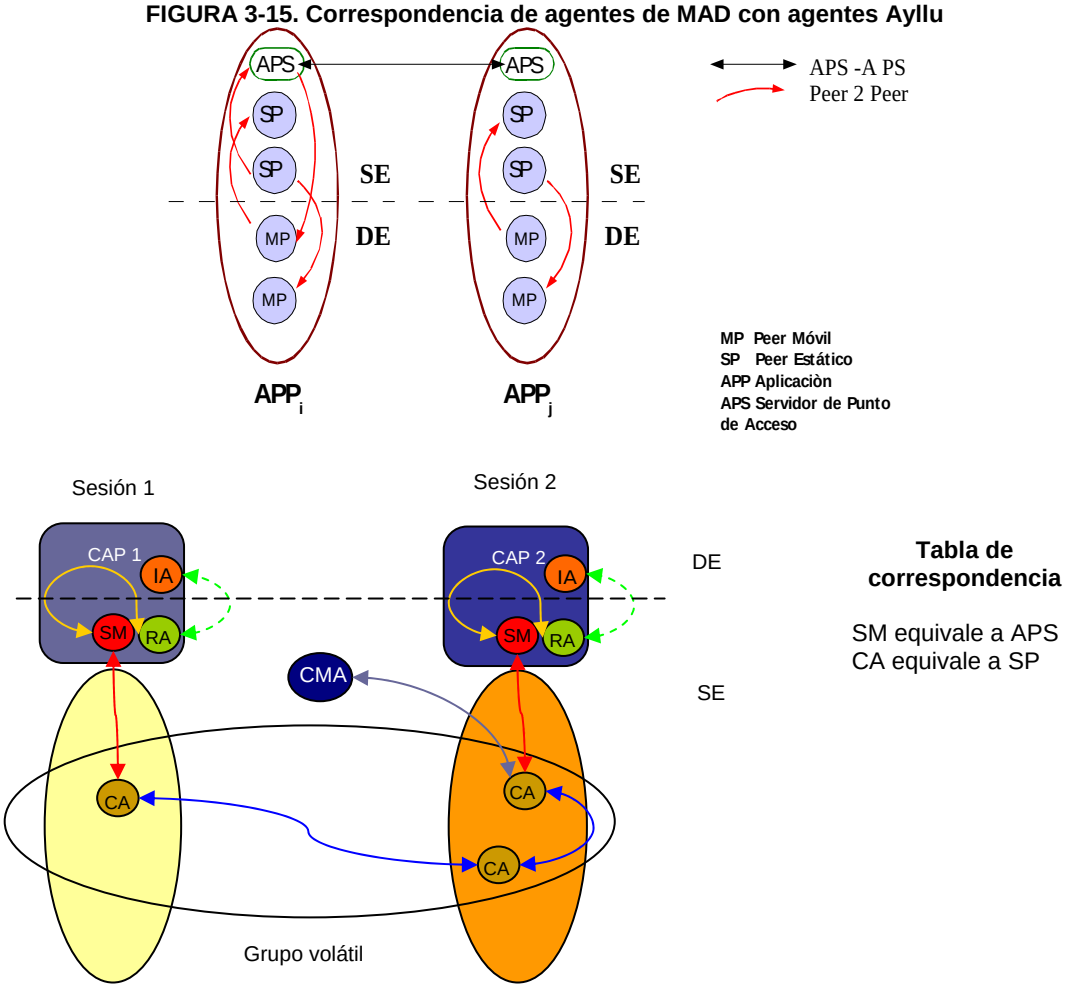
En la implementación se describirán los paquetes utilizados y se mostrará la distribución de la arquitectura Ayllu, que se basa en el modelo MAD [BA2004] para el manejo de la comunicación entre los dispositivos móviles y fijos (por medio del estándar bluetooth), y como soporte para la capa de recursos y framework de agentes junto con BESA y BESACE [GE2003].

3.4.1 Correspondencia entre la Arquitectura MAD y la Arquitectura Ayllu. Ayllu continúa con la misma filosofía que heredó de la arquitectura MAD al establecer una clara distinción entre el mundo estático y el mundo dinámico.

El concepto que maneja MAD de aplicación, corresponde a la sesión que se maneja en Ayllu; sin embargo, se realizó una pequeña variación para poder implementar los grupos volátiles y es permitir a los agentes de comunidad CA de una sesión poder comunicarse

con otros agentes de comunidad CA que se encuentren en otra sesión, es decir que en términos de MAD los peers de una aplicación se pueden comunicar con los peers de otra, sin necesidad de pasar por el APS de cada aplicación; en este caso el agente APS de MAD es el agente manejador de sesión SM en Ayllu. Esto fue posible gracias a que se creo una clave única de comunicación para los agentes de comunidad que participen en el grupo volátil. En la figura 3-15 se puede observar esta correspondencia entre las dos arquitecturas.

FIGURA 3-15. Correspondencia de agentes de MAD con agentes Ayllu



En cuanto a la comunicación, Ayllu se soportó completamente sobre la arquitectura MAD, que manejaba la intermitencia de la conexión (vía bluetooth) y los problemas inherentes entre los dispositivos del mundo dinámico y los dispositivos del mundo estático, permitiendo que se trabajara hacia el objetivo de Ayllu que es fomentar el groupware inteligente entre personas. Si se deseara utilizar otra tecnología de comunicación inalámbrica como por ejemplo WI-FI, los cambios se tendrían que realizar específicamente en la arquitectura MAD, cambiando solamente el código que permite la comunicación entre los mundos estático y dinámico; Ayllu no tendría problemas de integración, por que es independiente en ese sentido con MAD.

✓ *Correspondencia de los agentes de Ayllu con los agentes de MAD, BESA y BESA-CE.* Los agentes de ambos mundos fueron desarrollados utilizando la herencia que proporciona el lenguaje Java. Éstos heredaron tanto de algunos agentes de la arquitectura MAD, por presentar comportamientos similares, como del framework de agentes BESA y BESA-CE. En la tabla 3-1 se puede observar en detalle los agentes de la arquitectura Ayllu:

TABLA 3-1. Implementación de los agentes de la arquitectura Ayllu

Agente de la arquitectura Ayllu	***Plataforma del que heredó***
Agente de comunidad CA	AgentPeer de MAD
Agente manejador de comunidad CMA	AgentBESA de BESA
Agente manejador de sesión SM	AgentAPS de MAD
Agente representante RA	AgentBESA de BESA
Agente interfaz IA	AgentBESA BESA-CE, BESA
Agente administrador AA	AgentBESA de BESA
Agente fábrica FA	AgentBESA de BESA

3.4.2 Diseño de la Arquitectura Ayllu. Basándose en el modelo conceptual, se definieron un conjunto de agentes quienes son los encargados de soportar los diferentes requerimientos de la arquitectura Ayllu, que fueron analizados anteriormente. A continuación se explicarán detalladamente las metas, comportamientos e interacciones de cada uno de ellos. Adicionalmente se presentará el diagrama completo de interacción de todos los agentes y como anexo el diagrama de clases que dará soporte al diseño planteado (ver anexo 2). Algunos aspectos comunitarios para todos los agentes que se describirán a continuación son los siguientes:

- El protocolo de comunicación de las interacciones entre agentes es request-reply.
- Todos los agentes se registran automáticamente en las páginas blancas del directorio de agentes.

✓ *Agente de Comunidad CA*. El *agente de comunidad CA* es el encargado de realizar una tarea cooperativa. Éste es creado por el *Agente Manejador de Comunidad CMA* cuando un usuario ejecuta un servicio de la arquitectura. Éste agente junto con otros de su misma clase, conforman el G*rupo Volátil* para cooperar en una tarea. A continuación se introducen las características de éste agente:

- *Estado*
  - E1. Clave para comunicación con otros agentes de comunidad CA y que pertenezcan al mismo grupo volátil.
  - E2. Clave para comunicación con su agente manejador de sesión SM.
  - E3. Lista de alias de los otros agentes de comunidad CA que pertenezcan al mismo grupo volátil.
  - E4. Alias del agente manejador de sesión SM.
  - E5. Alias del agente manejador de comunidad CMA.
- *Metas*

M1. Manejar las interacciones con otros agentes de comunidad CA en el grupo volátil.
M2. Manejar las interacciones con su agente manejador de sesión SM.
M3. Iniciar a trabajar por orden del agente manejador de comunidad CMA.

- *Entradas Sensoriales*
  - *Sincrónicas*
    S1. Encontrar a otro agente de comunidad CA de su mismo grupo volátil.
    S2. Encontrar a su agente manejador de sesión SM.
    S3. Encontrar a su agente manejador de comunidad CMA.
  - *Asincrónicas*
    - No tiene.
- *Actuadores*
  A1. Registrarse en las páginas amarillas del directorio de agentes.
- *Comportamientos*
  B1. Comportamiento del agente de comunidad CA:
    - Recibir un request de otro agente de comunidad CA: cumple con la meta M1.
      Mensaje/Evento: COOPERATION_REQUEST, request de un agente de comunidad CA a otro agente de comunidad CA.
      Acción: depende del servicio a implementar, se proporciona un método llamado *handlerCooperationRequestAction,* que debe ser implementado por el programador, recibe un dato DataMessage que contiene el tipo de mensaje, el emisor, el receptor y el mensaje. Por ejemplo si el programador desea enviar un evento de reply puede llamar al método *replyCooperation* proporcionado por el agente de comunidad CA, desde el método a implementar que enviará un evento BESA COOPERATION_REPLY.

    - Recibir un reply de otro agente de comunidad CA: cumple con la meta M1.
      Mensaje/Evento: COOPERATION_REPLY, reply de un agente de comunidad CA a otro agente de comunidad CA.
      Acción: depende del servicio a implementar, se proporciona un método llamado *handlerCooperationReplyAction* que debe ser implementado por el programador, recibe un dato DataMessage que contiene el tipo de mensaje, el emisor, el receptor y el mensaje. Por ejemplo si el programador desea enviar un evento de request puede llamar al método *requestCooperation* proporcionado por el agente de comunidad CA, desde el método a implementar que enviará un evento BESA COOPERATION_REQUEST.

    - Recibir una respuesta de su agente manejador de sesión SM: cumple con la meta M2.
      Mensaje/Evento: QUESTION_REPLY, reply de un agente manejador de sesión SM a un agente de comunidad CA.
      Acción: depende del servicio a implementar, se proporciona un método llamdado *handlerSMQuestionReplyAction*, que debe ser implementado por el programador, recibe un dato DataRAQuestionReply, que contiene la respuesta a la pregunta realizada al agente manejador de sesión SM, junto con otros datos que identifican el emisor y receptor, entre otros. Por ejemplo si el programador desea enviar otra pregunta a su agente

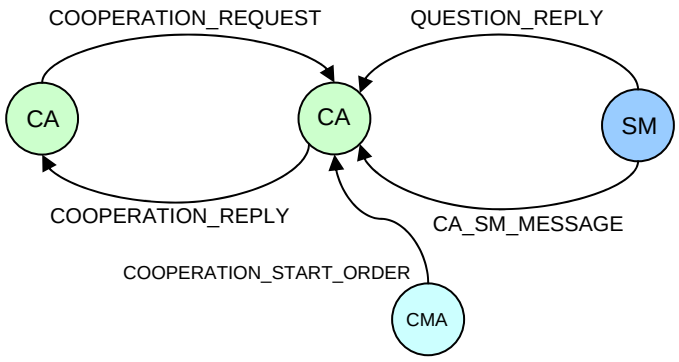
manejador de sesión SM puede llamar al método *askQuestionSM* proporcionado por el agente de comunidad CA, desde el método a implementar que enviará un evento BESA QUESTION_REQUEST.

- Recibir un mensaje de su agente manejador de sesión SM: cumple con la meta M2.
  Mensaje/Evento: CA_SM_MESSAGE, mensaje del agente manejador de sesión SM a un agente de comunidad CA.
  Acción: depende del servicio a implementar, se proporciona un método llamado *handlerSMMessage* que debe ser implementado por el programador, recibe un dato DataMessage que contiene el tipo de mensaje, el emisor, el receptor y el mensaje. Por ejemplo si el programador desea enviar un mensaje a su agente manejador de sesión SM puede llamar al método *sendMessageSM* proporcionado por el agente de comunidad CA, desde el método a implementar que enviará un evento BESA CA_SM_MESSAGE.

- Empezar a trabajar: cumple con la meta M3.
  Mensaje/Evento: COOPERATION_START_ORDER, orden para iniciar a trabajar.
  Acción: depende del servicio a implementar, se proporciona un método *handlerStartWorkingOrder* que debe ser implementado por el programador, recibe un dato DataMessage que contiene el tipo de mensaje, el emisor, el receptor y el mensaje.

En la figura 3-16, se puede observar todas las interacciones del agente de comunidad CA descritas anteriormente.

FIGURA 3-16. Interacción del agente de comunidad CA con otros agentes de Ayllu.



Nota: como el agente de comunidad CA hereda del AgentPeer de MAD, por esta razón también debe cumplir con las características de este agente descritas en la arquitectura MAD. Para mayor detalle remitirse a MAD [BA2004].

✓ *Agente Manejador de Comunidad CMA.* El *agente manejador de comunidad CMA* es el encargado de buscar a posibles candidatos para realizar una tarea cooperativa en el directorio de personas y en consecuencia crea, destruye y coordina los *agentes de*

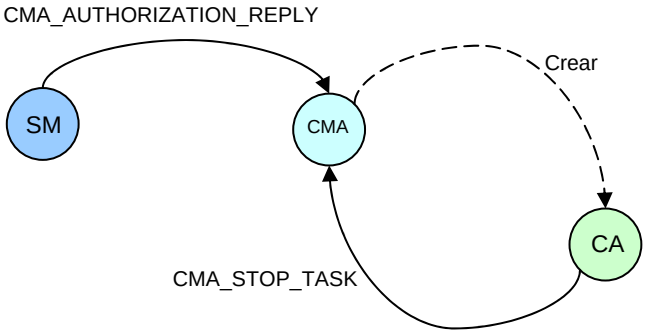
comunidad CA para un servicio en particular. A continuación se introducen las características de éste agente:

- *Estado*
  - E1. Clave que genera para la comunicación de los agentes de comunidad.
  - E2. Lista de los agentes de comunidad que éste crea.
  - E3. Lista de los agentes manejadores de sesión a los que pedirá autorización para que participen en una tarea cooperativa.
  - E4. Lista de usuarios no aceptaron cooperar en la tarea.
  - E5. Referencia al directorio de personas.
- *Metas*
  - M1. Crear agentes de comunidad CA.
  - M2. Destruir agentes de comunidad CA.
- *Entradas Sensoriales*
  - o *Sincrónicas*
    - S1. Encontrar a un agente manejador de sesión SM.
    - S2. Encontrar a un agente de comunidad CA.
  - o *Asincrónicas*
    - No tiene
- *Actuadores*
  - A1. Registrarse en las páginas amarillas del directorio de agentes.
  - A2. Buscar los usuarios candidatos para realizar una tarea cooperativa el directorio de personas.
- *Comportamientos*
  - B1. Comportamiento de un agente manejador de comunidad CMA:
    - Recibir la respuesta de autorización para la creación de agentes de comunidad CA: cumple con la meta M1.
      Mensaje/Evento: CMA_AUTHORIZATION_REPLY, reply del agente manejador de sesión SM al agente de manejador de comunidad CMA.
      Acción: depende del servicio a implementar, se proporciona el método *createCA*, que el programador debe implementar; éste método es llamado por el método *handlerAuthorizationReplyAction* proporcionado por el agente manejador de comunidad CMA, que recibe el mensaje de autorización de tipo *DataCMAAuthorizationReply* el cual contiene un tipo dato que indica si se debe crear para el agente manejador de sesión SM un agente de comunidad CA o no, para un servicio.

    - Recibir la orden para destruir un agente de comunidad CA: cumple con la meta M2.
      Mensaje/Evento: CMA_STOP_TASK, mensaje del agente de comunidad CA para notificar que debe ser destruido por el agente manejador de comunidad CMA.
      Acción: el agente manejador de comunidad proporciona dos métodos que deben ser implementados por el programador y son *destroyCA y destroyCA con argumentos.* Dependiendo del servicio el programador implementa uno o ambos. Estos métodos deben destruir a todos los agentes de comunidad o a uno específico.

En la figura 3-17, se puede observar todas las interacciones del agente manejador de comunidad CMA descritas anteriormente.

FIGURA 3-17. Interacciones del agente manejador de comunidad CMA con otros agentes de Ayllu.



✓ *Agente Manejador de Sesión SM*. El *agente manejador de sesión SM* es el encargado de verificar el estado del *Punto de Acceso a la Comunidad CAP*, otra de sus funciones es ser el punto de entrada y salida para la comunicación de los agentes de la *sesión* y el usuario que interactúa por medio del CAP. A continuación se introducen las características de éste agente:

- *Estado*
  E1. Tabla de los agentes interfaz IA.
  E2. Clave de comunicación con los agentes de comunidad CA de su sesión.
  E3. Alias de su agente representante RA.
- *Metas*
  M1. Manejar las interacciones con el agente manejador de comunidad CMA.
  M2. Manejar las interacciones con los agentes de comunidad CA.
  M3. Manejar las interacciones con los agentes interfaz IA.
  M4. Manejar las interacciones con el agente representante RA.
- *Entradas Sensoriales*
  - *Sincrónicas*
    S1. Encontrar a un agente manejador de comunidad CMA.
    S2. Encontrar a un agente de comunidad CA.
    S3. Encontrar a un agente interfaz IA.
    S4. Encontrar a un agente representante RA.
  - *Asincrónicas*
    S5. Monitorear periódicamente cual es el agente interfaz IA que esta activo, para enviarle peticiones o mensajes.
- *Actuadores*
  A1. Registrarse en las páginas amarillas del directorio de agentes.
- *Comportamientos*
  B1. Comportamiento del agente manejador de sesión SM:
    - Recibir la petición de autorización de un agente manejador de comunidad CMA para participar en un servicio cooperativo: cumple con la meta M1.
      Mensaje/Evento: CMA_AUTHORIZATION_REQUEST, request de participación en un servicio cooperativo, es enviado por el agente manejador de comunidad CMA.

Acción: el agente manejador de sesión SM llama al método *handlerAuthorizationRequestAction* que recibe un tipo de dato *DataCMAAuthorizationRequest* con los datos de la petición y el emisor de la pregunta, el método anterior se encarga de llamar al método *askQuestion* que recibe el mismo tipo de dato, quien envía un evento BESA QUESTION_REQUEST al agente representante RA. Los anteriores métodos son proporcionados por el agente manejador de sesión SM.

- Recibir una petición de un agente de comunidad CA: cumple con la meta M2.
  Mensaje/Evento: QUESTION_REQUEST, request enviado por un agente de comunidad CA al agente manejador de sesión SM.
  Acción: el agente manejador de sesión SM llama al método *handlerCAQuestionReq* que recibe un tipo de dato *DataRAQuestionReq* con los datos de la petición y el emisor de la pregunta, el método anterior se encarga de llamar al método *askQuestion*, que recibe el mismo tipo de dato, quien envía un evento BESA QUESTION_REQUEST al agente representante RA. Los anteriores métodos son proporcionados por el agente manejador de sesión SM.

- Recibir un mensaje de un agente de comunidad CA: cumple con la meta M2.
  Mensaje/Evento: CA_SM_MESSAGE, mensaje que envía el agente de comunidad CA al agente manejador de sesión SM.
  Acción: el agente manejador de sesión SM llama al método *handlerCAMessage*, que recibe un tipo de dato *DataMessage* con los datos del mensaje típicamente el tipo de mensaje, el emisor, el receptor y el mensaje. El método anterior llama al método *sendMessageToIA*, que recibe el mismo tipo de dato y que se encarga de enviar un evento BESA IA_SM_MESSAGE al agente interfaz IA. Los anteriores métodos son proporcionados por el agente manejador de sesión SM.

- Recibir un mensaje de un agente interfaz IA: cumple con la meta M3.
  Mensaje/Evento: IA_SM_MESSAGE, mensaje que envía el agente interfaz IA activo al agente manejador de sesión SM.
  Acción: el agente manejador de sesión SM llama al método *handlerIAMessage*, que recibe un tipo de dato *DataMessage* con los datos del mensaje típicamente el tipo de mensaje, el emisor, el receptor y el mensaje. El método anterior llama al método *sendMessageToCA*, que recibe el mismo tipo de dato y que se encarga de enviar un evento BESA CA_SM_MESSAGE al agente de comunidad CA especificado. Los anteriores métodos son proporcionados por el agente manejador de sesión SM.

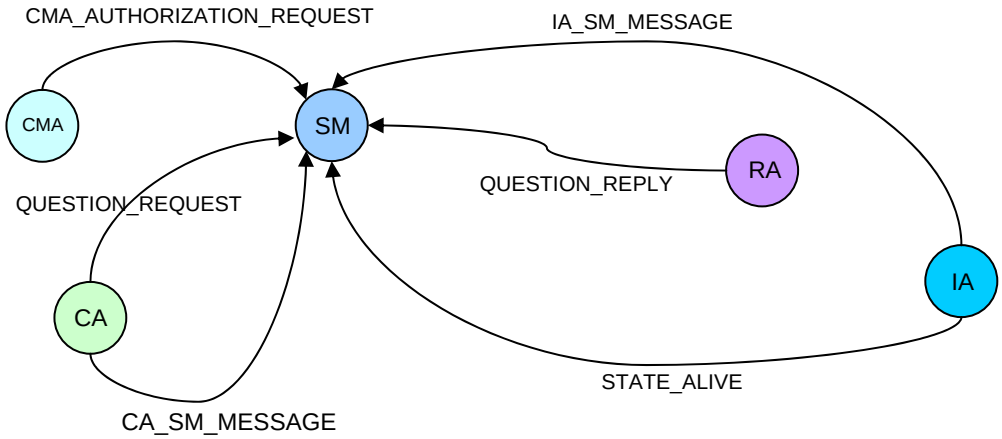
- Recibir un reporte del estado de un agente interfaz IA: cumple con la meta M3.
  Mensaje/Evento: STATE_ALIVE, reporte que envían los agentes de interfaz IA vivos al agente manejador de sesión SM.

Acción: el agente manejador de sesión SM llama al método *handlerIAReportState*, que recibe el tipo de dato *DataIAReportState* el cual contiene la información del tiempo de reporte del agente Interfaz IA que envía la notificación. El método anterior se encarga de actualizar la tabla de IA que tiene el agente manejador de sesión SM.

