A mi esposa, compañía y apoyo incondicional. A Brachel nueva eternidad familiar.

OBJETOS DE SOFTWARE MÓVILES PARA LA CONSULTA DE INFORMACIÓN EN BASES DE DATOS TIPO P.O.T.
(prototipo)

CRISMAN MARTINEZ BARRERA.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSE DE CALDAS
FACULTAD DE INGENIERIA
MAESTRIA EN TELEINFORMATICA

Bogotá D.C.
Agosto de 2003

OBJETOS DE SOFTWARE MÓVILES PARA LA CONSULTA DE INFORMACIÓN EN BASES DE DATOS TIPO P.O.T.
(Prototipo)

CRISMAN MARTINEZ BARRERA.

POYECTO DE TESIS PARA OPTAR AL TITULO
DE MAGISTER EN TELEINFORMATICA

Director: Ing. Msc. HENRY ALBERTO DIOSA
Docente Facultad de Ingeniería

UNIVERSIDAD DISTRITAL
FRANCISCO JOSE DE CALDAS
FACULTAD DE INGENIERIA
MAESTRIA EN TELEINFORMATICA

Bogotá D.C.
Agosto de 2003

Nota de Aceptación .

_____
_____
_____
_____
_____

_____
Presidente Jurado

_____
Jurado Calificador

_____
Jurado Calificador

Bogotá D.C., Agosto de 2003

AGRADECIMIENTOS

A la Universidad Distrital y a sus directivos por la calidad humana y científica con la cual dirigen la institución y la oportunidad que le ofrecen a los colombianos de acceder a la educación. A la directora de maestría Dra. Edith Aparicio, a los profesores del programa y en especial al ingeniero Henry Alberto Diosa, director de este proyecto; sus días de descanso los ha dedicado una y otra vez a la corrección de los documentos que hacen parte de este proyecto que ahora es realidad.

A los funcionarios de los municipios de tabio y tenjo que en un comienzo interactuaron para definir los requerimientos a modelar del plan de ordenamiento territorial.

A mi familia: Luz Helena esposa amada. A Brachel nuevo ser de luz. A mi mamá, mujer emprendedora y consejera eterna. A mi papá lugar de descanso en momentos de cansancio. A mis hermanos Jackeline, Maria Antonia, Johnson, Carolina, Peregrino e Hilda. A mis amigos: Oscar Leonardo, Liliana Mira, Juan Herrera., Elena Sandoval, a los tíos (as), primos(as) y a todos que de una u otra forma han contribuido al desarrollo de este proyecto.

A la Universidad Central por su apoyo incondicional, a la naturaleza, a lo visible e invisible gracias.

CONTENIDO

Pág.

INDICE DE FIGURAS

Pág.

INDICE DE TABLAS

Pág.

INDICE DE ESQUEMAS

LISTA DE ANEXOS

ANEXO A. Ley 388 De 1997, Por La Cual Se Modifica La Ley 9ª De 1989, Y La Ley 3a. De 1991 Y Se Dictan Otras Disposiciones. Disponible en CD que acompaña el libro de tesis.

ANEXO B. LEY 99 DEL 22 DE DICIEMBRE DE 1993. Por la cual se crea el MINISTERIO DEL MEDIO AMBIENTE, se reordena el Sector Público encargado de la gestión y conservación del medio ambiente y los recursos naturales renovables, se organiza el Sistema Nacional Ambiental -SINA- y se dictan otras disposiciones. Disponible en CD que acompaña el libro de tesis.

ANEXO C. Principales Clases que Componen a *Aglets.*

ANEXO D. Diccionario de datos, diccionario de clases, SQL generado para las consultas tipo P.O.T. y diagramas de secuencia.

ANEXO E. Diagrama de paquetes para los módulos gráficos.

ANEXO F. Manual del usuario para el sistema prototipo OSM-CONDABAPOT.

INTRODUCCION

El software y el hardware orientado a las comunicaciones de datos, han desarrollado una infraestructura excepcional para aplicaciones distribuidas utilizando sistemas heterogéneos. Sin embargo, las características específicas de estos sistemas y el tipo de problemas que en ellos se plantean suponen un serio reto para la Ingeniería del Software tradicional, cuyos métodos y herramientas se han visto enfrentados a estos nuevos requisitos.