- Recibir una respuesta de su agente representante RA: cumple con la meta M4.
  Mensaje/Evento: QUESTION_REPLY, reply del agente representante RA al agente manejador de sesión SM.
  Acción: el agente manejador de sesión SM llama al método *handlerQuestionReplyAction*, que recibe un tipo de dato *DataRAQuestionReply* con la respuesta a un request que se haya realizado. El método anterior identifica a quien va dirigido y envía un evento BESA CMA_AUTHORIZATION_REPLY en el caso que la respuesta sea de respuesta a un agente manejador de comunidad CMA, o envía un evento BESA QUESTION_REPLY cuando la respuesta es enviada a un agente de comunidad CA. También es posible que actualice la tabla de IA si se encontró que en los datos del mensaje se especificó que no hubo una respuesta por parte del agente interfaz IA al cual iba dirigida una petición.

En la figura 3-18, se puede observar todas las interacciones del agente manejador de sesión SM descritas anteriormente.

FIGURA 3-18. Interacciones del agente manejador de sesión SM con otros agentes de Ayllu.



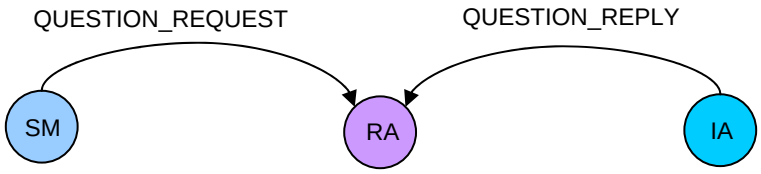
Nota: como el agente de sesión SM hereda del AgentAPS de MAD, por esta razón también debe cumplir con las características de este agente descritas en la arquitectura MAD. Para mayor detalle remitirse a MAD [BA2004].

✓ *Agente Representante RA.* El *agente representante RA* es el encargado de filtrar peticiones para evitar tener que preguntar al usuario cada vez que se requiera su participación en una tarea cooperativa. Este agente recuerda las respuestas a las peticiones; sin embargo podría utilizar otros mecanismos para ser más inteligente. A continuación se introducen las características de éste agente:

- *Estado*
  E1. Tabla donde almacena las respuestas a preguntas, con el fin de no tener que preguntar al usuario siempre por las mismas cosas.
- *Metas*
  M1. Manejar las interacciones con el agente manejador de sesión SM.
  M2. Recordar las respuestas a las peticiones que le realicen.
  M3. Manejar las interacciones con los agentes interfaz IA.
- *Entradas Sensoriales*
  - *Sincrónicas*
    S1. Encontrar a su agente manejador de sesión SM.
    S2. Encontrar a un agente interfaz IA.
  - *Asincrónicas*
    - No tiene
- *Actuadores*
  A1. Registrarse en las páginas amarillas del directorio de agentes.
- *Comportamientos*
  B1. Comportamiento del agente representante RA:
    - Recibir una petición del agente manejador de sesión SM: cumple con la meta M1 y M2.
      Mensaje/Evento: QUESTION_REQUEST, request de una petición, es enviada por el agente manejador de sesión SM al agente representante RA.
      Acción: el agente representante RA por medio del método *handlerRAQuestionRequestAction* que recibe un dato *DataRAQuestionReq* con la información de la petición, revisa la tabla de respuestas. Si conoce la respuesta a la petición que le están haciendo, se envía un evento BESA QUESTION_REPLY al agente manejador de sesión SM que envió la petición. En caso contrario se llama al método *askQuestion*, que recibe el mismo tipo de dato y se envía un evento BESA QUESTION_REQUEST al agente interfaz especificado en el dato *DataRAQuestionReq.*
    - Recibir una respuesta de un agente interfaz IA: cumple con la meta M2 y M3.
      Mensaje/Evento: QUESTION_REPLY, reply con la respuesta a una petición, es enviada por el agente interfaz IA al agente representante RA.
      Acción: el agente representante RA por medio del método *handlerRAQuestionReplyAction* que recibe un dato *DataRAQuestionReply* con la respuesta a una petición, guarda la respuesta a dicha petición y la reenvía al agente manejador de sesión SM por medio de un evento BESA QUESTION_REPLY.

En la figura 3-19, se puede observar todas las interacciones del agente representante RA descritas anteriormente.

FIGURA 3-19. Interacciones del agente representante RA con otros agentes de Ayllu.



✓ *Agente Interfaz IA.* El *agente interfaz IA* es el encargado de mostrar al usuario según el dispositivo en el que éste se encuentre trabajando los mensajes que provienen de sus agentes siempre y cuando sea necesario, también captura las peticiones que realiza el usuario y las delega al *agente fábrica FA* con el objetivo de dar inicio a una tarea cooperativa o al *agente administrador AA* para funciones administrativas de la arquitectura Ayllu. A continuación se introducen las características de éste agente:

- *Estado*
  - E1. Alias de su agente manejador de sesión SM.
  - E2. Referencia a la aplicación a la cual debe responderle.
  - E3. Referencia a la interfaz de administración.
  - E4. Thread para reportarse a su agente manejador de sesión SM (explicado posteriormente).
- *Metas*
  - M1. Manejar las interacciones con el agente representante RA.
  - M2. Manejar las interacciones con las interfaces administrativas.
  - M3. manejar las interacciones con el agente manejador de sesión SM.
  - M4. Mostrar mensajes al usuario
- *Entradas Sensoriales*
  - *Sincrónicas*
    - S1. Encontrar al agente representante RA.
    - S2. Encontrar al agente manejador de sesión SM
  - *Asincrónicas*
    - No tiene.
- *Actuadores*
  - A1. Registrarse en las páginas amarillas del directorio de agentes.
- *Comportamientos*
  - B1. Comportamiento del agente interfaz IA.
    - Recibir una petición del agente representante RA: cumple con la meta M1 y M4.

      Mensaje/Evento: QUESTION_REQUEST, request que envía el agente representante RA para ser atendido por el usuario, mediado por el agente interfaz.

      Acción: depende del servicio. El agente interfaz IA proporciona el método *handlerRAQuestionRequest* que debe ser implementado por el programador el cual recibe un dato *DataRAQuestionReq* que especifica la petición. En este método se debe mostrar al usuario cual es la petición que se le ha enviado. Generalmente se debe quedar esperando de una vez la respuesta a la petición y enviarla mediante un evento BESA QUESTION_REPLY al agente representante RA.

    - Recibir una respuesta del componente administrativo: cumple con la meta M2 y M4.

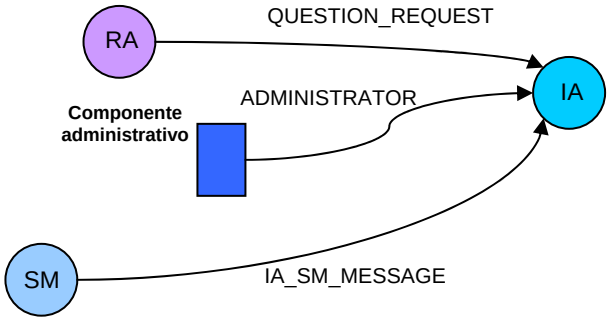
      Mensaje/Evento: ADMINISTRATOR, respuesta de una acción administrativa es enviada directamente por el componente administrativo que realiza dicha acción al agente interfaz IA.

Acción: depende del servicio el agente internar IA responde y muestra al usuario alguna información, todos estos sucesos deben ser implementados por el programador.

- Recibir un mensaje del agente manejador de sesión SM: cumple con la meta M3 y M4.
  Mensaje/Evento: IA_SM_MESSAGE, mensaje enviado por agente manejador de sesión SM al agente interfaz IA.
  Acción: depende del servicio, el agente interfaz llama al método *handlerSMMessage* que recibe un tipo de dato *DataMessage* que contiene el mensaje a mostrar al usuario por intermedio del agente interfaz IA. El programador debe implementar éste método.

En la figura 3-20, se puede observar todas las interacciones del agente interfaz IA descritas anteriormente.

FIGURA 3-20. Interacciones del agente interfaz IA con otros agentes de Ayllu.



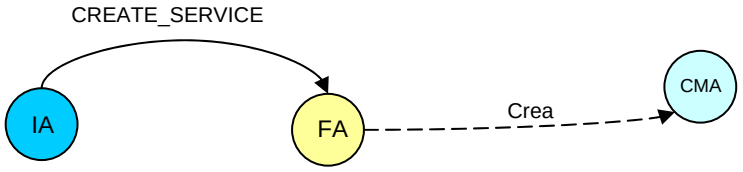
✓ *Agente Fábrica FA.* El *agente fábrica FA* es el encargado de crear por demanda a los *agentes manejadores de comunidad CMA*, para cada uno de los servicios de la arquitectura Ayllu. A continuación se introducen las características de éste agente:

- *Estado*
  E1. Referencia al directorio de competencias
- *Metas*
  M1. Crear a los agentes manejadores de comunidad CMA de un servicio en particular.
- *Entradas Sensoriales*
  - *Sincrónicas*
    - No tiene.
  - *Asincrónicas*
    - No tiene.
- *Actuadores*
  A1. Registrarse en las páginas amarillas del directorio de agentes.
- *Comportamientos*
  B1. Comportamiento del agente fabrica FA:
    - Recibir la orden para crear agentes manejadores de comunidad CMA: cumple con la meta M1.

Mensaje/Evento: CREATE_SERVICE, orden para crear a los agentes manejadores de comunidad CMA, es enviada por el agente interfaz IA.
Acción: dependiendo de los diferentes servicios a crear el programador debe implementar el método *createCMAAgents* que es llamado por el método *handlerCreateService* proporcionado por el agente. En este método se valida si el usuario tiene los permisos suficientes para poder ejecutar el servicio cooperativo y de ser así se llama al método *createCMAAgents*. Se recibe un dato *DataCMAInvocation* que dice que servicio es el que se va a invocar.

En la figura 3-21, se puede observar la interacción del agente fabrica FA descrita anteriormente.

FIGURA 3-21. Interacciones del agente fábrica FA con otros agentes de Ayllu.

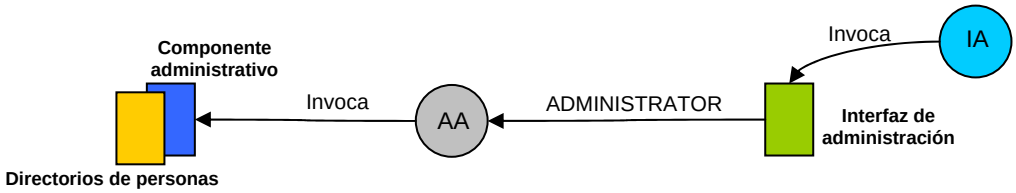


✓ *Agente Administrador AA*. El *agente administrador AA*, es el encargado de realizar tareas administrativas exclusivamente sobre el directorio de personas a través de un componente que realiza el trabajo pesado. A continuación se introducen las características de éste agente:

- *Estado*
  E1. Referencia al componente administrativo.
- *Metas*
  M1. Realizar una acción administrativa como por ejemplo permitir el login, CRUD de usuarios y CRUD de lista de contactos usado en servicios particulares.
- *Entradas Sensoriales*
  - *Sincrónicas*
    - No tiene.
  - *Asincrónicas*
    - No tiene.
- *Actuadores*
  A1. Registrarse en las páginas amarillas del directorio de agentes.
- *Comportamientos*
  B1. Comportamiento del agente administrador AA:
  - Realizar una operación administrativa: cumple con la meta M1.
    Mensaje/Evento: ADMINISTRATOR, orden enviada por la interfaz administrativa del mundo dinámico, para el agente administrador AA, con el fin de realizar alguna acción sobre el directorio de personas.
    Acción: dependiendo de la orden enviada, el agente administrador llama a alguno de sus métodos, que a su vez reenvían la solicitud al componente administrativo, quien se encarga de realizar el trabajo en los directorios de personas y retornar la respuesta.

En la figura 3-22, se puede observar la interacción del agente administrador AA descrita anteriormente.

FIGURA 3-22. Interacción del agente administrador AA con otros agentes y componentes de Ayllu.



3.4.3 Diagrama de Interacción de los Agentes en la Arquitectura Ayllu. En la figura 3-23 se muestran las interacciones de todos los agentes en la arquitectura Ayllu con los componentes de software y el usuario final. A continuación se describirá el diagrama de interacción:

- Descripción de un proceso de administración por medio de la arquitectura Ayllu:

1. Los agentes interfaz IA envían una petición administrativa a la interfaz de administración, mediante la invocación de un método.
   1.1. La interfaz administrativa envía un evento BESA para que reaccione el agente administrador AA.
      1.1.1. El agente administrador AA dependiendo de la petición, invoca el método asociado sobre el componente administrativo.
   1.2. El componenete administrativo envía la respuesta a la petición por medio de un evento BESA al agente interfaz IA.

- Descripción del proceso de invocación de servicios y creación de grupos volátiles:

2. Un usuario desea realizar una tarea cooperativa, invoca un servicio que el agente interfaz IA atiende y éste envía un evento BESA para que reaccione el agente fábrica FA.
   2.1. El agente fábrica FA, recibe la petición de invocación y crea al agente manejador de comunidad CMA del servicio.
3. El agente manejador de comunidad CMA inicia la búsqueda de los posibles candidatos invocando un método que busca a los usuarios que pueden cooperar para la tarea en los directorios de personas y recibe una respuesta a su petición.
4. El agente manejador de comunidad CMA, con los posibles candidatos identificados, realiza a cada uno de los agentes manejadores de sesión SM de cada usuario, una petición de autorización, en donde se le pide participar en la tarea cooperativa. Ésta acción la realiza mediante un evento BESA para que reaccione cada agente manejador de sesión SM.
   4.1. El agente manejador de sesión SM de cada usuario envía una petición, para preguntar si se desea participar en la tarea cooperativa, al agente representante RA, mediante un evento BESA para que reaccione éste agente.
      4.1.1. El agente representante RA si conoce la respuesta, la devuelve al agente manejador de sesión SM, mediante un evento BESA para

que reaccione éste agente y con esto se evita hacer preguntas innecesarias al usuario.

4.2. El agente representante RA si no conoce la respuesta, envía un evento BESA al cual reacciona el agente interfaz IA, quien dará a conocer al usuario la petición, para que éste decida que hacer con ella.

4.2.1. El agente interfaz IA envía la respuesta que haya dado el usuario al agente representante RA, mediante un evento BESA para que reaccione éste agente.

4.3. El agente representante RA guarda la respuesta y la envía al agente manejador de sesión SM, mediante un evento BESA para que reaccione éste agente.

5. El agente manejador de sesión SM, retorna la respuesta obtenida al agente manejador de comunidad CMA que le pidió la autorización, enviando un evento BESA para que reaccione éste agente.
6. Si el agente manejador de comunidad CMA recibe una respuesta afirmativa a la petición de autorización del agente manejador de sesión SM, éste procede a crear a los agentes de comunidad CA, para cada uno de los usuarios que aceptaron la petición.
7. Una vez creados los agentes de comunidad CA, el agente manejador de comunidad CMA, envía un evento BESA por el cual reaccionan éstos agentes y se colocan en disposición de trabajar. Ésta orden depende del servicio, pues el agente manejador de comunidad puede mandar a trabajar a uno solo o a varios agentes de comunidad.
8. Cuando los agentes de comunidad empiezan a cooperar para resolver una tarea, esto es conocido como grupo volátil. El envío y recepción de mensajes entre agentes de comunidad es mediante eventos BESA que los hacen reaccionar.
9. Cuando los agentes de comunidad terminan su tarea envían un evento BESA al agente manejador de comunidad, para que éste los destruya.

9.1. El agente manejador de comunicad CMA destruye a los agentes.

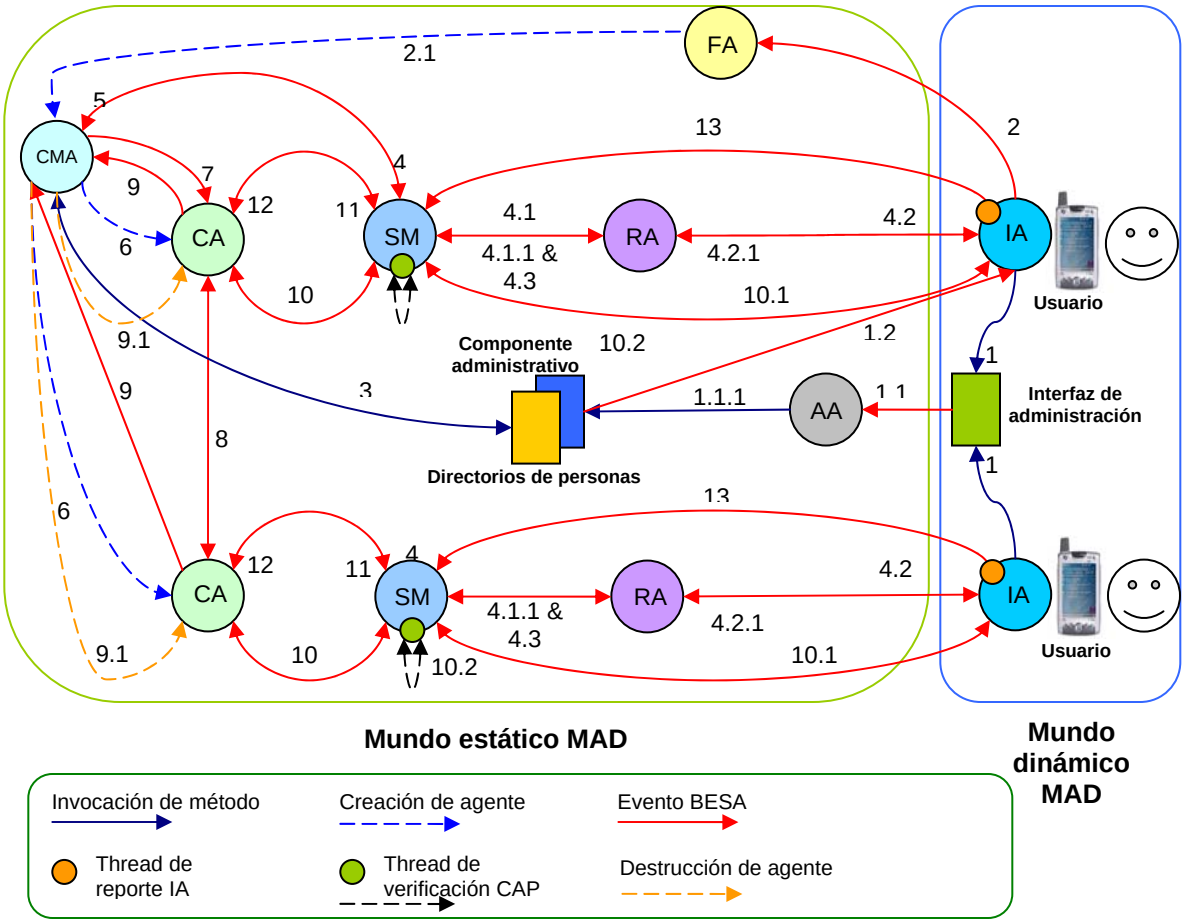
10. Dependiendo de servicio los agentes de comunidad CA pueden enviar mensajes y/o peticiones para realizar alguna acción. Un agente de comunidad CA envía un mensaje mediante un evento BESA al que reacciona el agente manejador de sesión SM.

10.1. El agente manejador de sesión SM envía el mensaje directamente al agente interfaz IA, mediante un evento BESA al que reacciona este agente, para que sea mostrado al usuario en el dispositivo.

10.2. El usuario también puede enviarle un mensaje a sus agentes mediante el agente interfaz IA, quien recibe el mensaje y envía un evento BESA para que reaccione el agente manejador de sesión SM.

11. Si el agente de comunidad debe enviar una petición para preguntar sobre alguna información que se necesita para continuar la tarea cooperativa, éste envía un evento BESA para que reaccione el agente manejador de sesión SM.
12. El agente manejador de sesión SM realiza los pasos descritos en los numerales 4.1 a 4.3 para enviar la petición al usuario, con la variación que éste agente responderá a un agente de comunidad CA mediante un evento BESA, para que reaccione.
13. Por último el agente interfaz IA debe estar continuamente reportando su estado y notificar que se encuentra vivo. Para esto envía un evento BESA periódicamente cada cierto tiempo para notificar su estado al agente manejador de sesión SM.

FIGURA 3-23. Diagrama de interacción de los agentes y componentes en la arquitectura Ayllu.



3.4.4 Implementación de la arquitectura Ayllu. La implementación de la arquitectura Ayllu se realizó sobre la arquitectura MAD [BA2004], que proporcionó el manejo de la capa de recursos para realizar las operaciones CRUD y manejó los problemas de los dispositivos móviles explicados en capítulos anteriores.

Adicionalmente el modelo de agentes con el que se implementaron los agentes de la arquitectura Ayllu, heredaron en gran parte de la arquitectura MAD y del framework para agentes BESA y BESACE [GE2003]. Todas las plataformas fueron desarrolladas sobre el lenguaje Java.

En el mundo estático se decidió implementar los siguientes agentes y componentes:

- Agente de comunidad CA.
- Agente manejador de comunidad CMA.
- Agente manejador de sesión SM.
- Agente representante RA.
- Agente administrador AA.
- Agente fábrica FA.

- Directorio de personas.
- Sensor Ayllu

Las razones para dejar este conjunto de agentes y componentes en el mundo estático fueron las siguientes:

- Mayor capacidad y procesamiento en los dispositivos del mundo estático, permite que las tareas cooperativas tengan más probabilidades de ser completadas.
- En el mundo estático los problemas de computación distribuida pueden ser corregidos más rápidamente y más confiablemente que en el mundo dinámico.
- Existe mayor confiabilidad en cuanto a comunicación se refiere, pues en el mundo estático la conexión no tiende a ser tan intermitente ni tiene limitaciones drásticas de alcance como si las tienen en el mundo dinámico.
- Los tiempos de respuesta para realizar tareas cooperativas son indudablemente menores, debido a que los agentes residen en equipos donde hay más recursos de máquina y puede haber escalabilidad horizontal y/o vertical.
- La seguridad pude ser manejada mucho más eficientemente que en el mundo dinámico. Puesto que en el futuro se pueden implementar mejores algoritmos para solucionar estos problemas, que podrían necesitar mayor poder computacional que no poseen los dispositivos móviles actuales.

En el mundo dinámico se implementó el agente interfaz IA. Las razones para dejar este agente y este componente en el mundo dinámico fueron las siguientes:

- Cada aplicación esta ligada a un agente interfaz que corre sobre un dispositivo móvil, aunque podría extenderse para poder correr una aplicación en un dispositivo estático también. Con esto se busca que haya movilidad para el usuario que la utilice y disponibilidad en cualquier lugar.
- Dado que existe la limitante de espacio y procesamiento en los dispositivos móviles se decidió dejar a un solo agente ligero que actuara como un service provider base [BA2004], permitiendo acceso a los recursos físicos del dispositivo.

✓ *Estructura de la implementación de la arquitectura Ayllu.* Para la implementación se decidió dejar de manera separada los agentes y componentes que se despliegan en el mundo estático del mundo dinámico.

La implementación se realizó con un enfoque orientado a objetos, utilizando paquetes para mayor organización y para mostrar las capas que se expusieron en el modelo conceptual. La siguiente es la organización de la estructura de paquetes manejada para ambos mundos. En la figura 3-24, se muestran los paquetes para el mundo estático y en la figura 3-25, se muestran los paquetes para el mundo dinámico. Como se observa en las figuras, existen paquetes que poseen el mismo nombre en ambos mundos; esto se hizo con el fin de mantener una estructura similar en ambos mundos, dado que existen clases que pertenecen a los dos mundos y realizan funciones similares:

FIGURA 3-24. Estructura de paquetes y clases del mundo estático.

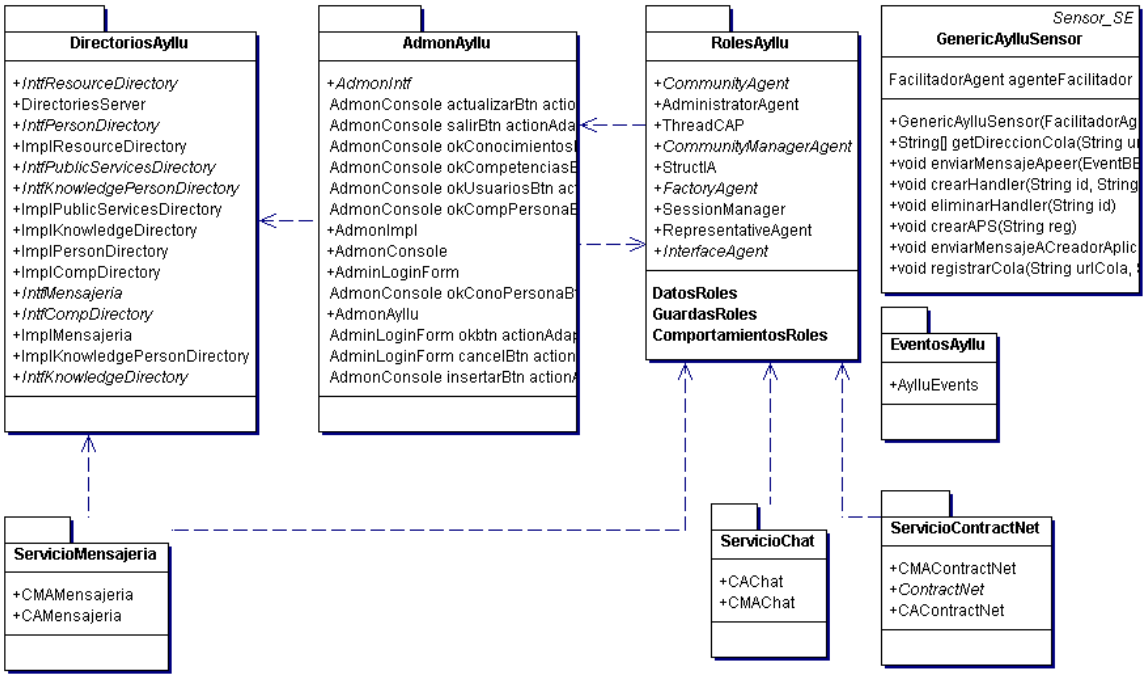
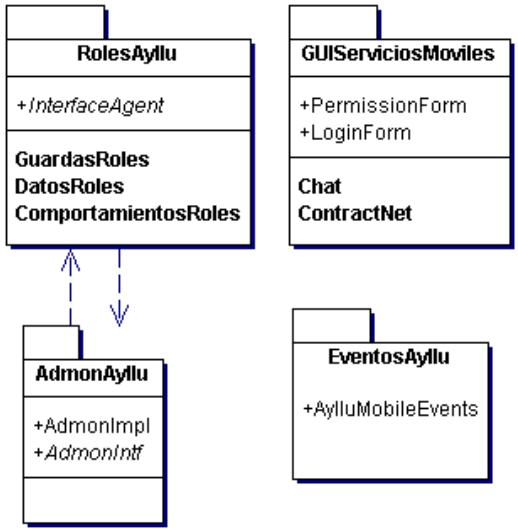


FIGURA 3-25. Estructura de paquetes del mundo dinámico.



✓ *Implementación de componentes de la arquitectura.* Ayllu también utiliza otros componentes para apoyarse en las tareas cooperativas que son: el directorio de personas, la interfase de administración y el componente de administración, los componentes GUI orientados a los dispositivos móviles, el sensor Ayllu, el thread de reporte IA y el thread de verificación CAP.

El directorio de personas: el directorio se implementó utilizando la tecnología Java RMI, donde se definieron varias interfaces remotas que proporcionaban los servicios de directorios y sus implementaciones respectivas. Las interfaces remotas son:

IntfCompDirectory.java, IntfPersonDirectory.java, *IntfKnowledgeDirectory.java, IntfKnowledgePersonDirectory.java, IntfPublicServicesDirectory.java* e *IntfResourceDirectory.java*; las clases que implementan estas interfaces remotas son: *ImplCompDirectory.java* y *ImplPersonDirectory.java, ImplKnowledgeDirectory.java, ImplKnowledgePersonDirectory.java, ImplPublicServicesDirectory.java* e *ImplResourceDirectory.java;* todas éstas trabajan en conjunto para completar la funcionalidad de los directorios de personas de la arquitectura Ayllu.

La finalidad del directorio de personas es permitir una comunicación con una base de datos proporcionada por la arquitectura Ayllu, donde se puedan realizar búsquedas para los distintos servicios cooperativos que se ofrecen en la arquitectura y las acciones administrativas. Al directorio de personas tiene acceso el agente manejador de comunidad CMA para la búsqueda de candidatos para participar en una tarea cooperativa y el componente de administrativo explicado posteriormente.

La interfase de administración y el componente administrativo: adicionalmente al agente de manejador de comunidad CMA, al directorio de personas también tiene acceso *el componente administrativo*, representado por una clase java, llamado *AdmonAyllu.java* y ubicada en el lado estático; es manejada por el agente administrador AA, quien por medio de éste puede realizar las operaciones administrativas de login, autenticación y CRUD de usuarios y servicios.

Para que el agente administrador AA pueda realizar alguna acción sobre los directorios de personas, se debe apoyar en otros componentes que son una interfaz e implementación, *AdmonIntf.java y AdmonImp.java* respectivamente, localizadas en el mundo dinámico y que en adelante serán llamadas *la interfase de administración*, ésta le brinda transparencia para las tareas administrativas al agente administrador AA. La implementación de esta interfaz envía eventos BESA que hacen reaccionar el agente administrador AA, quien delega todo el acceso hacia directorio de personas al componente administrativo descrito anteriormente.

Componentes GUI: la arquitectura Ayllu proporciona una serie de interfaces visuales para poder utilizar fácilmente algunos de los servicios implementados. Estas interfaces están orientadas hacia los dispositivos móviles y cumplen con el estándar MIDP 2.0 de java. A continuación se introducen en detalle estas interfaces:

- Interfaz de login: ésta permite reutilizar la pantalla de login de Ayllu, la lógica de los botones debe ser implementada en el midlet principal de la aplicación. La clase java que se proporciona se llama *LoginForm.java* y se encuentra en el paquete *Ayllu.GUIServiciosMoviles*.

- Interfaz de permisos: ésta permite mostrar una pantalla cada vez que se invoca algún servicio, permitiendo saber que algún usuario desea que se coopere con él. La lógica de los botones debe ser implementada en el midlet principal de la aplicación. La clase java que se proporciona se llama *PermissionForm.java* y se encuentra en el paquete *Ayllu.GUIServiciosMoviles*.

- Interfaz de chat: ésta permite el manejo del servicio de chat. Se compone de 3 clases java que son las siguientes: *ChatPacket.java*, *MessageUI.java*, e *InputUI.java*. éstos tres componentes son reutilizados de otra aplicación llamada BlueChat que puede ser encontrada en la siguiente dirección web: http://www.benhui.net/bluetooth; solamente estas tres interfaces fueron utilizadas de ese proyecto. Cada una de las clases presenta una funcionalidad diferente y pueden ser encontradas en el paquete *Ayllu.GUIServiciosMoviles.Chat*.

  La primera de ellas *ChatPacket.java*, es una clase que encapsula la lógica de un mensaje que será utilizado en el chat. La segunda de ellas *MessageUI.java*, presenta la interfaz gráfica del chat, la lógica de los botones que ésta clase proporciona debe ser implementada en el midlet principal de la aplicación. La tercera *InputUI.java*, es una interfaz especializada en la escritura de los mensajes para enviar a la clase *MessageUI.java*, una vez más la lógica de los botones debe ser implementada en el midlet de la aplicación.

- Interfaz ContractNet: ésta permite el manejar el servicio de ContractNet. Se compone de cinco clases dos de las cuales el programador debe sacar una herencia, para poder manejar los datos específicos del tipo de ContractNet que está implementado para una aplicación en Ayllu.

  Este conjunto de clases se encuentra en el paquete *Ayllu.GUIServiciosMoviles.ContractNet*. La lógica de los botones de las clases debe ser implementada en el midlet principal de la aplicación. Las clases son en su orden *Offer.java*, *OfferForm.java*, *ProposalForm.java*, *OfferAcceptedForm.java* y *Proposal.java*. Las clases *Offer.java*, *OfferAcceptedForm.java* y *Proposal.java* son destinadas a mostrar respectivamente a los usuarios la oferta, el mensaje de aceptación o rechazo de la oferta y la propuesta de ContractNet. Las clases *OfferForm.java*, y *ProposalForm.java* son aquellas de las que el programador debe heredar para especificar los datos del ContractNet que implemente para la aplicación.

El sensor Ayllu: éste componente es heredado de la arquitectura MAD [BA2004] y consiste en una clase java llamada *GenericAylluSensor.*java, que se puede encontrar en el paquete *Ayllu* y se ubica en el mundo estático. Éste hereda de Sensor_SE de MAD, su función es actuar como una fachada de mediación receptora de los mensajes entre el mundo dinámico y el mundo estático. Para ver más información acerca de éste sensor remitirse a la arquitectura MAD [BA2004].

Thread de reporte IA: este thread es parte del agente interfaz IA, su función es notificar al agente manejador de sesión SM el estado del CAP (desconectado, vivo o activo), con el fin que el agente manejador de sesión SM pueda saber con certeza a que agente interfaz enviar peticiones o mensajes. Para lo anterior el agente manejador de sesión SM, se debe apoyar en otro thread que se explicará a continuación.

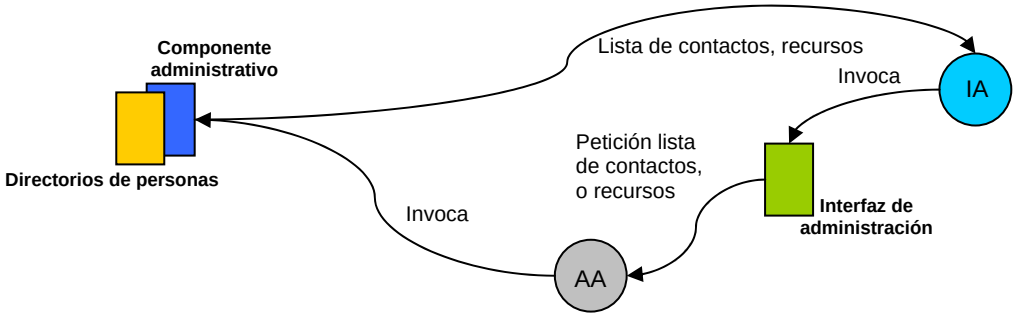
Thread de verificación CAP: este thread apoya al agente manejador de sesión SM a verificar la tabla de agentes IA que el agente contiene. Su finalidad es refrescar esta tabla para que el agente manejador de sesión SM siempre tenga la información actualizada de los estados del CAP que envía un agente interfaz IA.

✓ *Implementación de servicios cooperativos de Ayllu.* Como se explicó en el capítulo de diseño conceptual de la arquitectura, Ayllu proporcionaría una serie de servicios cooperativos orientados al groupware, siempre teniendo presente el paradigma de las 5C. De toda esa gama de servicios propuestos se decidieron implementar cinco, que fueron seleccionados debido a que representaban las principales funcionalidades de la arquitectura. A continuación se introducen de manera detallada.

Grupo de interés y detección de presencia: éste servicio pertenece al grupo de servicio de directorio de personas. Permite encontrar a otros usuarios de un determinado grupo en el Ayllu y agregarlos a la lista de contactos que cada usuario tiene. Existe la posibilidad de cambiar el estado en el que se encuentra el usuario, permitiendo que cuando en otros servicios se requiera de la presencia de dicho usuario se pueda saber en que estado se encuentra.

En éste servicio intervienen el agente interfaz IA y el agente administrador AA. El primero de ellos invoca el servicio a la interfaz de administración que se encuentra en el mundo dinámico. La interfaz java *AdmonIntf.java* recibe la solicitud y manda un evento BESA, por intermedio de la implementación de la interfaz *AdmonImpl.java*, para que reaccione el agente administrador AA, quien atiende la solicitud y delega el trabajo pesado al componente administrativo representado por la clase *AdmonAyllu.java*, éste realiza las operaciones respectivas en el directorio de personas y luego retorna la respuesta, en este caso la lista de contactos o la lista de usuarios agrupados por conocimiento, al agente interfaz IA del usuario que invocó el servicio. Éste proceso se puede ver representado en la figura 3-26.

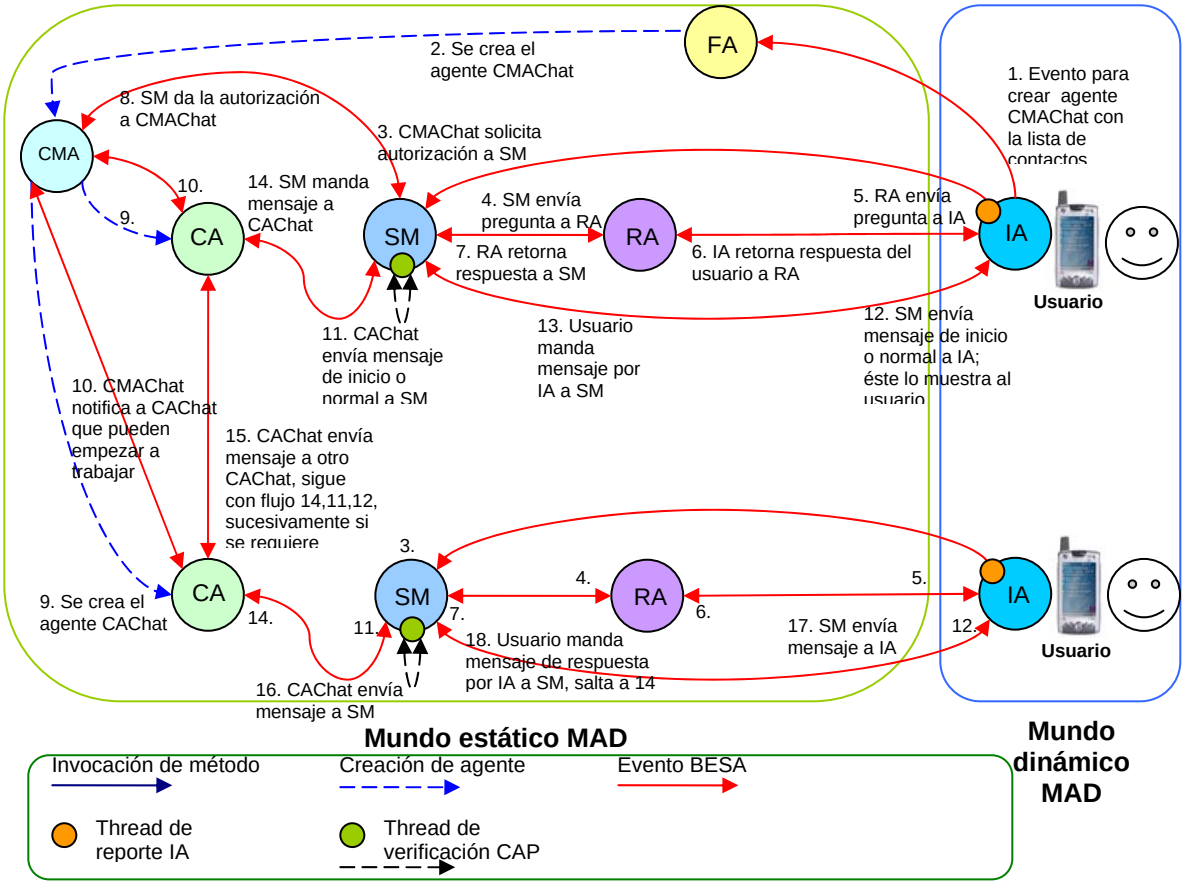
FIGURA 3-26. Servicio grupo de interés y detección de presencia y búsquedas distribuidas



Búsquedas distribuidas: éste servicio también pertenece al grupo de servicio de directorio de personas. Permite buscar un recurso dentro del Ayllu. En éste servicio intervienen el agente interfaz IA y el agente administrador AA. El primero de ellos invoca el servicio a la interfaz de administración que se encuentra en el mundo dinámico. La interfaz java *AdmonIntf.java* recibe la solicitud y manda un evento BESA, por intermedio de la implementación de la interfaz *AdmonImpl.java*, para que reaccione el agente administrador AA, quien atiende la solicitud y delega el trabajo pesado al componente administrativo representado por la clase *AdmonAyllu.java*, éste realiza las operaciones respectivas en el directorio de personas y luego retorna la respuesta, en este caso el recurso a buscar, al agente interfaz IA del usuario que invocó el servicio. Éste proceso se puede ver representado en la figura 3-26, con la variación que lo que se solicita es un recurso y lo que retorna no es la lista de contactos si no la lista de recursos.

Servicio de chat: éste servicio pertenece al grupo de servicios de comunicación. Permite establecer una comunicación con algún o algunos de los usuarios de la lista de contactos. Su principal función es apoyar a cualquier otro servicio cooperativo dentro del Ayllu, estableciendo una comunicación entre los usuarios. En éste servicio intervienen el agente interfaz IA, el agente fábrica FA y los agentes del paquete *Ayllu.ServicioChat*, que son herencia del agente de comunidad CA y el agente manejador de comunidad CMA. Éstos son *CAChat.java* y *CMAChat.java* respectivamente. Para este servicio primero se invoca automáticamente el servicio de grupo de interés y detección de presencia, trayendo la lista de contactos del usuario que invoca el servicio chat como se puede observar en la figura 3-26. Luego el conjunto de éstos agentes anteriormente descritos realiza el proceso de creación de grupos volátiles como se puede apreciar en la figura 3-27. Adicionalmente se utilizan los componentes del paquete *Ayllu.GUIServiciosMoviles.Chat* para la interfaz gráfica del servicio.

FIGURA 3-27. Servicio de chat.



Distribución de tareas por Contract Net: éste servicio pertenece al grupo de servicios de cooperativos. Permite distribuir tareas mediante un protocolo de oferta y demanda, donde algún usuario envía una propuesta y utilizando los grupos volátiles, otros usuarios que puedan cooperar envían ofertas que son analizadas, escogiendo finalmente a la o las ganadoras. Éstas son presentadas al usuario que envió la propuesta, quien es en última instancia si decide aceptarla y realizar el "contrato" con el oferente ganador o rechazarla.

En éste servicio intervienen el agente interfaz IA, el agente fábrica FA y los agentes y la clase del paquete *Ayllu.ServicioContractNet*; los agentes son herencia de los agentes de comunidad CA y el agente manejador de comunidad CMA. Éstos son *CAContractNet.java* y *CMAContractNet.java*; la clase *ContractNet.java* del paquete, es una clase abstracta que implementa el manejo de las ofertas y propuesta y además es la única de la cual el programador debe heredar para especificar la forma de escogencia de la mejor oferta a una propuesta. En las figura 3-28 se muestra claramente el proceso para realizar el servicio contractNet, la cual se apoya en la figura 3-29 donde se puede apreciar cómo el agente manejador de comunidad busca en los directorios de personas y cómo el agente de comunidad CA con rol manager interactúa con la clase contractNet; adicionalmente se utilizan los componentes del paquete *Ayllu.GUIServiciosMoviles.ContractNet* para la interfaz gráfica del servicio.

FIGURA 3-28. Servicio de contractNet.