El componente de software ha adquirido un papel esencial en la ingeniería de los sistemas y servicios de telecomunicaciones. La funcionalidad de los sistemas, su gestión y en buena parte su valor agregado depende de los componentes de software integrados. Los agentes pretenden facilitar la interoperabilidad de sistemas [13], Proveen un bus software (conceptual) independiente de: ***quién*** los diseñó, ***qué*** plataforma lo soporta, ***cómo*** se programaron (lenguaje) y ***dónde*** se ejecutan [14].

El prototipo "Objetos de Software Móviles para la Consulta de Información en Bases de Datos Tipo Plan de Ordenamiento Territorial" (OSM-CONDABAPOT) es una aplicación innovadora que utiliza tecnología de objetos de software móviles, una plataforma de agentes y un sistema de bases de datos tipo P.O.T. en su implementación.

La investigación propone un modelo solución a la consulta de datos a una Base de Datos tipo P.O.T. soportada en una metodología de diseño orientada a objetos basada en UML y en herramientas de implementación ya mencionadas.

Por último, el prototipo orienta la utilización de la tecnología de OSM para obtener información de entidades nacionales (ej: municipios ó localizaciones específicas) referentes al plan de ordenamiento territorial en Colombia.

1. MARCO TEORICO REFERENCIAL

1.1. MARCO REFERENCIAL DE DOMINIO DEL PROBLEMA

1.1.1 AGENTES DE SOFTWARE MÓVILES Y OBJETOS DE SOFTWARE MÓVILES

El objeto de software móvil (OSM) se diferencia del agente de software móvil (ASM) en el grado de autonomía [15], inteligencia y otras características que a continuación se presentan con más detalle.

1.1.1.1 AGENTES DE SOFTWARE MÓVILES

1.1.1.1.1 Concepto

Existen tres disciplinas informáticas fundamentales en el desarrollo y definición de agentes [1].

1. Inteligencia artificial.
2. Programación orientada a objetos y programación concurrente.
3. Diseño de interfaces hombre-máquina.

La definición formal de agente de software es una de las grandes ausentes cuando se trabaja con la tecnología de agentes. Cada autor frecuenta dar su propia definición [22]. A continuación se presentan algunas definiciones, las más usadas:

- Según el Diccionario de la Lengua Española[2]:
  "Agente: Del latín agens, entis, p. a. de agere, hacer.
  1. adj. Que obra o tiene virtud de obrar.
  2. m. Persona o cosa que produce un efecto.
  3. Persona que obra con poder de otro …".

- Según el *Object Management Group* [3]:
  "Un agente es un programa de ordenador que actúa autónomamente en nombre de una persona u organización".

- Según Nils J. Nilson en Inteligencia Artificial[29]:
  “Los agentes disponen de varias modalidades sensoriales para percibir sus mundos y de distintas formas de actuar sobre ellos. Los más complejos tendrán la capacidad de recordar propiedades y de almacenar modelos internos del mundo.”

- Según Russell y Norving en su artículo "*Artificial Intelligence: a Modern Approach*" [4]:
  "Un agente puede verse como aquello que percibe su entorno a través de sensores y que actúa sobre él, mediante efectores".

- Según Hayes-Roth [5]:
  Menciona que un agente inteligente debe tener la capacidad para ejecutar necesariamente tres funciones:

  1. Percibir dinámicamente las condiciones del ambiente
  2. Tomar acción para incidir en las condiciones del ambiente
  3. Tener razones para interpretar las percepciones, solucionar problemas, hacer deducciones y tomar acciones.

- La guía del usuario de *AgentBuilder* agrupa las características de un agente inteligentes en[28]:

  – Se ejecuta autónomamente
  – Tiene la capacidad para comunicarse con otros agentes o usuarios
  – Tiene la capacidad para monitorear su estado en el ambiente de ejecución.

- Según Newell, A. (1988)[30]:
  Argumenta que los agentes de software pueden ser considerados como inteligentes, si poseen las siguientes características:

  – Capaz de extraer conocimiento de gran cantidad de información.
  – Tolerante a fallas inesperadas o entradas erróneas
  – Capaz de usar símbolos y abstracciones
  – Capaz de adaptarse, tiene comportamiento orientado a objetivos
  – Capaz de aprender del ambiente
  – Capaz de operar en tiempo real
  – Capaz de comunicarse usando lenguaje natural

- Según I.B.M. *Aglets* [19]:

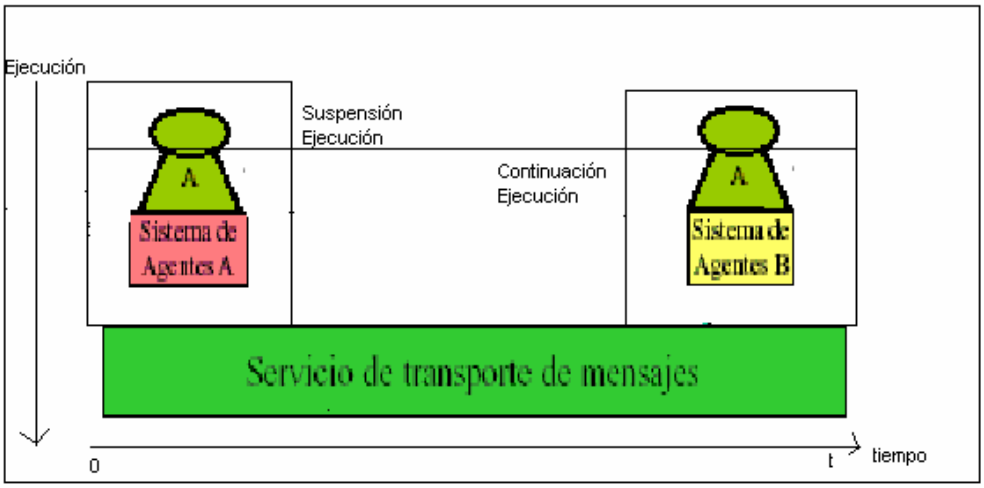
  "Los agentes inteligentes son entidades programadas que llevan a cabo una serie de operaciones en nombre de un usuario o de otro programa, con algún grado de independencia o autonomía, empleando algún conocimiento o representación de los objetivos o deseos del usuario".

- En resumen:

  Los agentes móviles corresponden a un conjunto de instrucciones inteligentes que realizan un objetivo y pueden estar construidos con herramientas de desarrollo estructuradas, orientadas a objetos y/o concurrentes.

  Este agente actúa de manera autónoma, esta programado para que perciba y aprenda de su entorno a través de sensores, razonan sobre lo aprendido, eligen una o varias soluciones para resolver problemas o consultar requerimientos automáticos o humanos, pueden o no viajar sobre la plataforma de comunicaciones de una red LAN o WAN.

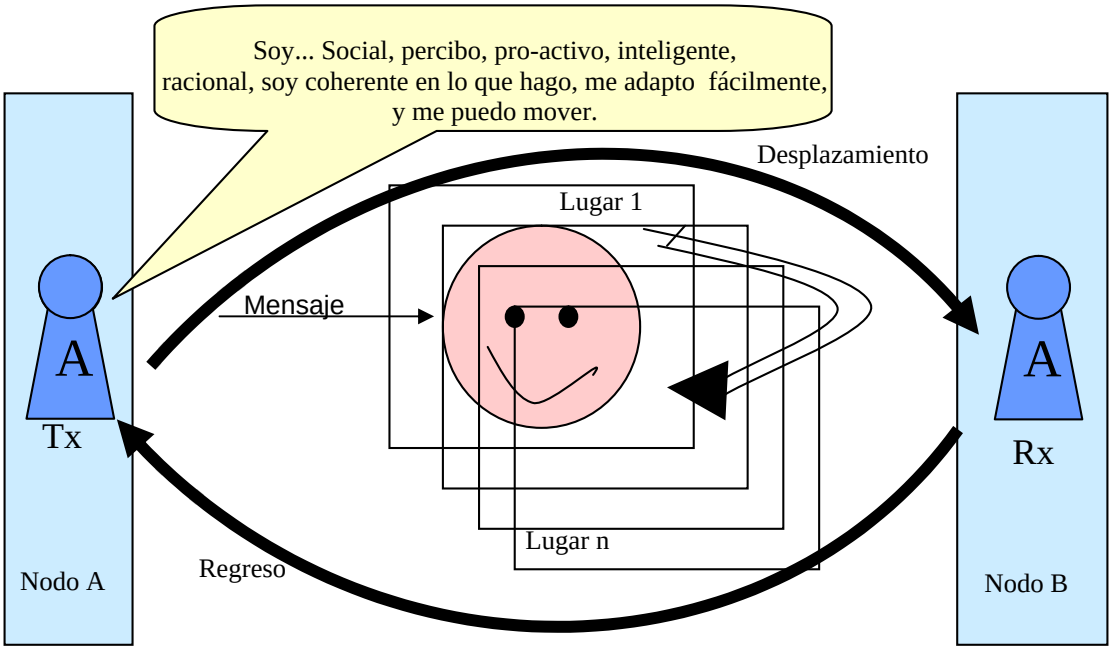
Figura 1. Ciclo de ejecución de un agente móvil