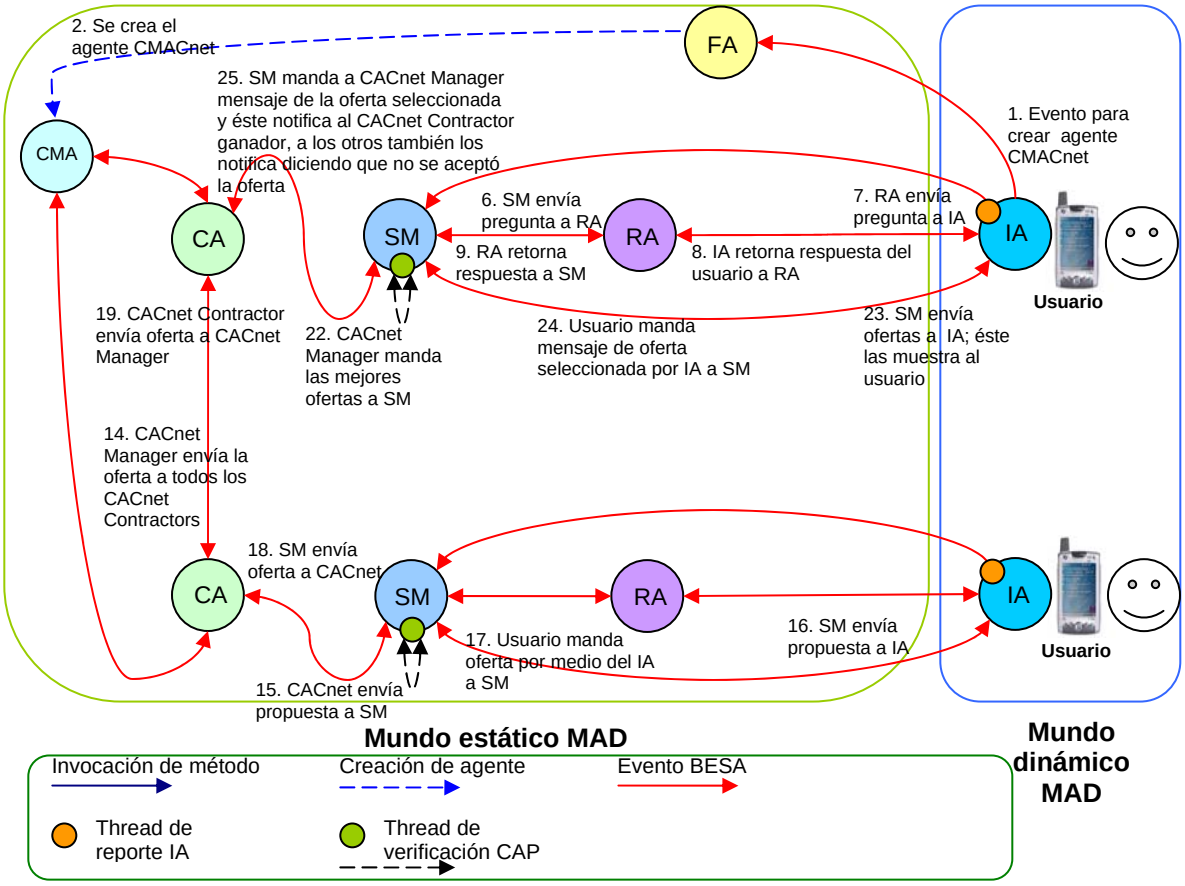
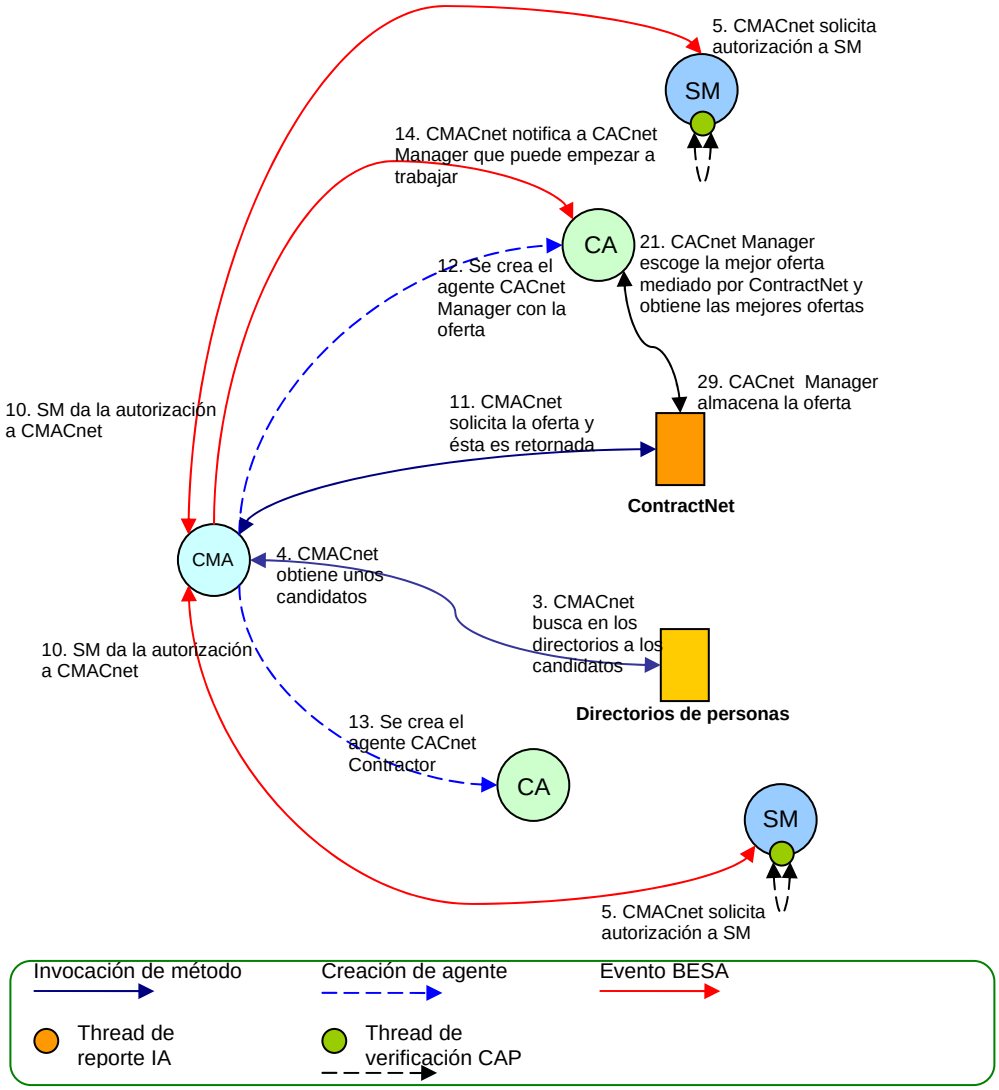


FIGURA 3-29. Servicio de contractNet – interacción con los directorios y la clase contractNet.

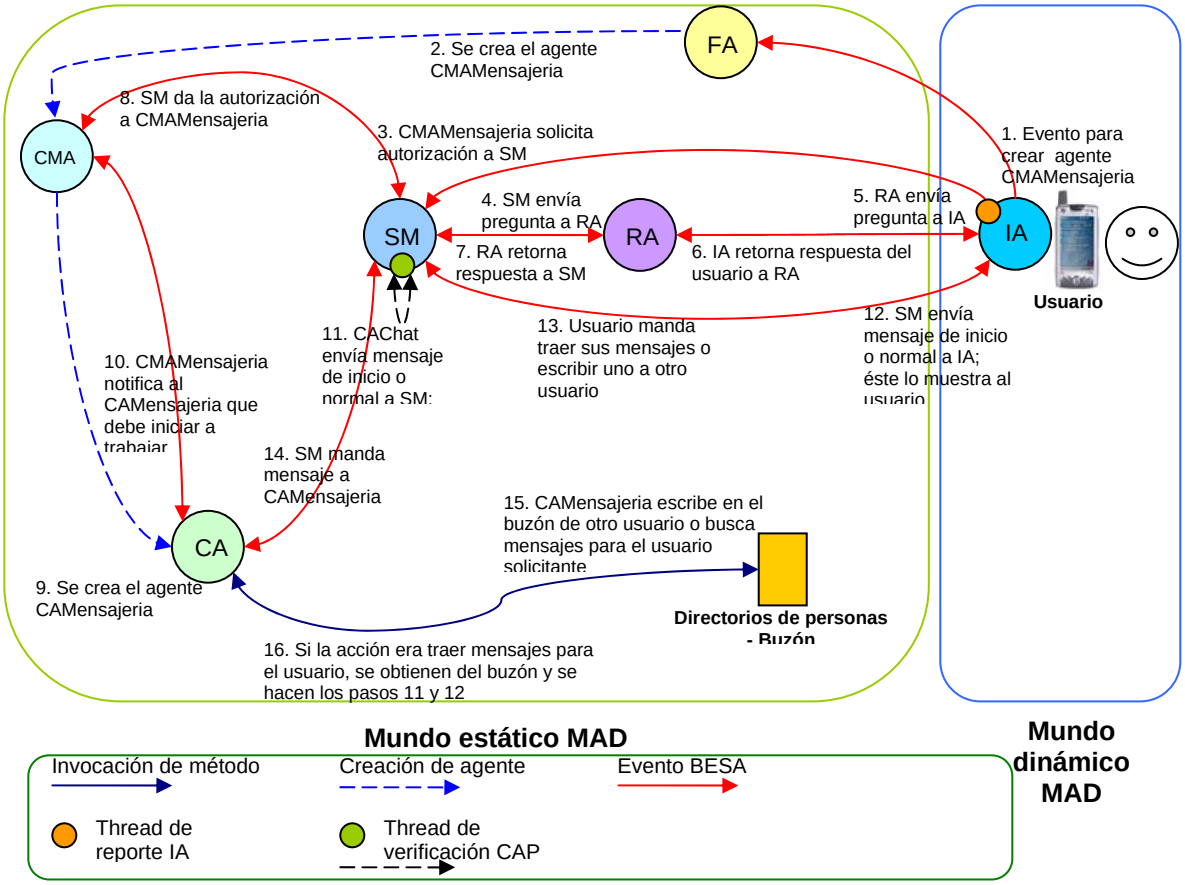


Servicio de mensajería: este servicio pertenece al grupo de servicios de comunicación. Permite que el usuario pueda revisar cuando este desee, un buzón de mensajes. Otros usuarios pueden escribirle mensajes y éstos quedan guardados en el buzón del usuario destinatario.

El servicio se implementó mediante el uso de la tecnología java RMI, definiendo en el paquete *Ayllu.DirectoriosAyllu* la interfaz *IntfMensajeria.java* y la implementación *ImplMensajeria,java*, las cuales se encargan de realizar las interacciones en el directorio de personas donde residen los mensajes de los usuarios. Para utilizar este servicio se debe utilizar los agentes del paquete *Ayllu.ServicioMensajería* que son el agente manejador de comunidad CMA *CMAMensajeria.java* y el agente de comunidad *CAMensajería.java*.

El servicio de mensajería utiliza el mecanismo de grupos volátiles, con la única diferencia que se crea un agente manejador de comunidad CMA para cada usuario y un solo agente de comunidad CA. En otras palabras crea un grupo volátil para cada usuario y el agente de comunidad CA se encarga de leer o escribir en el buzón. En la figura 3-30 se puede apreciar el servicio de mensajería descrito anteriormente.

FIGURA 3-30. Servicio de mensajería.

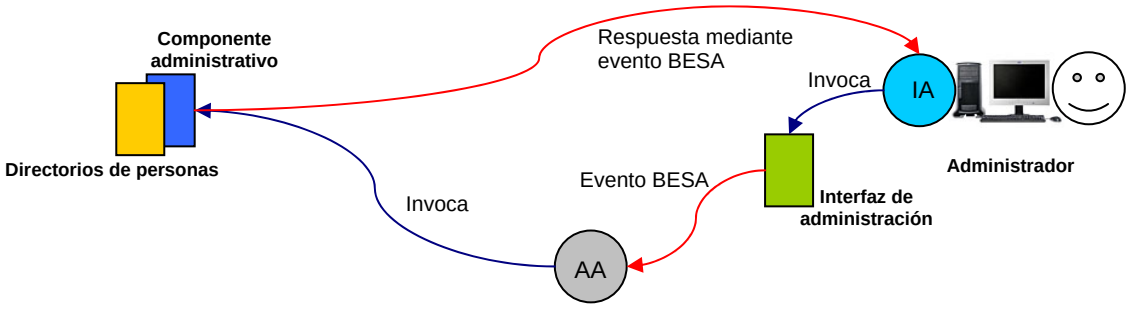


✓ *Implementación del administrador.* El administrador fue desarrollado como una aplicación del lado estático, la cual tiene asociado un agente interfaz IA que se debe implementar heredando del agente *InterfaceAgent.java* del paquete *Ayllu.RolesAyllu*. La aplicación se encuentra en el paquete *Ayllu.AdmonAyllu* y se llama *AdmonConsole.java* para acceder a ella se implementó la forma de logueo, *AdminLoginForm.java*, ubicada en el mismo paquete. Para poder utilizarlas, se debe crear una clase que instancia a *AdminLoginForm.java.*

Entre las funciones implementadas para el administrador se encuentran realizar operaciones CRUD sobre los usuarios, sobre las competencias y sobre los conocimientos. Adicionalmente se incluyó la funcionalidad de asignar, eliminar, seleccionar y modificar competencias y asignar, eliminar y seleccionar conocimientos a usuarios.

Para que el administrador pueda realizar estas operaciones, se utilizan las clases del paquete *Ayllu.AdmonAyllu* llamadas *AdmonIntf.java y AdmonImpl.java*, quienes conforman *la interfaz administrativa* del mundo estático. El funcionamiento se puede apreciar en la figura 3-31, donde un agente interfaz IA del lado estático realiza las peticiones a la interfaz administrativa, ésta envía un evento BESA al agente administrador AA, quien delega el procesamiento *al componente administrativo*. Luego de realizar las operaciones respectivas sobre el directorio, *el componente administrativo* envía un evento BESA al agente interfaz IA que realizó la petición, para que éste se lo presente al administrador.

FIGURA 3-31. Administrador en la arquitectura Ayllu



Mundo estático MAD

3.4.5 Pruebas realizadas a la arquitectura. Las pruebas realizadas a la arquitectura consistieron en probar la comunicación básica entre los agentes, con el fin de verificar el correcto funcionamiento de todos los componentes implementados.

Entre las pruebas realizadas se encuentran las que se pueden apreciar en la tabla 3-2, que incluyen la comunicación entre los agentes y la comunicación de agentes con los componentes de la arquitectura. Los resultados obtenidos fueron exitosos y en el siguiente capítulo se estudiarán con mayor detalle, validados a través del caso de estudio.

TABLA 3-2. Pruebas funcionales realizadas a la arquitectura.

Prueba	Agentes o componentes involucrados	Descripción	Resultado
Comunicación entre agentes del mundo estático	Todos los agentes del mundo estático	Envío de request-reply de unos a gentes a otros.	Satisfactorio
Comunicación entre agente Interfaz del mundo dinámico y agente representante RA y agente manejador de sesión SM del mundo estático y viceversa	Agente interfaz IA, agente representante RA y agente manejador de sesión SM	Envío de reply sobre alguna petición por parte del agente interfaz IA al agente representante RA. Envío de mensajes por parte del agente interfaz IA al agente manejador de sesión SM. Envío de mensajes por parte del agente manejador de sesión SM al agente interfaz IA.	Satisfactorio

		Envío de request desde el agente representante RA hacia el agente interfaz IA	
Comunicación por medio de RMI desde el agente manejador de comunidad CMA hacia los directorios de personas	Agente manejador de comunidad CMA y directorios de personas	El agente manejador de comunidad realiza una petición al directorio de personas y éste le responde con la información solicitada	Satisfactorio
Comunicación de agentes de comunidad de un mismo grupo volátil	Agentes de comunidad CA	Los agentes de comunidad CA de un grupo volátil se comunicaron entre si por medio de la clave generada por el agente manejador de comunidad CMA. Otros agentes de otros grupos volátiles no se comunicaron con otros de otro grupo volátil	Satisfactorio
Comunicación entre el agente administrador AA y el componente de administración	Agente administrador AA y componente de administración	El agente administrador AA se comunicó con el componente de administración enviándole una petición, para que se realizara alguna labor sobre los directorios de personas	Satisfactorio
Creación del agente manejador de comunidad por parte del agente fábrica FA.	Agente fábrica FA y agente manejador de comunidad CMA	El agente fábrica FA crea el agente manejador de comunidad CMA para algún servicio cooperativo	Satisfactorio
Creación de agentes de comunidad CA para un grupo volátil	Agente de comunidad CA y agente manejador de comunidad CMA	El agente manejador de comunidad CMA crea a los agentes de comunidad de algún servicio cooperativo, para que conformen un grupo volátil.	Satisfactorio

4. CASO DE ESTUDIO

Con el fin de poder probar la arquitectura Ayllu, se ha decidido construir una aplicación de caso de estudio, que pueda probar al menos los servicios más importantes que ésta ofrece. Por esta razón en la selección del caso de estudio, se debe encontrar un tipo de aplicación sea un reto que incluya los requerimientos identificados para Ayllu en el capítulo de implementación de la arquitectura.

Mediante el estudio de otras arquitecturas se identificaron varias propuestas de caso de estudio, que eran soportadas por diferentes arquitecturas. Estas propuestas se analizan más adelante.

Sin embargo, con éstas aplicaciones no se podrían validar los servicios más significativos de Ayllu (mencionados en el capítulo del modelo conceptual de la arquitectura), a excepción de lo que propone [RM2003], con su proyecto de un Agente Asesor e Biblioteca, que aunque no sería exactamente la misma aplicación, serviría como punto base para proponer alguna otra.

Para seleccionar el caso de estudio más apropiado, se debe tener en cuenta que la arquitectura Ayllu busca orientar el groupware con el enfoque del paradigma de las 5C, mediada por agentes de software y manejando a su vez los problemas de los dispositivos móviles, por lo tanto, el caso de estudio debe cumplir y validar estos requisitos.

Teniendo en cuenta lo anteriormente mencionado, se identificaron tres casos de estudio posibles, que son los siguientes:

- Una aplicación en el campo de la medicina.
- Una aplicación para bibliotecas.
- Una aplicación para fuerza de ventas.

Durante el desarrollo de este capítulo se esbozarán cada una de las aplicaciones encontradas durante el estudio de otras arquitecturas y los casos de estudio candidatos; luego se establecerá cual es el caso de estudio seleccionado justificándolo dentro del contexto de Ayllu, para luego entrar a describir el proceso de diseño e implementación del mismo. Finalmente, se mostrarán los resultados de las pruebas realizadas para validar la arquitectura.

4.1 APLICACIONES DE LAS ARQUITECTURAS ESTUDIADAS PARA EL ENTORNO MÓVIL

Durante el proceso de análisis de arquitecturas se encontraron varias aplicaciones que se perfilan como posibles casos de estudio de nuestro proyecto, o por lo menos nos ofrecen ideas para poder elegir uno apropiado para validar la arquitectura propuesta.

4.1.1 Calendario Colaborativo [RJ2001]. Dada la situación en que varias personas poseen un dispositivo handheld quieren programar reuniones de trabajo en un futuro, se necesita una aplicación que permita hallar un espacio en el cual todos los miembros estén disponibles para agendar la cita.

Cada dispositivo contiene la información personal de su respectivo dueño, con todas las entradas que indican si está disponible o no y cual es el motivo.

Cuando las personas se reúnan para coordinar sus citas, cada uno podrá visualizar su propia agenda y la de su(s) compañeros. Cada una está debidamente etiquetada para identificarla de las demás.

4.1.2 Recolector de Tarjetas de Presentación [RJ2001]. Las tarjetas de presentación en los negocios generalmente se almacenan y guardan una sola vez sin volver a ser modificadas.

Esto da pie a la idea de una aplicación que permita publicar este tipo de tarjetas y cada vez que alguien entre a una reunión, el programa descargará la lista de todas las tarjetas de presentación que han sido publicadas, permitirá seleccionar las que sean de interés y almacenándolas de manera persistente.

4.1.3 Agente Asesor de Biblioteca [RM2003]. Cuando una persona anota en su lista de pendientes de su PDA el nombre de un libro de interés para sacar de la biblioteca, se activa un agente de software encargado de recordarle a la persona que debe sacar dicho libro.

Cuando la persona vaya a la biblioteca, el agente comenzará las negociaciones con el agente local de la biblioteca, quien le dará la información del libro, su disponibilidad y ubicación. El agente residente en la PDA no se desactivará hasta que no se confirme que se ha pedido en préstamo el libro. Cuando esto suceda, se borrará la anotación de la lista de pendientes.

4.1.4 Servicio de publicidad replicada [RO2003]. La arquitectura eNcentive fue diseñada para ilustrar un modelo de mercadeo móvil, que utiliza un Framework de agentes, el cual permite la replicación de publicidad utilizando comunicación directa P2P.

La publicación ha sido probada en el siguiente escenario: un almacén con el sistema eNcentive envía inalámbricamente una promoción al dispositivo móvil de un cliente que pasa cerca o visita el negocio. El dispositivo móvil está configurado para recibir cierto tipo de ofertas, con lo que se evita el spam y procesamiento innecesario del dispositivo.

La persona que ha recibido el cupón u oferta en su dispositivo móvil puede replicar inconscientemente dicha publicidad a otros dispositivos con el sistema eNcentive, siempre y cuando esos dispositivos acepten su oferta. El hecho de replicar la oferta que

inicialmente había adquirido una persona en el negocio, permite según el sistema eNcentive recibir una mayor bonificación cuando se vaya a redimir el cupón.

4.1.5 Colaboración por medio de dispositivos móviles a través de redes Ad-hoc, en situaciones de emergencia [KS2004]. En situaciones de emergencia existen una gran cantidad de dispositivos móviles que pueden usar el personal de emergencias. Según Stan Kurkovsky [KS2004], la idea es aplicar su modelo de resolver tareas en un esquema de computación de red, en situaciones de emergencias, para intercambiar datos o códigos.

Actualmente la arquitectura fue probada como un simulador de distribución de la carga computacional de tareas, teniendo en cuenta la movilidad de los dispositivos.

4.1.6 Colaboración sobre redes ad-hoc utilizando la arquitectura YCab [BD2001]. La arquitectura YCab fue una de las más robustas que se analizaron en este documento, proporcionaba servicios colaborativos interesantes sobre redes ad-hoc. Algunas de las aplicaciones podrían ser las siguientes:

- Esfuerzos de rescate en situaciones de emergencia son fácilmente coordinables en la ausencia de redes de topología fija.
- Grupos atendiendo a una conferencia podrían realizar durante esta una reunión virtual, para realizar comentarios entre si.
- Inteligencia militar y equipos de asalto, podrían ser más fácilmente coordinados, para obtener un mejor tiempo de respuesta.
- En operaciones de supervivencia donde no haya una red de topología fija.
- Un equipo de construcción trabajando en un lugar sin una red de topología fija, podrían compartir planos y esquemas.
- El staff de seguridad en la organización de un evento, manejando la seguridad y el acceso de personas.

Actualmente la arquitectura YCab podría implementarse en cualquiera de estos escenarios, pues una de sus características es la flexibilidad para adaptarse a cualquier escenario.

4.1.7 RCSM en entornos educativos [BD2001]. RCSM fue probada para identificar, analizar y realizar una acción el contexto educativo en los siguientes casos:

- En una exposición el dispositivo móvil del maestro capta que está cerca del video beam, adicionalmente capta la ausencia de luz. El contexto indica al dispositivo móvil que se trata de una presentación y entonces la PDA distribuye el material a los estudiantes, sin ninguna interacción por parte del instructor.
- Durante una discusión de grupo, el instructor se mueve de un grupo de estudiantes a otro, para chequear el progreso de los estudiantes. Cuando el instructor se encuentra cerca de uno de los grupos, el dispositivo móvil interpreta esta actividad como un interés en el grupo y esto permite que el instructor se una

al grupo de estudiantes por un corto tiempo para bajar el material de discusión que han desarrollado los estudiantes.

4.1.8 DreamView y Designer [RJ2000]. DreamView es un web browser cooperativo desarrollado sobre la arquitectura DreamTeam. Esta aplicación permite a un grupo de usuarios navegar Internet cooperativamente. Adicionalmente esta aplicación permite a los usuarios hacer anotaciones en las páginas web; estas pueden ser vistas por todos los participantes de la sesión.

Otra aplicación para validar la arquitectura DreamTeam es una herramienta para diseñar cooperativamente diseños como diseños de entidad relación, llamado Designer. Ésta aplicación permite manipular diagramas, objetos e imágenes, para que puedan ser editadas y rediseñadas fácilmente.

4.2 CASOS DE ESTUDIO CANDIDATOS

4.2.1 Una aplicación en el campo de la medicina. Por su naturaleza implícita, el trabajo en medicina implica colaboración, comunicación y coordinación, principios Fundamentales del Groupware y de la arquitectura Ayllu. La aplicación que se ha pensado desarrollar en este campo, pretende explotar estos 3 conceptos.

Básicamente esta aplicación estará orientada al personal médico y busca agilizar las tareas de una manera más coordinada, compartir recursos e ideas, generar conocimiento, mediante grupos de expertos, a los cuales se les pueda consultar inquietudes del área. Todos estos son requerimientos que se pueden cumplir por medio de los servicios que ofrece Ayllu.

Sin embargo, buscando darle un sentido social, se pueden incluir dentro de este contexto a los pacientes, quienes serían otros actores que interactuarían con la aplicación, mediante el uso de servicios que les permitan acceder de forma más rápida a especialistas, sin la necesidad del desplazamiento de grandes distancias, para realizar preguntas, o por ejemplo para realizar trámites administrativos, como sacar citas con médicos o consultar servicios médicos, como radiografías o exámenes de laboratorio. Otra posibilidad, es ver a los pacientes como beneficiarios de la aplicación, dado que los médicos pueden emplear la aplicación con el fin de realizar reuniones virtuales en donde se emiten diagnósticos de manera grupal o se negocian visitas reales entre especialistas, agilizando proceso de tratamiento de enfermedades o urgencias en los pacientes.

Esta tipo de aplicación permitiría validar de manera apropiada la arquitectura Ayllu, debido a que involucra trabajo en grupo, comunicación entre distintas personas para lograr un objetivo y manejo de recursos pero, es probable que debido a la gran cantidad de servicios que se pueden implementar para ésta aplicación, se deban seleccionar solamente los más significativos con el fin de validar la arquitectura.

Por último, todas estas operaciones serían soportadas por dispositivos móviles en principio, para lograr explotar al máximo las características de movilidad y productividad que éstos ofrecen, así como proporcionar comunicación en cualquier momento y lugar, dentro del área de cobertura de la señal en el hospital.

4.2.2 Una aplicación para las Bibliotecas. En la actualidad, las bibliotecas han dejado de ser un lugar de simple consulta; gracias a los avances tecnológicos, las bibliotecas se han convertido en sitios donde es posible interactuar con otras personas a través salas de reunión e Internet, incluso es posible sacar provecho de una gran variedad de material interactivo, e incluso acceder a Internet a través de los Access Point que muchos de estos sitios tienen instalados.

Por estas razones, se considera que es necesario proporcionar un mecanismo que promueva y facilite las labores de trabajo en grupo. Una aplicación de Groupware es una muy buena alternativa de solución ya que permitiría formar grupos de estudio y consulta para tareas de investigación; también podría apoyar las labores administrativas de la biblioteca.

Bajo esta perspectiva, sería factible poder probar los servicios que proporciona Ayllu: los grupos de usuarios utilizarían el servicio de grupo y detección de presencia, así como los de búsqueda distribuida. De igual manera, se pueden crear salas virtuales de discusión soportadas en el servicio de chat, así como servicios de suscripción y notificación para saber cuando llega un libro concerniente a un tema de interés para el usuario. Las labores administrativas se verían beneficiadas por los mecanismos de resolución de conflictos, colaboración y coordinación para la realización de tareas por parte de los empleados, según se requiera.

Este tipo de aplicativos no tienen una visión social tan evidente como la aplicación de medicina si se observa a primera vista; su valor social está enfocado hacia la promoción de la investigación y aprovechamiento de los recursos de las bibliotecas, razón que es consecuente con los planes de desarrollo iniciados por las últimas administraciones locales y su iniciativa de crear más bibliotecas.

4.2.3 Una aplicación para Fuerzas de Ventas móviles. Hoy en día las organizaciones han dejado de ser estáticas para ser mucho más dinámicas y llevar sus productos y servicios hacia los clientes. Como consecuencia de esta nueva tendencia, nacieron las fuerzas de ventas móviles; estas emplean dispositivos móviles como soporte esencial de sus labores diarias.

La oportunidad que se ha identificado en éste caso, va orientada a la distribución y ejecución de tareas. Una empresa tiene una oficina central desde donde coordina las labores de cada uno de los miembros de su fuerza de ventas; esta oficina central tiene la responsabilidad de delegar las labores y coordinarlas para obtener un resultado al final del día; esto se relaciona de manera explícita con la funcionalidad que el Groupware mediado por agentes de Ayllu.

El administrador puede delegar tareas al miembro del grupo (trabajadores de la fuerza de ventas), quienes las recibirán en sus dispositivos y reportarán su estado al final del día. Cada usuario puede interactuar con los demás compartiendo documentos (como por ejemplo facturas), realizando búsqueda distribuida de información común para todos y detectando la presencia de algún compañero.

Todas estas características, resultan en una mayor productividad, al atender a un mayor número de clientes en poco tiempo y en satisfacción al cliente, porque sus necesidades serán atendidas más rápidamente. El administrador podrá tener un mejor control sobre las operaciones de su grupo y adicionalmente, puesto que Ayllu permitirá comunicación entre diferentes grupos, se puede dar un mejor soporte a los clientes de la empresa.

4.3 ALTERNATIVA SELECCIONADA

El caso de estudio seleccionado para validar la arquitectura Ayllu, es la aplicación en el campo de la medicina. Para llegar a esta decisión, se tomaron como base en los posibles casos de estudio mencionados anteriormente y se realizó un proceso de selección con el fin de escoger el más apropiado, teniendo en cuenta los siguientes criterios que se deben cumplir para Ayllu:

- Requerimientos de groupware Ayllu:
    - Cooperación: el caso de estudio debe proporcionar mecanismos que fomenten y faciliten la cooperación entre usuarios.
    - Colaboración: los usuarios pueden distribuir tareas entre ellos para conseguir un objetivo; el caso de estudio debe ofrecer mecanismos idóneos para este fin.
    - Coordinación: se deben proporcionar mecanismos para articular las acciones individuales de cada individuo, para lograr que el objetivo final sea coherente.
    - Resolución de Conflictos: el caso de estudio debe ofrecer mecanismos de negociación para solucionar conflictos entre usuarios, o para el acceso a recursos compartidos o limitados.
    - Comunicación: debe existir comunicación entre los diferentes usuarios del sistema; la comunicación es la base para la implementación del groupware enfocado dentro de paradigma de las 5C.
    - Carácter social o comercial: el caso de estudio puede tener carácter orientado al comercio y aplicaciones de uso masivo, o puede estar orientado a solucionar problemáticas de tipo social y/o humanitario.
    - Group Awareness: es el conocimiento del grupo se logra a través de los servicios de la arquitectura.

- Requerimientos de infraestructura:
    - *P2P:* las arquitecturas P2P facilitan la colaboración ad-hoc, y compartir información de manera distribuida sin la necesidad de tener un servidor específico.

- Multicapas: se deben definir como mínimo tres capas fundamentales, en las cuales se diferencia claramente la lógica, modelo de datos y presentación de las aplicaciones.
- Protocolos: en las arquitecturas orientadas a comunidades virtuales, se definen protocolos orientados al manejo de grupos y usuarios.
- Servicios: con base en los protocolos mencionados anteriormente, también se definen servicios esenciales para la colaboración.
- Manejo de sesiones: se debe garantizar que uno de los participantes inicie la sesión; una vez haya iniciado la sesión, otros participantes se pueden unir a la ella y colaborar.

Adicionalmente, se deben tener en cuenta las características de MAD [BA2004] para poder manejar las limitaciones de los dispositivos móviles, con lo cual se agregan dos nuevos criterios para ser evaluados:

- Ambientes Cerrados: el ambiente del caso de estudio debe estar delimitado por el área de cubrimiento [BA2004].
- Orientada a servicios: se deben ofrecer unos servicios dentro del ambiente cerrado, que lo describan como tal [BA2004].

Para de evaluar los criterios, se establecieron indicadores para medir los grados de satisfacción de requerimientos de cada caso candidato frente a cada uno de los criterios establecidos para poder evaluar Ayllu. Estos indicadores se muestran en la tabla 4-1 y los resultados de la evaluación se muestran en la tabla 4-2:

TABLA 4-1. Indicadores para evaluar los posibles casos de estudio

Muy Bajo	satisface el requerimiento entre el 0% y el 24.9%
Bajo	satisface el requerimiento entre el 25% y el 49.9%
Medio	satisface el requerimiento entre el 50% y el 74.9%
Alto	satisface el requerimiento entre el 75% y el 99.9%
Ideal	satisface el criterio en un 100%

TABLA 4-2. Resultados de la evaluación de los posibles casos de estudio

	Medicina	**Fuerza de Ventas**	**Biblioteca**
Requerimientos de groupware Ayllu	Ideal	Alto	Alto
Requerimientos de infraestructura	Alto	Medio	Alto
Ambiente cerrado	Ideal	Muy Bajo	Ideal
Orientado a Servicios	Alto	Medio	Alto
Carácter social o comercial	Social	Comercial	Comercial
Total	87.5%	43.75%	81.25%

Como se aprecia en la tabla 4-2, la aplicación en el campo de la medicina es la que mejor se acomoda a los criterios establecidos para evaluar Ayllu.

4.4 ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DEL CASO DE ESTUDIO: TELEMEDICINA AYLLU

Con el fin de validar de manera adecuada la arquitectura Ayllu, se definió el contexto dentro del cual el caso de estudio podría ser útil no sólo para este fin, sino también para prestar un servicio social. La aplicación fue desarrollada siguiendo los pasos y procedimientos especificados en el manual de programador de Ayllu, que se incluye como anexo de éste proyecto.

A continuación se describen el contexto de la aplicación y el proceso de análisis de requerimientos, diseño e implementación del caso de estudio.

4.4.1 Contextualización de la aplicación. La medicina ha tenido innumerables avances en los últimos tiempos, permitiendo mejorar la calidad de vida de millones de personas alrededor del mundo.

Este crecimiento no podía estar aislado del crecimiento tecnológico. La computación junto con la medicina, ofrecen una nueva gama de servicios y oportunidades tanto para pacientes como para profesionales de la salud.

Como consecuencia de estas nuevas oportunidades, nace un concepto conocido como Telemedicina que se define como un nuevo tipo de medicina en la cual se utiliza información electrónica y tecnología computacional para entregar apoyo médico a distancia [LE2001].

Sus limitaciones son:

- Capacidad tecnológica.
- Incompatibilidad entre equipos existentes.
- Especialización y conocimiento en el área informática y médica (médicos no especialistas en el área informática e informáticos no especialistas en el área médica).
- Capacidad en cuanto a redes y su masificación.
- Cambios en mentalidad.
- Plataformas tecnológicas de alta complejidad y uso especializado.
- Insuficientes directrices gubernamentales en la aplicación de estas tecnologías.

Sus ventajas son:

- Masificación de la informática.
- Aplicación de la informática a la realidad cotidiana.
- Mejor acceso a los servicios de salud.

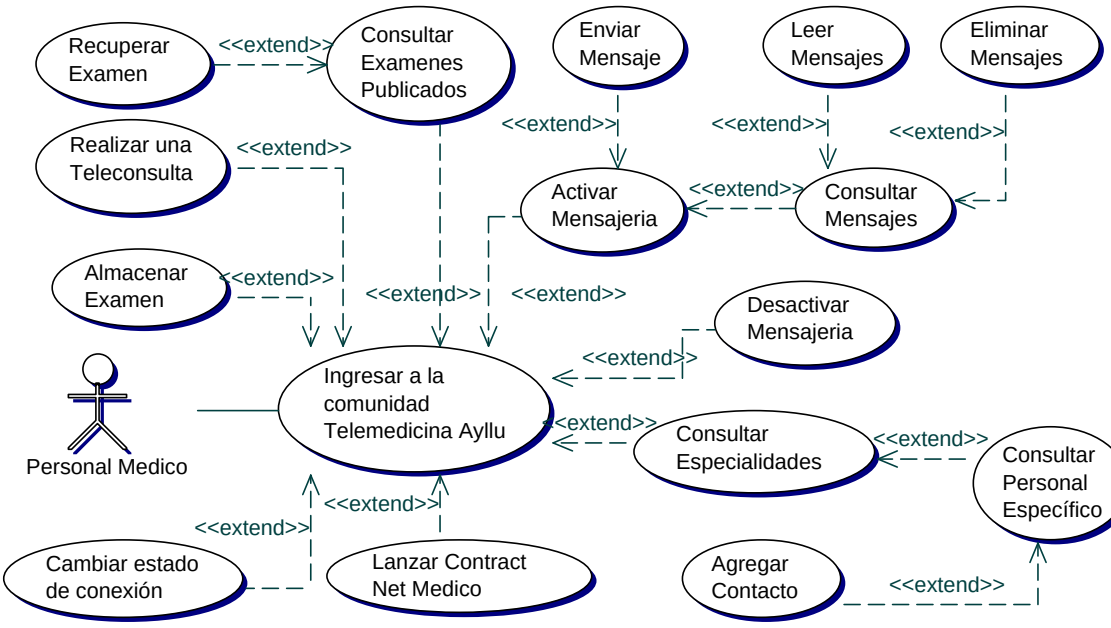
- Diagnóstico y tratamiento temprano.
- Costos reducidos.
- Acorta distancias (traslado de pacientes y personal medico).
- Mejor calidad de vida para el paciente.
- Ofrece oportunidades de investigación y educación por medio de colaboración electrónica:
- Investigación y realización de debates clínicos.
- Facilita la publicación digital.
- Permite la participación de seminarios a distancia.
- Promueve el trabajo cooperativo en la red.

Como consecuencia de los conceptos acá presentados, el caso de estudio empleado para la validación de Ayllu se denominará Telemedicina Ayllu y pretende ser un mecanismo de cooperación entre el personal médico, proporcionando los mecanismos necesarios para entregar apoyo médico al los pacientes.

4.4.2 Análisis de requerimientos para Telemedicina Ayllu. La aplicación de Telemedicina Ayllu funciona dentro de un ambiente cerrado que en este caso particular, es un hospital.

Los casos de uso identificados para la aplicación se muestran en la figura 4-1:

FIGURA 4-1. Casos de uso para Telemedicina Ayllu



- *Ingresar a la comunidad Telemedicina Ayllu:* antes de poder hacer uso de la funcionalidad de la aplicación, el usuario debe ser autenticado y autorizado utilizando el servicio de directorios Ayllu.

- *Cambiar estado de conexión:* el usuario puede establecer su estado de conexión ante otros usuarios de Ayllu, por ejemplo no disponible ó ausente.

- *Lanzar Contract Net Médico:* si se requiere ubicar a médico para que colabore con una urgencia, se lanza un Contract Net especificando los detalles de la misma; cada uno de los médicos pertenecientes a la comunidad Telemedicina Ayllu que estén conectados reciben la propuesta y responden con una oferta o en su defecto, negándose a colaborar. El sistema muestra al emisor de la propuesta las mejores alternativas permitiéndole decidir cual aceptar.

- *Realizar una Teleconsulta:* cada vez que un usuario necesite el criterio u opinión de algunos colegas que no se encuentren presentes en el momento, pero si disponibles dentro de la comunidad Telemedicina Ayllu, podrá lanzar una teleconsulta con el fin de interactuar en tiempo real y llegar a una conclusión en equipo.

- *Consultar especialidades:* un usuario puede consultar las áreas de conocimiento (especialidades) que se encuentran disponibles.

- *Consultar Personal Específico:* se puede consultar la lista de usuarios con un conocimiento específico.

- *Agregar Contacto:* un usuario puede agregar a su lista de contactos usuarios que pertenezcan a especialidades específicas, para luego poder interactuar con ellos.

- *Consultar Exámenes Publicados:* un usuario puede consultar los exámenes publicados dentro de la comunidad.

- *Recuperar Examen:* un usuario puede recuperar los resultados de un examen publicado, siempre y cuando haya sido el mismo quien lo haya hecho, de lo contrario, no se tiene acceso a la información.

- *Almacenar Examen:* un usuario puede almacenar en el repositorio de datos los resultados de un examen practicado a un paciente.

- *Activar Mensajería:* un usuario decide si quiere recibir mensajes asincrónicos de otros usuarios; esto le brinda la posibilidad de seguir interactuando de alguna manera en la comunidad sin necesidad de estar conectado.

- *Desactivar Mensajería:* se puede desactivar el servicio de mensajería si no se desea utilizarlo.

- *Enviar Mensaje:* un usuario puede enviar mensajes asincrónicos a otros usuarios luego de haber activado este servicio.

- *Consultar Mensajes:* se pueden consultar los mensajes que se encuentren almacenados en el buzón de mensajes asincrónicos.

- *Leer Mensajes:* se pueden recuperar los mensajes del buzón de mensajería asincrónica.

- *Eliminar Mensajes:* se pueden eliminar los mensajes del buzón de mensajería asincrónica

Con los casos de uso mencionados, se pueden validar y probar las características de la arquitectura Ayllu.

4.4.3 Diseño e Implementación para Telemedicina Ayllu. Para Telemedicina Ayllu, se contemplan además de los servicios cooperativos proporcionados por la arquitectura, los servicios de almacenamiento y recuperación de exámenes y el servicio de Contract Net.

El servicio de almacenamiento y recuperación de exámenes se implementó basándose en la estructura de un servicio de artefactos MAD, que consiste en un agente que accede a los servicios de recursos (AgenteAMTelemedicina), los servicios a los cuales accede éste agente (Guardar Examen y Recuperar Examen) y un peer móvil encargado de realizar las solicitudes para almacenar y recuperar los recursos (AgenteManejadorRecursos, AMR).

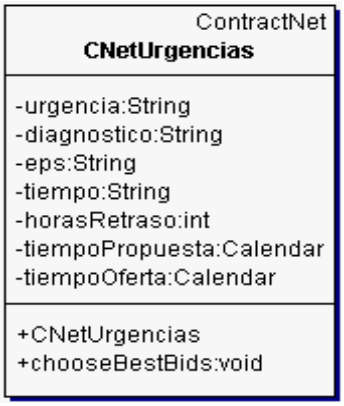
Tal como se explicó en el capítulo anterior, el servicio de Contract Net no hace parte de los servicios basicos propuestos para Ayllu, sin embargo, durante su desarrollo se pudo generar un servicio que puede ser reutilizado por otras aplicaciones, simplemente especializando una clase (clase ContractNet). Esta clase es la que le da sentido a los datos dentro del servicio mapeándolos de acuerdo a sus necesidades, y además implementa el método para elegir las mejores ofertas dependiendo del contexto dentro del cual se lleva a cabo la negociación.

Para la implementación de todas las demás características genéricas de Ayllu, se implementaron los agentes IATelemedicina, FactoryTelemedicina. El agente IATelemedicina implementa todas las guardas heredadas de *InterfaceAgent*, necesarias para manejar los eventos de la aplicación de Telemedicina Ayllu, permitiendo el acceso a los recursos base del dispositivo donde está corriendo. El FactoryTelemedicina implementa el método createCMAAgents necesario para crear los agentes CMA de cada servicio cooperativo implementado en Telemedicina Ayllu, heredado de *FactoryAgent*.

Las interacciones tanto del agente interfaz como del agente fábrica se describen en el capitulo *Implementación de la Arquitectura Ayllu* en la sección *Diseño de la arquitectura Ayllu*. La interfaz para los dispositivos móviles se desarrolló utilizando J2ME, decisión justificada en el Anexo 1. A continuación se muestra el diseño de los agentes y servicios de Telemedicina Ayllu.

- ✓ Servicio CNetUrgencias: este servicio es una clase especializada de ContractNet, que contiene los atributos y métodos necesarios para llevar a cabo un Contract Net de Urgencias. La figura 4-2 muestra el diagrama de la clase:

FIGURA 4-2. Clase CNetUrgencias



- Atributos:

    - Urgencia: es la descripción del contrato dentro del contexto de un Contract Net genérico. Para el caso específico de Telemedicina Ayllu, se refiere al tipo de urgencia que se presenta, por ejemplo accidente automovilístico o alguna enfermedad de consideración.
    - Diagnóstico: es la evaluación inicial que da el personal de urgencias al paciente y que se desea confirmar o tratar.
    - Eps: es la eps a la cual está afiliado el paciente. Los médicos que respondan la oferta deben pertenecer a la misma eps.
    - Tiempo: hora para la cual se requiere la asistencia del especialista.
    - HorasRetraso: es el retraso máximo permitido para la llegada del especialista y depende de la gravedad del paciente.

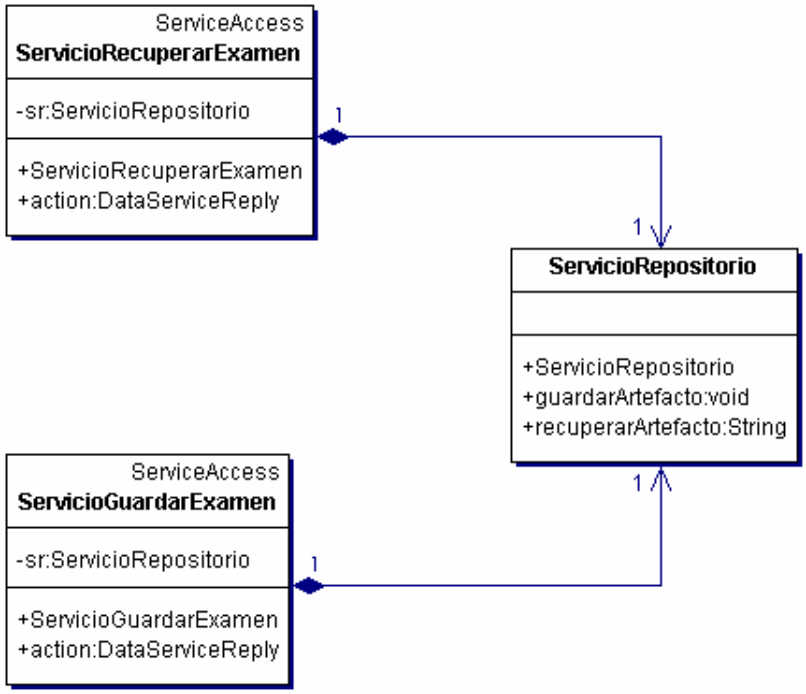
- Métodos:

    - chooseBestBids: Éste método elige las mejores ofertas dentro de todas las ofertas enviadas por los especialistas que aceptaron el ContractNet.

✓ Servicios Guardar Examen y Recuperar Examen: estos servicios permiten almacenar y recuperar los resultados de un examen en un repositorio del mundo estático con el fin de liberar carga y espacio en los dispositivos móviles.

Tanto el servicio de recuperar como el de guardar, se apoyan en la clase ServicioRepositorio que es la encargada de manera directa de escribir el flujo de datos a disco y guardarla en un archivo. La figura 4-3 muestra el diagrama de clases para estos servicios:

FIGURA 4-3. Diagrama de clases para los servicios de guardar y recuperar exámenes



- Métodos de Servicio Repositorio:

    - guardarArtefacto: este método convierte los datos que le llegan a n flujo que es escrito en un archivo de texto en disco. Recibe como parámetros el nombre del usuario, la descripción del examen y los datos del mismo. El archivo que queda almacenado tiene el siguiente formato de nombre:
    - path://NombreUsuario+DescripcionExamen.txt

    - recuperarArtefacto: recupera un artefacto almacenado en disco. Recibe como parámetros el nombre de usuario y la descripción del examen.

- Métodos de Servicio Guardad Examen:

    - Action: este método llama a guardarArtefacto de ServicioRepositorio para almacenar el examen.

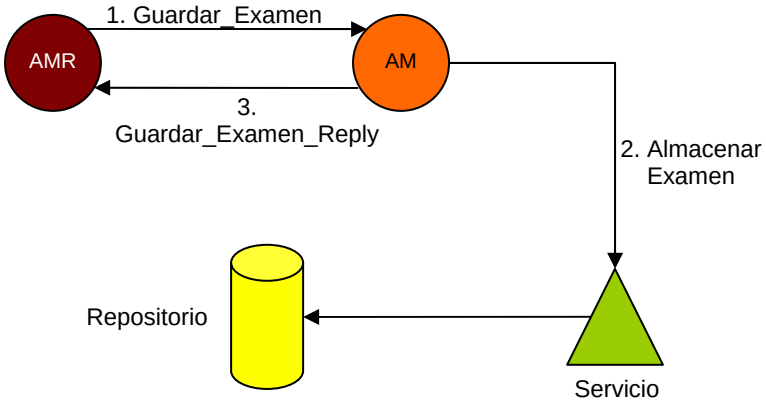
- Métodos de Servicio Recuperar Examen:

    - Action: este método llama a recuperarArtefacto de ServicioRepositosio para recuperar los datos del examen.

✓ AgenteAMTelemedicina:

- *Estado:*
  E1. Referencia al directorio de páginas amarillas
  E2. Referencia al servicio de recuperar y almacenar artefactos.
- *Metas:*
  M1. Almacenar examen.
  M2. Recuperar examen.
- *Entradas Sensoriales:*
  - *Asincrónicas*
    - No tiene.
  - *Sincrónicas*
    - No tiene.
- *Actuadores:*
  A1. Registrarse en las páginas amarillas y blancas.
- *Comportamientos:*
  B1. Almacenar examen: Cumple con M1
  Mensaje / Evento: Guardar_Examen, solicitud de almacenamiento de un examen en el repositorio. La solicitud está compuesta por un mensaje que indica el usuario que quiere almacenar el recurso, la descripción y los datos del examen para almacenar.
  Acción: El AgenteAMTelemedicina hace una invocación al servicio encargado de almacenar el examen en el repositorio, enviándole los datos del examen, la descripción y el id del usuario. Cuando se realiza el almacenamiento, el servicio le devuelve un reply confirmando el éxito o fracaso de la operación.
  Respuesta: Se envía un reply hacia el AMR que envió la solicitud. En la figura 4-4 se muestra en diagrama del comportamiento:

FIGURA 4-4. Comportamiento almacenar examen.

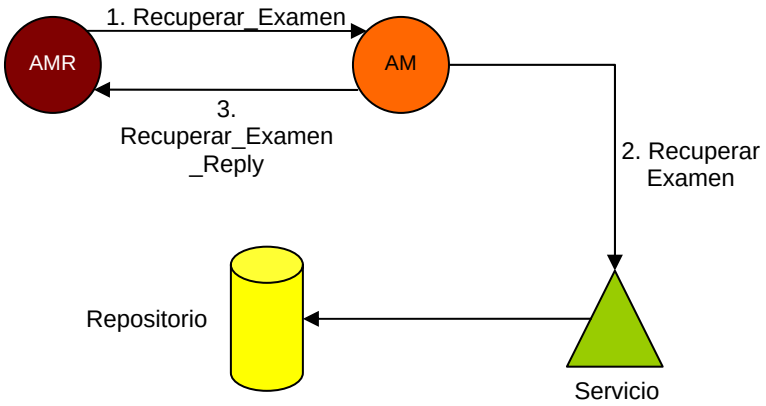


B2. Recuperar examen: Cumple con M2
Mensaje / Evento: Recuperar_Examen, solicitud para recuperar un examen que se encuentra en el repositorio. La solicitud está compuesta por un mensaje que indica el usuario que solicita el examen y la descripción.
Acción: El AgenteAMTelemedicina verifica que el usuario que autorizado para recuperar el examen, y la descripción sean los mismos. Si esto se

cumple, el AgenteAMTelemedicina hace una invocación al servicio encargado de recuperar el examen en el repositorio, enviándole el nombre de usuario y la descripción.
Respuesta: Se devuelve el artefacto al AgenteManejadorRecursos que lo solicitó. Si surgió alguna excepción porque el usuario que solicitaba el artefacto no estaba autorizado para recuperarlo, se envía la notificación respectiva con un reply.

En la figura 4-5 se muestra el diagrama del comportamiento:

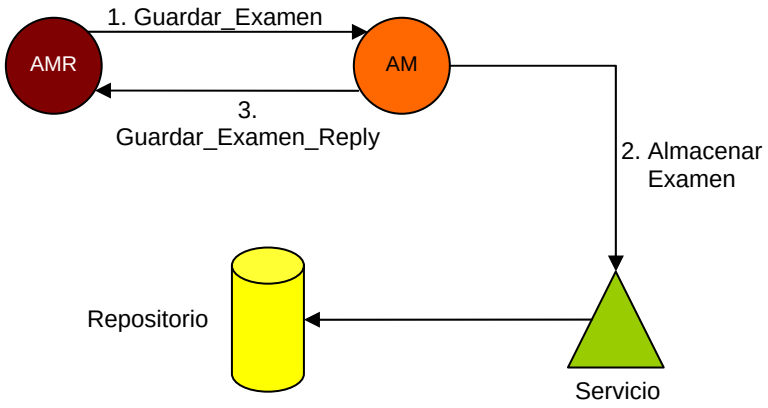
FIGURA 4-5. Comportamiento recuperar examen.



✓ AgenteManejadorRecursos (AMR):

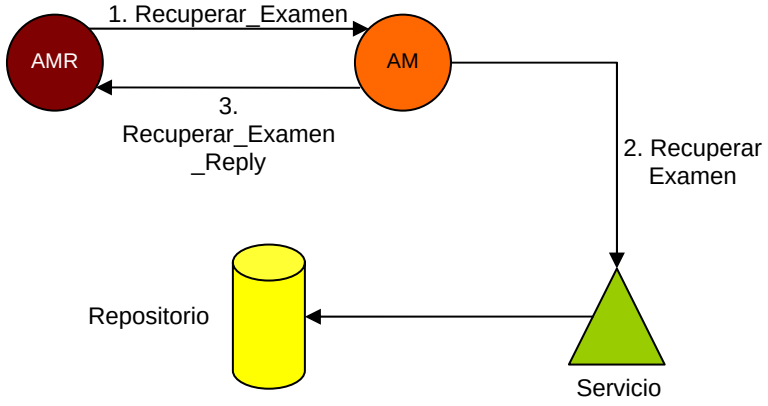
- *Estado*:
  E1. Password de aplicación a la que pertenece.
  E2. Identificador del IA al que pertenece
  E3. Referencia al directorio de páginas amarillas y blancas.
- *Metas*:
  M1. Almacenar un examen en el repositorio.
  M2. Recuperar un examen almacenado previamente en el repositorio.
- *Entradas Sensoriales*:
  - Asincrónicas
    - No hay.
  - Sincrónicas
    S1. Encontrar al agente AM en las páginas blancas y amarillas.
- *Actuadores*:
  A1. Registrarse en las páginas amarillas y blancas.
- *Comportamientos*:
  B1. Almacenar Examen: Cumple con M1.
  Mensaje/Evento: Guardar_Examen, se solicita almacenar un examen en el repositorio. Se devuelve un ACK de éxito o fracaso de la operación.
  Acción: se envía un reply al IATelemedicina y se despliega en pantalla un mensaje de éxito o error al usuario. En la figura 4-6 se ilustra este comportamiento:

FIGURA 4-6. Comportamiento almacenar examen desde el AMR.



B2. Recuperar examen: Cumple con la meta M2.
Mensaje / Evento: Recuperar_Artefacto, se le devuelve el examen solicitado que se encontraba almacenado en el repositorio. Si se presenta una excepción, se devuelve la notificación correspondiente
Acción: se despliega la información del examen en pantalla al usuario utilizando el IATelemedicina del dispositivo. En la figura 4-7 se ilustra este comportamiento:

FIGURA 4-7. Comportamiento recuperar examen desde el AMR.



4.5 PRUEBAS REALIZADAS Y ANÁLISIS DE RESULTADOS

Las pruebas realizadas para validar Ayllu se dividen en tres grupos:

- Pruebas funcionales
- Pruebas de estrés
- Pruebas de escalabilidad

Las pruebas funcionales tienen como objetivo comprobar que Ayllu puede proporcionar todas las características mencionadas en el capítulo implementación de la arquitectura. Las pruebas de estrés validan la capacidad de la plataforma de soportar múltiples peticiones consecutivas. Finalmente, las pruebas de escalabilidad tienen como objetivo establecer cuantos grupos volátiles se pueden crear y soportar de manera concurrente.

Las pruebas fueron realizadas en emulador por dos razones:

- Facilidad para simular múltiples clientes, debido a que con dispositivos reales es prácticamente imposible si se tiene en cuenta que los dispositivos disponibles actualmente en la universidad son pocos.
- Mayor facilidad para transcribir los resultados obtenidos a un archivo de texto y posteriormente a Excel para ser tabulados. En un dispositivo real con J2ME, la manera adecuada de almacenar archivos es a través de un RecordStore que es simplemente una pequeña base de datos que se almacena en el dispositivo. Debido a su formato, es imposible obtener información de manera legible para el usuario, ni fácil de tabular.

Este tipo de pruebas trae consigo limitaciones, que ya fueron mencionadas en MAD [BA2004], y aplican de la misma manera para las pruebas de Ayllu.

4.5.1 Especificación del Protocolo de Pruebas. Con el fin de realizar correctamente cada una de las pruebas, se debe asegurar la correcta ejecución de los experimentos para poder obtener los datos más fiables y así poder validar correctamente la arquitectura Ayllu.

Por esta razón se debe definir un protocolo de pruebas, que proporcionará la rigurosidad de los experimentos. De acuerdo a planeación de las pruebas que se van a realizar, explicadas anteriormente, se define que este protocolo aplicará para las pruebas de estrés y escalabilidad, debido a que por su naturaleza presentan las condiciones necesarias para medir las variables que más adelante se introducirán. Para las pruebas funcionales, este protocolo de pruebas no aplica, debido a que esta clase de prueba va a probar que la arquitectura Ayllu funcione mediante el caso de estudio.

El protocolo de pruebas propuesto contendrá la definición de las variables independientes y dependientes para las pruebas, el dominio de cada una de las variables y el ámbito de las pruebas.

✓ *Variables independientes.* Para las pruebas se deben definir las variables independientes, que son aquellas que influirán el resultado a medir y sobre las cuales se tiene control. A continuación se definen las siguientes variables independientes que serán utilizadas en cada uno de los experimentos:

- Número de peticiones: esta variable aplica para la prueba de estrés y medirá la cantidad de peticiones que se lanzan a la aplicación, para saber como responde la arquitectura Ayllu.

- Número de clientes: esta variable aplica para la prueba de escalabilidad y se utilizará para observar el comportamiento de la arquitectura Ayllu, cuando hay uno o varios clientes lanzando peticiones.

- Número de agentes: esta variable aplica para la prueba de escalabilidad y se utilizará para variar el número de agentes que se crearán por grupos volátil y así poder observar el comportamiento de la arquitectura Ayllu.

✓ *Variables dependientes.* En cada una de las pruebas, también se deben definir las variables dependientes que son aquellas que se van a medir, no se tiene control sobre estas y están determinadas por las variables independientes utilizadas en los experimentos. A continuación se definen las variables dependientes que serán utilizadas en cada uno de los experimentos:

- Tiempo de respuesta: esta variable es definida para la prueba de estrés. Con este tiempo se desean obtener los tiempos de respuesta de la arquitectura Ayllu, cuando ésta recibe múltiples peticiones.

- Tiempo de creación: esta variable es definida para la prueba de escalabilidad. Con este tiempo se desean obtener los tiempos de creación de los grupos volátiles de la arquitectura Ayllu, cuando ésta recibe múltiples peticiones variando el número de clientes y de agentes para los grupos volátiles.

✓ *Rango de las variables.* En la tabla 4-3, se muestra los rangos de valores que tomarán cada una de las variables y se resume la información anteriormente descrita.

TABLA 4-3. Rangos de las variables.

Variable	Tipo	Rango	Prueba
Número de peticiones	Independiente	0-1000 (peticiones)	Estrés
Número de clientes	Independiente	1 y 2 (clientes)	Escalabilidad
Número de agentes	Independiente	1,3 y 5 (agentes por grupo volátil)	Escalabilidad
Tiempo de respuesta	Dependiente	n/a	Estrés
Tiempo de creación	Dependiente	n/a	Escalabilidad

✓ *Ámbito de pruebas.* Para la ejecución de las pruebas se contó con 2 máquinas, cada una de las cuales se encargó de realizar algún experimento de las pruebas. En la tabla 4-4 se muestra la configuración de cada máquina utilizada y se lista el tipo de prueba que junto con el experimento que ejecutó.

Aunque las configuraciones de las máquinas eran distintas, se decidió ejecutar las pruebas en éstas para observar los comportamientos de la arquitectura Ayllu. Cada una de las máquinas debe correr el IDE que proporciona el emulador y los ejecutables de la arquitectura Ayllu.

TABLA 4-4. Ámbito de pruebas.

Máquina	Procesador	Memoria RAM	Prueba Ejecutada	Experimento
Máquina 1	3.0 Mhz	1 GB	Estrés	1000 peticiones
			Escalabilidad	1 Cliente, 3 y 5 agentes por grupo volátil
				2 clientes, 3 y 5 agentes por grupo volátil
Maquina 2	2.4Mhz	512MB	Escalabilidad	1 Cliente, 1 agente por grupo volátil
				2 clientes, 1 agente por grupo volátil

4.5.2 Pruebas Funcionales. Empleando el caso de estudio descrito a lo largo de este capítulo, se probaron los requerimientos establecidos para Ayllu, propuestos en el capítulo de *Implementación de la Arquitectura Ayllu*. En la tabla 4-5 se muestra como los casos de uso de Ayllu fueron corroborados con el caso de estudio.

TABLA 4-5. Resultados de las pruebas funcionales.

Framework de Agentes		
Requerimientos/Caso Uso Telemedicina	**Comunicar Agentes**	**Registrar Agentes en los directorios**
Contract Net Medico	✓	✓
Teleconsulta	✓	✓
Ingresar a la comunidad	✓	✓
Cambiar estado de conexión	✓	✓
Consultar especialidades	✓	✓
Consultar personal especifico	✓	✓
Agregar contacto	✓	✓
Consultar exámenes publicados	✓	✓
Almacenar exámenes	✓	✓
Recuperar exámenes	✓	✓
Activar Mensajería	✓	✓
Desactivar Mensajería	✓	✓
Enviar Mensaje	✓	✓
Consultar Mensajes	✓	✓
Eliminar Mensajes	✓	✓
Leer Mensajes	✓	✓

Capa de Recursos				
Requerimientos/Caso Uso Telemedicina	**Consultar Recurso**	**Modificar Recurso**	**Solicitar Recurso**	**Solicitar Servicio**
Consultar exámenes publicados	✓	No aplica	No aplica	No aplica

Recuperar Examen	No aplica	No aplica	✓	✓
Almacenar Examen	No aplica	✓	No aplica	No aplica

Capa de Roles		
Requerimientos/Caso Uso Telemedicina	**Definir rol**	**Asociar responsabilidad**
Contract Net Medico	✓	✓
Teleconsulta	✓	✓

Capa de Comunidad		
Requerimientos/Caso Uso Telemedicina	**Contener las sesiones de usuario**	**Conformar Grupos Volátiles**
Ingresar a la comunidad	✓	No aplica
Contract Net Medico	✓	✓
Teleconsulta	✓	✓
Almacenar Recursos	✓	✓
Recuperar Recursos	✓	✓

Capa de Cooperación			
Requerimientos/Caso Uso Telemedicina	**Proporcionar Servicios de Cooperación**	**Proporcionar Servicio de Directorios**	**Proporcionar Servicios de Comunicación**
Contract Net Medico	✓	✓	✓
Teleconsulta	✓	✓	✓
Ingresar a la comunidad	No aplica	✓	No aplica
Cambiar estado de conexión	✓	✓	No aplica
Consultar especialidades	No aplica	✓	No aplica
Consultar personal especifico	No aplica	✓	No aplica
Agregar contacto	No aplica	✓	✓
Consultar exámenes publicados	No aplica	✓	No aplica
Activar Mensajería	✓	✓	✓
Desactivar Mensajería	✓	✓	✓
Enviar Mensaje	✓	✓	✓
Consultar Mensajes	No aplica	✓	No aplica
Eliminar Mensajes	✓	✓	
Leer Mensajes	✓	✓	✓

Capa de Aplicación				
Requerimientos/Caso Uso Telemedicina	**Iniciar aplicación cooperativa**	**Hacer deploy de aplicación cooperativa**	**Comunicar aplicación cooperativa**	**Destruir aplicación cooperativa**
Contract Net Medico	✓	✓	✓	✓
Teleconsulta	✓	✓	✓	✓
Ingresar a la comunidad	✓	✓	✓	✓
Cambiar estado de conexión	✓	✓	✓	✓
Consultar	✓	✓	✓	✓

especialidades				
Consultar personal especifico	✓	✓	✓	✓
Agregar contacto	✓	✓	✓	✓
Consultar exámenes publicados	✓	✓	✓	✓
Almacenar exámenes	✓	✓	✓	✓
Recuperar exámenes	✓	✓	✓	✓
Activar Mensajería	✓	✓	✓	✓
Desactivar Mensajería	✓	✓	✓	✓
Enviar Mensaje	✓	✓	✓	✓
Consultar Mensajes	No aplica	✓	✓	✓
Eliminar Mensajes	✓	✓	✓	✓
Leer Mensajes	✓	✓	✓	✓

CAP		
Requerimientos/Caso Uso Telemedicina	**Permitir acceso a la comunidad**	**Reportar estado de conexión**
Contract Net Medico	✓	✓
Teleconsulta	✓	✓
Ingresar a la comunidad	✓	✓
Cambiar estado de conexión	✓	✓
Consultar especialidades	✓	✓
Consultar personal especifico	✓	✓
Agregar contacto	✓	✓
Consultar exámenes publicados	✓	✓
Almacenar exámenes	✓	✓
Recuperar exámenes	✓	✓
Activar Mensajería	✓	✓
Desactivar Mensajería	✓	✓
Enviar Mensaje	✓	✓
Consultar Mensajes	No aplica	✓
Eliminar Mensajes	✓	✓
Leer Mensajes	✓	✓

Como se aprecia en la tabla 4-5, se cumplen todos los requerimientos planteados para Ayllu de forma satisfactoria.

4.5.3 Pruebas de estrés. Las pruebas de estrés tienen como objetivo medir el tiempo de respuesta de la arquitectura Ayllu ante las peticiones que se le lancen y así validarán la capacidad de la arquitectura para atender múltiples peticiones.

Para realizar esta prueba se ejecutó el siguiente experimento. Se simularon mil peticiones para consultar listas de contacto de un usuario de manera consecutiva. Este proceso involucraba comunicación entre el mundo estático y el dinámico para poder acceder

desde el dispositivo al agente administrador, el cual consultaba en los directorios y retornaba la respuesta al dispositivo. En el desarrollo de esta prueba no se tiene en consideración el proceso de autenticación para ingresar a la comunidad.

Se establecieron cuatro puntos de control para medir los tiempos:

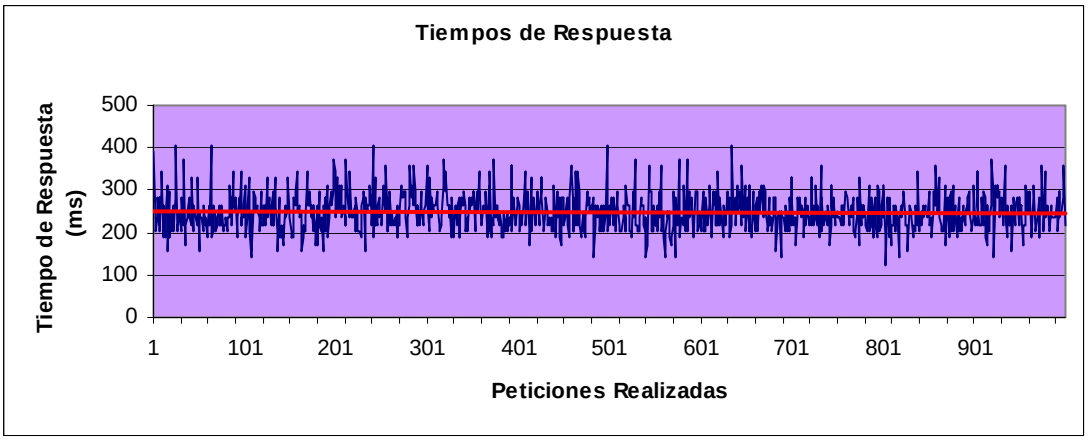
1) Al enviar la petición desde el dispositivo.
2) Al recibir la petición en el mundo estático (agente administrador).
3) Al enviar la respuesta a la solicitud desde el mundo estático (luego de consultar los directorios).
4) Al recibir la respuesta en el dispositivo (en el handler correspondiente del IA).

Con estos tiempos es posible calcular otros de interés, que se enumeran a continuación:

1) Tiempo entre el envío del mundo dinámico al mundo estático
2) Tiempo de procesamiento
3) Tiempo entre el envío de la respuesta del mundo estático al dinámico
4) Tiempo de respuesta

El experimento se realizó varias veces encontrando que los resultados arrojaron una respuesta prácticamente determinística, es decir, los resultados eran casi siempre los mismos, con variaciones mínimas. Por esta razón a continuación se incluyen con detalle las diferentes gráficas que se obtuvieron de uno de los experimentos:

FIGURA 4-7. Tiempos de respuesta.



En la figura 4-7 se observa un comportamiento variable de los tiempos de respuesta, encontrando un máximo de 407ms y un mínimo de 125ms con una media general de 235ms y varianza de 2048,18. La línea de tendencia de la gráfica muestra que a través del tiempo, el tiempo de respuesta tiende a la media, y en consecuencia se concluye que la arquitectura soporta un gran número de peticiones simultáneas a través del tiempo, manteniendo muy buen desempeño.

La figura 4-8 muestra una tendencia similar a la gráfica anterior, pero en tiempo de procesamiento; el tiempo promedio es de 62ms y su varianza es de 36,54, lo cual permite

concluir que el mecanismo de consulta al directorio es bastante efectivo y no representa mayor carga de tiempo.

En las figuras 4-9 y 4-10 se muestran los resultados obtenidos para el envío de peticiones entre los dos mundos. Se observa que la media de tiempos de envío del mundo dinámico al estático es 110ms, con varianza 1654,84 y la media de envío del mundo estático al dinámico es de 63ms, con varianza 432,22. Esto se debe a qué cuando se envía un mensaje desde el mundo dinámico, debe pasar por una capa de middleware antes de que pueda alcanzar al agente administrador AA (punto de control de tiempo) y poder comenzar el procesamiento; el objetivo de la capa de middleware es aislar la manera en que se realiza el procesamiento de la manera en que es accedido por el dispositivo, permitiendo mayor transparencia y escalabilidad.

De manera opuesta, cuando se envía la respuesta del mundo estático al dinámico, el mensaje simplemente llega al handler correspondiente en el agente interfaz IA, sin pasar por ninguna capa intermedia.

FIGURA 4-8. Tiempo de procesamiento.

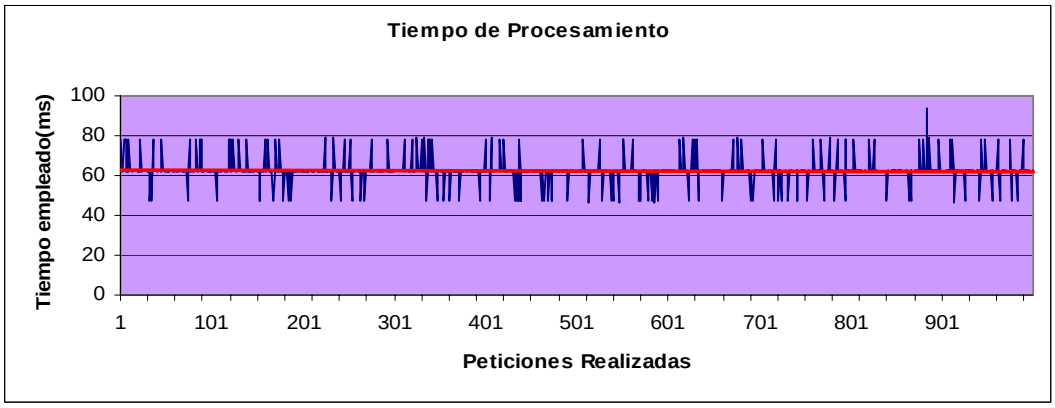


FIGURA 4-9. Tiempo entre dispositivo y mundo estático.

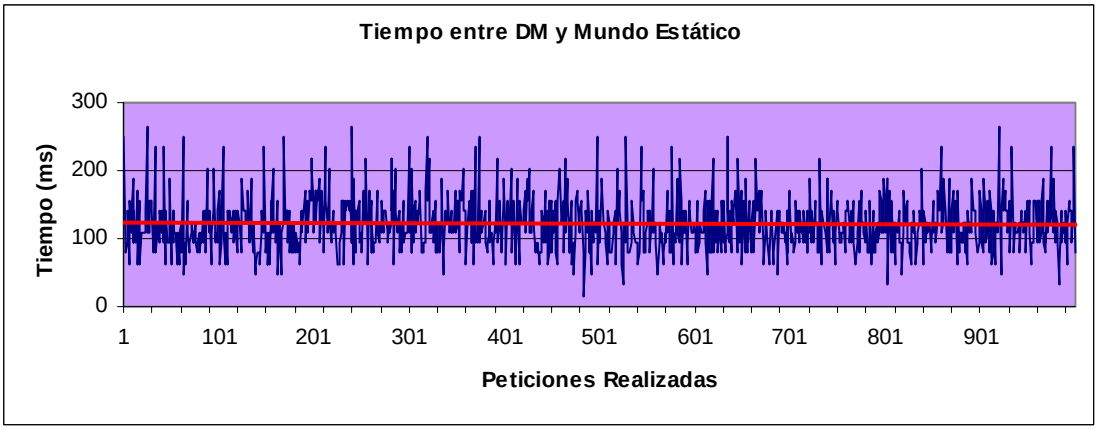
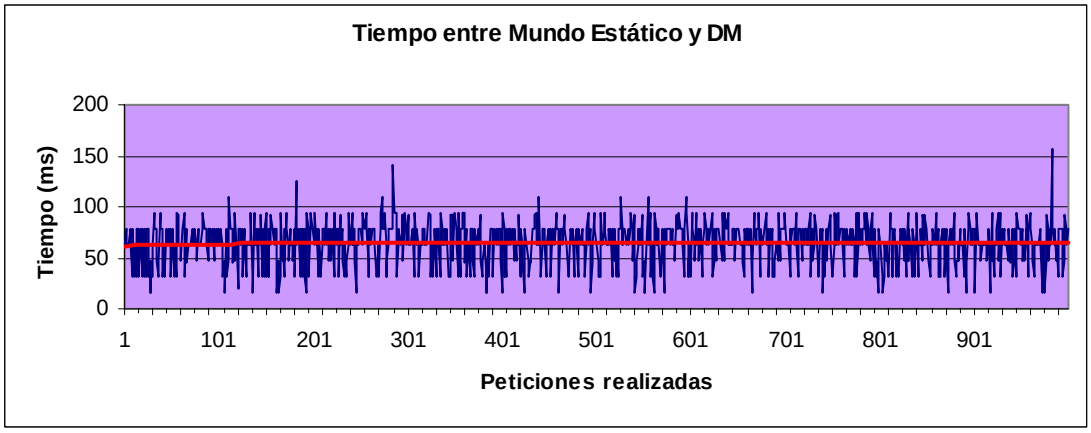


FIGURA 4-10. Tiempo entre mundo estático y DM.



4.5.4 Pruebas de escalabilidad. El objetivo de esta prueba es determinar cuantos grupos volátiles que soportan tareas cooperativas pueden ser creados de manera consecutiva dentro de Ayllu y cuanto tiempo lleva su creación a medida que aumenta el número de peticiones.

Para esta prueba se realizarán dos experimentos, en donde se modificarán los valores de las variables independientes identificadas. A continuación se explican los experimentos:

- Experimento 1
  - Un cliente que lanzará mil solicitudes de servicio.
  - Para cada grupo volátil que se cree por una solicitud, el número de agentes de comunidad CA que los conformarán tomará los siguientes valores:
    - Primera repetición: 1 agente.
    - Segunda repetición: 3 agentes.
    - Tercera repetición: 5 agentes.

- Experimento 2
  - Dos clientes que lanzarán mil solicitudes de servicio cada uno.
  - Para cada grupo volátil que se cree por solicitud, el número de agentes de comunidad CA que los conformarán tomará los siguientes valores:
    - Primera repetición: 1 agente.
    - Segunda repetición: 3 agentes.
    - Tercera repetición: 5 agentes.

En ambos experimentos se involucran los siguientes pasos:

- Se debe pasar del mundo estático al dinámico cuando se realiza la solicitud.
- Se deben verificar los permisos del usuario para iniciar el servicio; esta labor la realiza el agente fábrica FA.
- Se crea el agente manejador de comunidad del servicio CMA; está labor también la realiza el agente fábrica FA.

- El CMA debe buscar a todos los usuarios que puedan cooperar y pedir su autorización para iniciar el servicio. Esta petición debe viajar del mundo estático al dinámico, pasando por el agente manejador de sesión SM, el agente representante RA y finalmente al agente interfaz IA. La petición sólo llega hasta IA la primera vez que se pregunta.
- El agente interfaz IA responde la solicitud, y la respuesta es enviada de nuevo al mundo estático, quedando almacenada en el RA, y es reenviada al respectivo SM quien a su vez, envía la respuesta definitiva al CMA.
- El CMA crea el agente de comunidad CA del servicio.
- El CA envía un mensaje indicando que está listo para cooperar hasta el IA en el mundo dinámico, pasando sólo por el SM quien actúa como intermediario.

Se establecieron tres puntos de control para medir los tiempos:

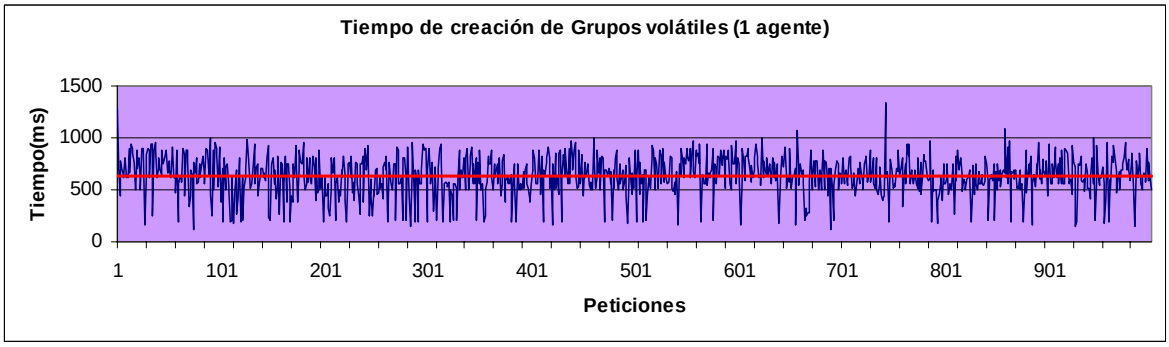
1) Al enviar la solicitud de servicio
2) Al recibir la pregunta de autorización por parte del CMA (este tiempo sólo se toma la primera vez)
3) Al recibir el mensaje del CA indicando que está listo para cooperar.

Para el desarrollo de estos experimentos, no se tiene en cuenta el proceso de autenticación para ingresar a la comunidad.

✓ *Resultados del Experimento 1.* En el experimento, se observó que el promedio de grupos volátiles que soportó Ayllu, cuando un cliente lanzaba múltiples peticiones con 1, 3 y 5 agentes es de 614, para un promedio de 2006 agentes (incluyendo los agentes manejadores de comunidad CMA para cada grupo volátil) creados casi de manera simultánea solamente para los grupos volátiles.

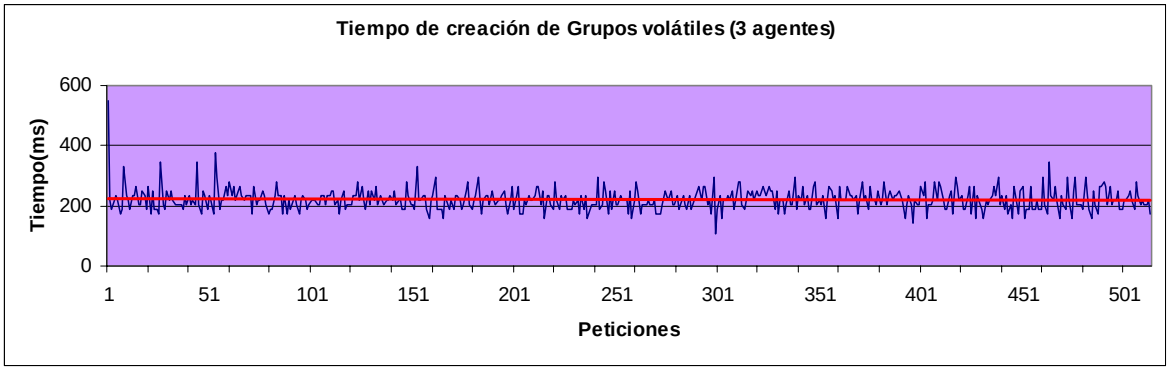
En las figuras 4-11, 4-12 y 4-13 se muestra la tendencia de los tiempos de respuesta vs el número de peticiones realizadas.

FIGURA 4-11. Tiempos para la creación de grupos volátiles con 1 cliente y 1 agente por grupo volátil.



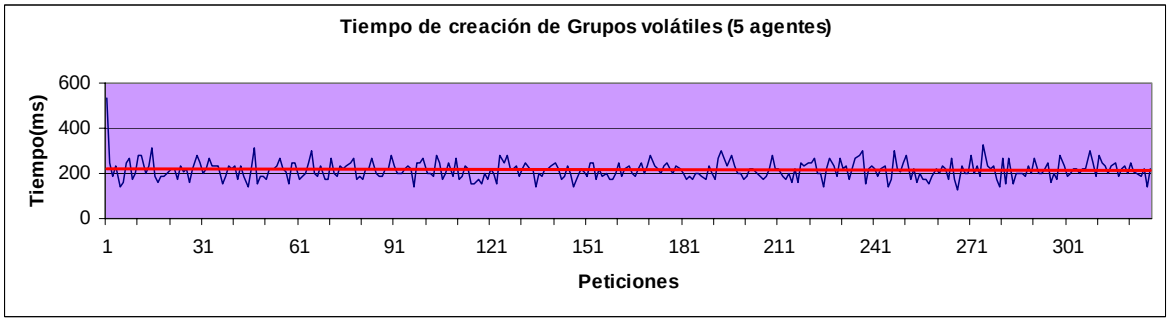
La media de tiempo para este experimento fue de 636ms con un máximo de 1343ms y un mínimo de 125ms y con varianza de 37230,27, soportando 2000 agentes y 1000 grupos volátiles.

FIGURA 4-12. Tiempos para la creación de grupos volátiles con 1 cliente y 3 agentes por grupo volátil.



La media de tiempo para este experimento fue de 221ms con un máximo de 547ms y un mínimo de 109ms y con varianza de 1426,49, soportando 2056 agentes y 514 grupos volátiles.

FIGURA 4-13. Tiempos para la creación de grupos volátiles con 1 cliente y 5 agentes por grupo volátil.



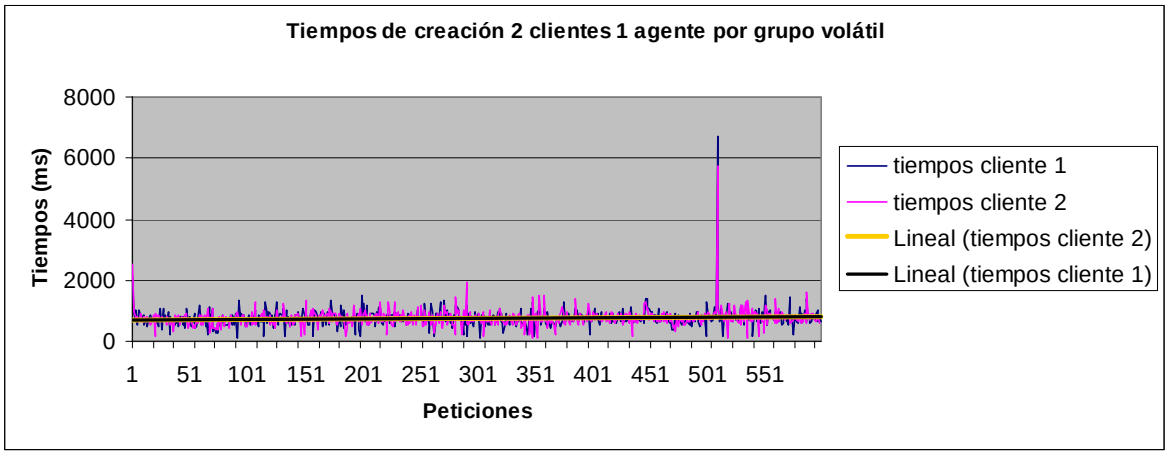
La media de tiempo para este experimento fue de 214ms con un máximo de 531ms y un mínimo de 125ms y con varianza de 1681,78, soportando 1962 agentes y 327 grupos volátiles.

Se observa de manera general, que las líneas de tendencia son constantes, lo que indica que la arquitectura Ayllu soporta un gran número de grupos volátiles sin afectar el desempeño. Como se especificó en la tabla 4-4, los experimentos se realizaron en 2 máquinas distintas, razón por la cual el experimento en que se crean los grupos volátiles con 1 agente, presenta mayor tiempo de procesamiento que los otros dos.

✓ *Resultados del Experimento 2*. En el experimento, se observó que el promedio de grupos volátiles que soportó Ayllu, cuando dos clientes lanzaban múltiples peticiones con 1, 3 y 5 agentes es de 679, para un promedio de 2136 agentes (incluyendo los agentes manejadores de comunidad CMA para cada grupo volátil) creados casi de manera simultánea solamente para los grupos volátiles.

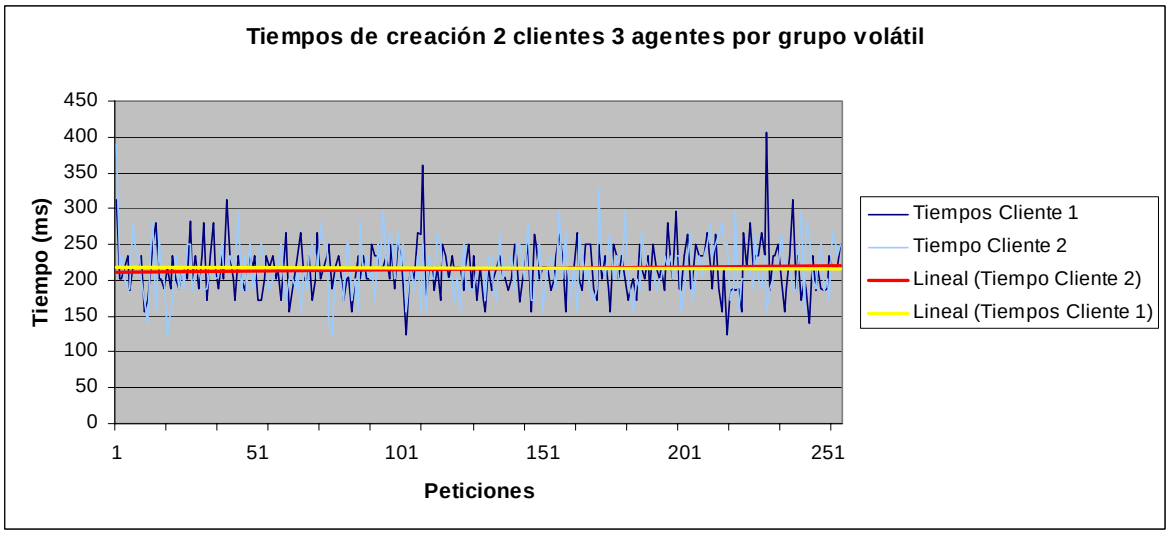
En las figuras 4-14, 4-15 y 4-16 se muestra la tendencia de los tiempos de respuesta vs el número de peticiones realizadas.

FIGURA 4-14. Tiempos para la creación de grupos volátiles con 2 clientes y 1 agente por grupo volátil.



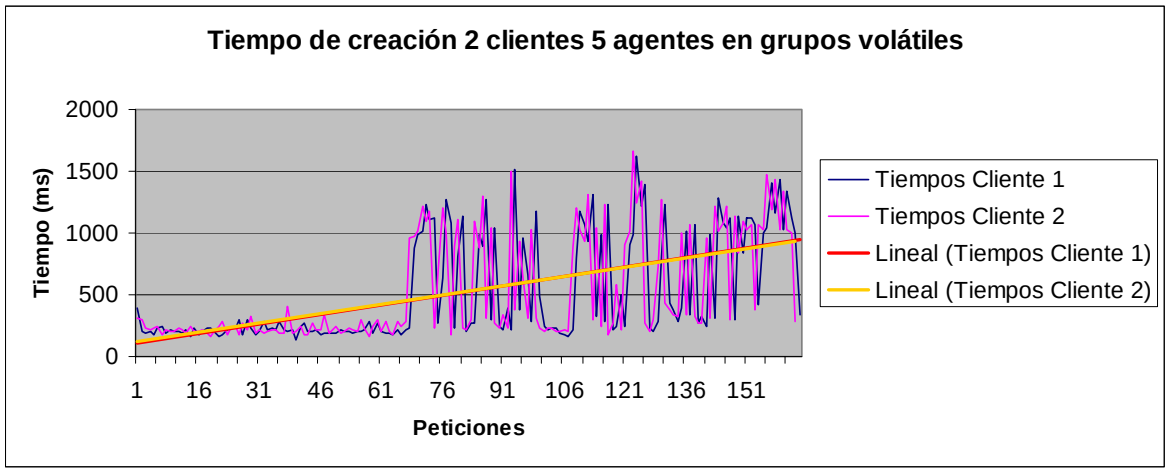
La media de tiempo para este experimento fue de 763ms con un máximo de 6718ms y un mínimo de 93ms, con varianza de 104432,24, soportando 2394 agentes y 1197 grupos volátiles entre los dos clientes.

FIGURA 4-15. Tiempos para la creación de grupos volátiles con 2 clientes y 3 agentes por grupo volátil.



La media de tiempo para este experimento fue de 567ms con un máximo de 9641ms y un mínimo de 15ms, con varianza de 1307,00, soportando 2052 agentes y 513 grupos volátiles entre los dos clientes.

FIGURA 4-16. Tiempos para la creación de grupos volátiles con 2 clientes y 5 agentes por grupo volátil.



La media de tiempo para este experimento fue de 524ms con un máximo de 1656ms y un mínimo de 141ms, con varianza de 174553,12, soportando 1962 agentes y 327 grupos volátiles.

De manera general, se observa que las líneas de tendencia de cada uno de los experimentos son similares para los dos clientes. Una vez más como se especificó en la tabla 4-4, el experimento en donde se crea 1 agente por grupo volátil, tiene más procesamiento que los otros dos, por haber sido ejecutado en una máquina distinta.

En los tres experimentos donde se crean 1, 2 y 5 agentes por cada grupo volátil, claramente se aprecia que el tiempo de creación de un grupo volátil aumentará con el incremento de peticiones a servicios. Este comportamiento se debe a que se congestiona de manera dramática el canal de envío de mensajes desde el mundo estático hasta el dinámico, ocasionando que el emulador deje de responder de manera adecuada.

En las figuras 4-17, 4-18 y 4-19, se muestran las líneas de tendencia de las pruebas, agrupadas por la cantidad de agentes que se creaban en los grupos volátiles y la cantidad de clientes. En las gráficas se puede apreciar la comparación usando 1 solo cliente y con 2 clientes simultáneos.

FIGURA 4-17. Líneas de tendencia utilizando 1 cliente y 2 clientes simultáneos, creando 1 agente por grupo volátil.

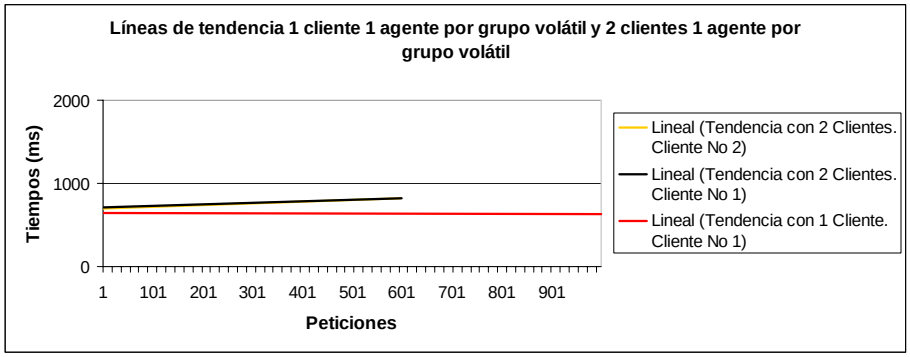


FIGURA 4-18. Líneas de tendencia utilizando 1 cliente y 2 clientes simultáneos, creando 3 agentes por grupo volátil.

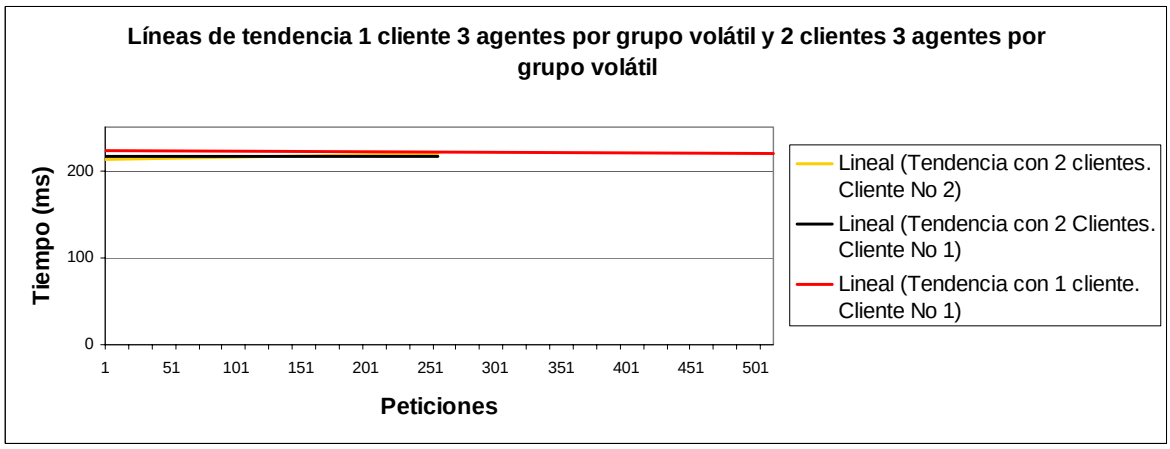
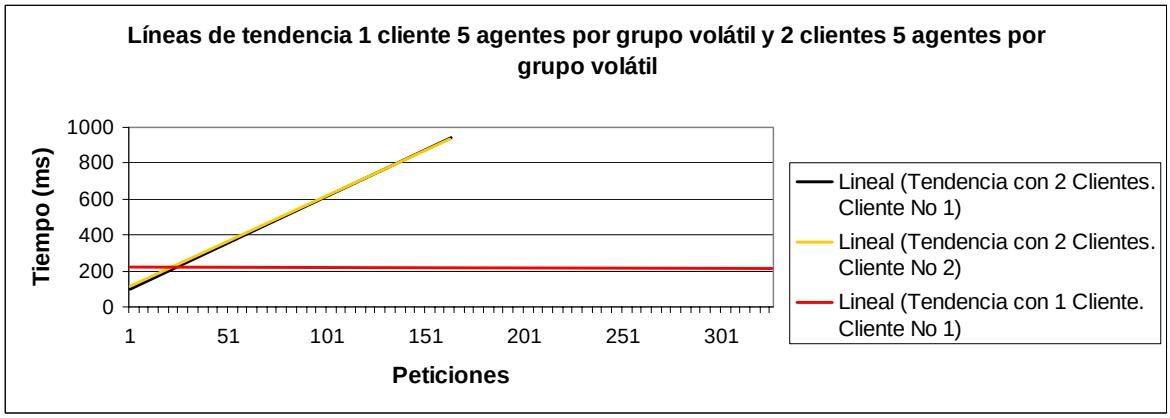


FIGURA 4-19. Líneas de tendencia utilizando 1 cliente y 2 clientes simultáneos, creando 5 agentes por grupo volátil.



En las gráficas anteriores se puede apreciar la tendencia de las gráficas cuando se utilizó un solo cliente representado por la línea roja y cuando se utilizaron dos clientes simultáneos representados por las líneas amarilla y negra. En éstas se puede evidenciar

que el uso de más clientes exige un mayor procesamiento de la plataforma y por consiguiente las tendencias son crecientes.

Como se puede observar en todos los casos las líneas de tendencia, cuando se utilizaban dos clientes simultáneos, son similares, lo que evidencia que la plataforma responde de igual manera a las peticiones lanzadas por éstos. Por otra parte las líneas de tendencia, cuando se utiliza un solo cliente, tienden a ser constantes en el tiempo, representando la manera como la plataforma responde a las peticiones lanzadas.

4.6 CONCLUSIONES SOBRE LA EVALUACIÓN DE AYLLU

Como resultado del proceso de validación de la arquitectura a través del caso de estudio y los resultados de las pruebas realizadas, se puede establecer lo siguiente:

- Continuando con la tendencia de BESA [GE2003] y MAD [BA2004], ambas soporte de Ayllu, se permite heredar de clases abstractas que proporcionan la funcionalidad necesaria para definir nuevos servicios cooperativos, reduciendo de manera significativa los tiempos de desarrollo.
- Los servicios básicos proporcionados por Ayllu también permiten disminuir los tiempos de desarrollo, debido a que en este caso, no es necesario heredar de ninguna clase para poder utilizarlos en cualquier aplicación cooperativa como consecuencia de su naturaleza genérica.
- Ayllu ofrece escalabilidad en servicios cooperativos, soportando gran cantidad de grupos volátiles.
- En las pruebas de escalabilidad, se observó que BESA no es capaz de soportar mas de 2400 agentes aproximadamente, generando en repetidas ocasiones un error en la plataforma, que no permitía continuar creando grupos volátiles.
- Las características de autenticación, autorización y verificación de permisos proporcionan mayor seguridad evitando que los usuarios accedan a servicios o recursos sin tener los permisos y privilegios necesarios.
- Aporta una característica importante para el manejo de las limitaciones de los dispositivos móviles, complementando MAD: el reporte del estado de conexión. Esto permite manejar dentro de una aplicación el ciclo de vida del dispositivo, facilitando la toma de decisiones adecuadas respecto a que acciones tomar y como se deben realizar.
- Ayllu se destaca como un nuevo paradigma para aplicaciones de groupware, extendiendo el concepto de cooperación entre personas a través de la implementación de agentes de software que sirven como mediadores para sus actividades, facilitando la consecución de un objetivo común de manera exitosa.

DISCUSIÓN Y CONCLUSIONES

- El crecimiento de la computación móvil que se ha vendido observando durante los últimos años, abre la posibilidad de diseñar e implementar aplicaciones que se adapten a éstos ambientes computacionales, teniendo en cuenta las necesidades actuales de información que requieren los usuarios consumidores de tecnología. Existen algunas restricciones por el momento, inherentes a los dispositivos móviles, que se esperan mitigar en los próximos años, como la capacidad de almacenamiento, la duración de las baterías, la capacidad de procesamiento y la intermitencia de la señal, para aquellos dispositivos que tengan capacidades de comunicación inalámbrica. Por esta razón al diseñar e implementar aplicaciones de cualquier índole se deben tener presentes estas limitaciones.

- La popularización de dispositivos móviles en los mercados del mundo, revela la tendencia de sus usuarios a buscar el sentido de pertenencia a una comunidad y a fomentar la ubicuidad, gracias a las nuevas tecnologías de comunicación inalámbrica como WIFI, bluetooth y próximamente WIMAX, así como las tecnologías orientadas a los teléfonos móviles como GSM, GPRS y EDGE entre otras.

- Adicionalmente a las tendencias expuestas anteriormente, se puede incluir una más y es el fomento del trabajo en equipo, sobre todo en los ambientes corporativos, aunque aplicable también en otros campos como el entretenimiento y la educación. La interactividad que se aprecia entre estas tres tendencias se puede resumir en una sola palabra *groupware.* El paradigma del groupware se ha venido desarrollando durante años orientándolo a los dispositivos tradicionales de cómputo como los PC o portátiles, sin embargo, aprovechando las posibilidades que están ofreciendo las nuevas tecnologías este paradigma se ha empezado a orientar también sobre los dispositivos móviles.

- Aunque existen muchas arquitecturas orientadas a fomentar el groupware, éstas son muy estáticas y vienen orientadas para centralizarse en uno de los aspectos que encierra este paradigma, limitando a los usuarios. La idea es desarrollar nuevas formas para que el groupware sea más orientado a las personas y a lo que realmente éstas necesitan, con el fin de lograr una mejor experiencia y mejores resultados que concluyan en mayor eficiencia y efectividad.

- Es por esta razón que se ha definido un nuevo paradigma para el groupware llamado 5C, que incluye la utilización de agentes de software para soportar la cooperación mediante la colaboración, coordinación y la resolución de conflictos, apoyada en todo momento por la comunicación, que es la base del groupware y permite incluir a las personas como parte de un sistema multi-agente, viéndolas como agentes humanos que se relacionan con otros agentes de software para cooperar en una tarea. De esta manera se logra orientar el groupware más hacia

las personas y se logra hacerlo más inteligente y abierto para las tareas cooperativas.

- De esta forma definimos a Ayllu que significa *Comunidad*, no solamente como una arquitectura orientada al groupware mediada por agentes si no como un paradigma para el groupware, por que aplica el paradigma de las 5C y se apoya en otro concepto interesante llamado grupo volátil, que fue diseñado para que varios procesos se conformen dinámicamente y realicen una labor conjunta. En este contexto se especializó para apoyar una tarea cooperativa, abordando desde un punto de vista diferente la problemática asociada al desarrollo de groupware mediado por agentes de software.

- La arquitectura Ayllu, se enfocó para trabajar el groupware sobre los dispositivos móviles, aunque es tan abierta que se podría aplicar en combinación con dispositivos de cómputo tradicionales proporcionado mayor escalabilidad. Para que Ayllu pudiera contrarrestar los problemas inherentes a los dispositivos móviles, fue desarrollada sobre la arquitectura MAD, que propuso una solución interesante para la comunicación entre dispositivos móviles ubicados en un mundo dinámico y dispositivos fijos ubicados en un mundo estático. Adicionalmente Ayllu aportó un mecanismo de identificación del estado de conexión de los agentes creados en el mundo dinámico, para asegurar que los mensajes sean enviados a un usuario que se encuentre utilizando una aplicación sobre Ayllu.

- Ayllu también se soportó en MAD, en el framework de agentes BESA y su versión para dispositivos móviles BESA-CE, para el desarrollo de los agentes que proporcionarían el desarrollo del paradigma de las 5C y participarían en los grupos volátiles resolviendo una tarea cooperativa del groupware. Con esto se demostró que el diseño de estas tres arquitecturas es altamente funcional y permiten ser extendidas según se requiera.

- Conceptualmente Ayllu se diseño como una arquitectura por capas que se relacionan entre sí, pero cada una es especializada en una tarea específica, que permitió transparencia entre los distintos componentes y en el momento de la implementación se vio reflejado por que en el desarrollo se pudieron crear todos los componentes y agentes rápida y eficazmente.

- Para del diseño de aplicaciones cooperativas sobre Ayllu, se debe emplear una metodología basada en diagramas de interacción, que no incluyen solo a los agentes de software sino también a las personas como parte integral de la aplicación, permitiendo identificar de una manera sencilla los servicios y componentes involucrados, facilitando la transición entre las etapas de diseño e implementación.

- La estructura jerárquica de Ayllu, permite disminuir notablemente los tiempos de desarrollo, debido a la reutilización de código que presta funcionalidades básicas para las tareas cooperativas, liberando al programador de la carga que significa implementar este tipo de componentes.

- Ayllu proporciona una serie de servicios cooperativos genéricos que pueden ser utilizados en cualquier aplicación. Adicionalmente existe la posibilidad de crear nuevos servicios cooperativos, teniendo en cuenta la metodología de desarrollo de aplicaciones sobre Ayllu y que pueden ser fácilmente integrables a la arquitectura. Existe dentro de Ayllu un administrador que permite el manejo de estos servicios y realiza interacciones sobre el directorio de personas, permitiendo la configuración de éstos servicios a los usuarios.

- Ayllu demostró en las pruebas realizadas, ser confiable y robusta, capaz de soportar muchos agentes y grupos volátiles realizando tareas cooperativas concurrentemente.

- La arquitectura Ayllu realiza un gran aporte a los trabajos realizados por el grupo de investigación SIDRe, al ampliar los desarrollos orientados al uso de los agentes de software, como solución tecnológica innovadora para el software. Todo lo anterior expone el resultado del trabajo para la integración con otros desarrollos como MAD y BESA y por consiguiente la profundización en la construcción del conocimiento. Al igual que demostró ser versátil al proporcionar mediante el caso de estudio un impacto social y tecnológico en el área de la medicina y que aunque solo se pretendía enfocarse en un pequeño problema, puede convertirse en la base para proyectos posteriores en distintas áreas donde exista la necesidad de groupware y cooperación entre personas.

TRABAJOS FUTUROS

Esta es la primera versión de la arquitectura Ayllu, en el futuro se pueden agregar muchas más funcionalidades como proporcionar más inteligencia al agente representante RA, que en el momento utiliza una tabla hash para recordar los servicios a los que el usuario le han pedido cooperar. Otro aspecto que el en futuro se puede mejorar es proporcionar un mecanismo de encripción entre los mensajes que pasan del mundo estático al dinámico y viceversa. También se podría proporcionar en el deployment de los servicios, configuración para cada usuario, es decir que las aplicaciones, en este caso orientadas a los dispositivos móviles, se puedan auto configurar para mostrar los servicios activos para un usuario en un momento específico.

En el presente Ayllu, además de complementar los trabajos realizados por el grupo SIDRe de la Pontificia Universidad Javeriana, es la base para un proyecto de grado que se está comenzando a desarrollar referente la cooperación de personas al interior de organizaciones.

También se podría utilizar el paradigma propuesto por Ayllu para el desarrollo de una metodología para diseñar servicios cooperativos al interior de organizaciones, basados o no en agentes de software.

BIBLIOGRAFÍA Y FUENTES DE INFORMACIÓN

1. [AP2000] AGEN, Persi G. *El Ayllu* en http://www.telser.com.pe/assen/ayllua.htm. Último acceso Marzo 17 de 2005 11:20 a.m.

2. [BA2004] BOTERO Alejandro, GIRALDO Hugo, MOYANO Alexandra. *MAD: Mobility Supported by Agent-Based Devices* en http://pegasus.javeriana.edu.co/~mad. Último acceso 19 de Noviembre de 2004.

3. [BD2001] BUSZKO Dominic, LEE Wei – Hsing (Dan), HELAL Abdelsalam (Sumi). *Decentralized Ad – Hoc Groupware API and Framework for Mobile Collaboration*. Motorola iDEN Group, departamento CISE, Universidad de la Florida. 2001

4. [BF2004A] BELLIFEMINE, Fabio. *Using JADE* en http://jade.tilab.com/. Último acceso Noviembre 17 de 2004 8:38 p.m.

5. [BF2004B] BELLIFEMINE Fabio, CAIRE G, POGGI A, RIMASSA G. *JADE: A White Paper* en http://exp.telecomitalialab.com/upload/articoli/V03N03Art01.pdf. Último acceso Noviembre 17 de 2004 9:22 p.m.

6. [BR1995] BALTER Roland, BEN ATALLAH Slim, KANAWATI Rushed. *Architecture for Synchronous Groupware Application Development*. HCI, Tokyo, Japan. 1995.

7. [CG1994] COULOURIS George, DOLLIMORE Jean, KINDBERG Tim. *The Amoeba multicast Protocol* en http://www.cdk3.net/coordination/Ed2/AmoebaMulticast.pdf. Último Acceso: 9 de Noviembre de 2004. 11:22 a.m.

8. [CG2001] COULOURIS George, DOLLIMORE Jean, KINDBERG Tim. *Sistemas Distribuidos: Análisis y Diseño*. Tercera Edición. Pearson Education S.A, Madrid: 2001, Capítulo 11.

9. [DS2002] DUSTDAR Scharam, GALL Harald. *Architectural Concerns in Distributed and Mobile Collaborative Systems*. Viena University of Technology, 2002.

10. [FD2002] FARIAS Damian, PÉREZ Adith. *Cooperación en Sistemas Multi-agente, un caso de estudio: RoboCup*. Pontificia Universidad Javeriana, Ingeniería de Sistemas, 2002 en http://pegasus.javeriana.edu.co/~robocup/. Último Acceso: 25 de enero de 2005. 10:42 a.m.

11. [FJ1999] FERBER, Jacques. Multiagent *Systems: An Introduction to Distributed Artificial Intelligence.* Addison – Wesley. 1999, Capítulos 2,6 y 7.

12. [FP2004] FIPA. *FIPA Agent Management Specification*. FIPA TC Agent Management, SC00023k, 2004.

13. [GE2003] GONZÁLEZ Enrique, BUSTACARA César, ÁVILA Jamir. *BESA: Behavior-oriented Event-driven Social-based Agent Framework*, Pontificia Universidad Javeriana, Ingeniería de Sistemas, 2003.

14. [GM2003] GEROSA, Marco Aurelio. *Towards an Engineering Approach for Groupware Development*. Software Engineering Laboratory (LES) – Computer Science Department Catholic University of Rio de Janerio (PUC-Rio) 2003.

15. [JF2004] JADE. *Java Agent Development Framework* en http://jade.tilab.com/. Último acceso Noviembre 17 de 2004 8:38 p.m.

16. [KE2002] KIRDA Engin, FENKAM Pascal, REIF Gerald , GALL Hararld. *A service architecture for mobile Teamwork*. Technical University of Vienna, 2002.

17. [KG2002] KORTUEM, Gerb. *Proem: A Middleware Platform for Peer-to-Peer Computing*. Mobile Computing and Communications Review*,* Volume 6, Number 4, 2002.

18. [KP2002] KALRA Puja. *J2ME ReferentePoint Suite*. SkillSoft Press, 2002.

19. [KS2004] KURKOVSKY Stan, BHAGYAVATI, RAY Arris. *A Collaborative Problem-Solving Framework for Mobile Devices*. Columbus State University. 2004.

20. [LE2001] LÁZARO, Eduardo. *Telemedicina y Bioingeniería.* SimiConsultora, 2001 en http://www.simiconsultora.com.ar/publicaciones/telemedicina.asp. Último acceso: Mayo 20 de 2005 12:00 m.

21. [LR2000] LITIU Radu, PRAKASH Atul. *Developing Adaptive Groupware Applications Using a Mobile Component Framework.* Department of Electrical Engineering and Computer Science, University of Michigan. 2000.

22. [MC2001] MOSQUERA, Celestino. *Análisis y estudio experimental de herramientas basadas en agentes*. Universidad de Coruña, Facultad de Informática, 2001.

23. [MI2002] MARSIC, Ivan. *An Architecture for Heterogeneous Groupware Applications.* The State University of New Jersey, USA.2002.

24. [MS2004] MICROSOFT CORPORATION. *Microsoft Developers Network* en http://www.msdn.com. Último Acceso: Noviembre 18 de 2004 3:00 p.m.

25. [RJ2000] ROTH Jörg , UNGER Claus. *DreamTeam a plataform for synchronous collaborative applications.* Department of Computer Science - University of Hagen, 2000.

26. [RJ2000A] ROTH, Jörg. *A Taxonomy for Synchronous Groupware Architectures*. Department of Computer Science - University of Hagen, 2000.

27. [RJ2001] ROTH Jörg, Unger Claus, “Using Handheld Devices in Synchronous Collaborative Scenarios”. Department of Computer Science - University of Hagen, 2001.

28. [RM2003] RODRÍQUEZ Marcela, FAVELA Jesús. *A Framework for Supporting Autonomous Agents in Ubiquitous Computing Enviroments*. Departamento de Ciencias de la Computación, CICESE, Ensenada, Mexico. 2003.

29. [RO2003] RATSIMOR Olga, FININ Tim, JOSHI Anupan, YELENA Yesha, *eNcentive: A Framework for Intelligent Marketing in Mobile Peer-To-Peer Enviroments*. Departamento of Computer Science and Electrical Engineering, University of Maryland Baltimore Country, 2003.

30. [SM2004] SUN MICROSYSTEMS. *Java 2 Platform, Mircro Edition (J2ME)* en http://java.sun.com/j2me. Último Acceso: Noviembre 17 de 2004, 2:00 p.m.

31. [WW1999] WANG Weicong, DOROHONCEAU Bogdan, MARSIC Ivan. *Design of the DISCIPLE Synchronous Collaboration Framework*. The State University of New jersey, USA. 1999.

32. [YS2002] YAU Stephen S, KARIM Fariaz, WANG Yu, WANG Bin, GUPTA Sandeep K.S. *Reconfigurable Context Sensitive midleware for Pervasive Computing.* Departamento de ciencias computacionales e ingeniería Universidad de Arizona. 2002