Para el caso de un agente de software móvil que esta sobre la arquitectura de gestión de agentes (ver Figura 1) y para cumplir su objetivo, el agente realiza los siguientes pasos:

1. Inicia la ejecución en el transmisor.
2. Realiza o no una serie de eventos en el transmisor.
3. Suspende la ejecución en el transmisor.
4. Transfiere el código por la red. (Canal de comunicaciones).
5. Despierta en el receptor.
6. Continúa su ejecución (en el receptor) en el punto donde se había suspendido (en el transmisor).
7. Realiza o no una serie de eventos en el receptor.
8. Finaliza ó suspende la ejecución para devolver el requerimiento al transmisor o a otro nodo de la red (ver Figura 2).

Figura 2. Agente de software móvil



1.1.1.1.2 Elementos de un sistema de agentes móviles

Los dos conceptos fundamentales en el modelo de agentes móviles son agente y lugar.

- **El agente**: Un agente de software móvil tiene cinco atributos básicos que debe conservar consigo[16]:

  - Estado: Cuando un agente viaja, transporta su estado junto con él. Debe hacerlo a fin de reanudar su ejecución en el *host* destino. Esto consiste en tomar los valores de los atributos del objeto.

  - Implementación: Como cualquier otro programa, un agente móvil necesita su código para ejecutarse.

  - Interfaz: La interfaz que le permite a otros agentes y sistemas comunicarse con él. Se entiende por interfaz el conjunto de métodos que permiten a otras aplicaciones y agentes comunicarse en "lenguaje de agentes" como por ejemplo con KQML.

  - Identificador: Cada agente tiene un identificador que es único durante su tiempo de vida. Este debe ser único para que pueda emplearse en operaciones a nivel global.

  - Características de desarrollo: cada agente tiene nombre, empresa, propietario, lugar o plataforma.

- **El ambiente de ejecución del agente (Lugar)**: es un contexto en el cual el agente se puede ejecutar y un punto de entrada para un agente visitante.

  El lugar tiene cuatro conceptos importantes:

  - Motor de ejecución *(Engine)*: un lugar no puede por si mismo ejecutar agentes. Para hacerlo, los agentes deben residir dentro de un motor de ejecución, una máquina virtual para lugares y agentes.

  - Recursos: el motor de ejecución y el lugar, proveen acceso controlado a los recursos locales y a los servicios como son las bases de datos, procesadores, memoria, discos, entre otros.

  - Localización: la localización de un agente en ejecución es la combinación de dos características: el nombre del lugar en el cual se ejecuta y la dirección de red donde se encuentra el motor de ejecución. Generalmente la localización se escribe como una dirección IP más el puerto asignado al motor de ejecución.

- Características**:** un lugar debe tener dos características principales: identificación de la persona u organización a quien sirve el lugar y el nombre del desarrollador o autor del mismo.

1.1.1.1.3 Características de un agente

Las características de un sistema de agentes de software móviles según el estándar FIPA *(Foundation for Intelligent Physical Agents)* son[6]:

- **Autónomo**

  La autonomía es una de las características más importantes del concepto de agente.

  El software tradicional suele ejecutarse en entornos interactivos de tal forma que responde a órdenes directas del usuario, es decir, el usuario tiene que decir paso a paso qué es lo que se tiene que hacer.

  La idea de los agentes consiste en crear programas informáticos que tengan una serie de objetivos y posean unos conocimientos del mundo, de tal forma que partiendo de sus conocimientos, los agentes sean capaces de aproximarse lo más posible a sus objetivos sin necesidad de que ningún usuario los guíe paso a paso hacia ellos.

- **Social**

  Cuando se habla de agente no se suele pensar en una única entidad que se ejecuta de forma aislada. Más bien se piensa en sistemas complejos (multi-agente) en la que una serie de agentes colaboran entre sí para llevar a cabo una tarea.

  Este modelo denominado tradicionalmente como “divide y vencerás” presupone que los agentes son capaces de interactuar entre sí y al mismo tiempo con entidades externas al propio sistema como es el caso del usuario.

- **Reactivo**

A pesar de su autonomía, un agente debe ser capaz de percibir estímulos externos tanto para actuar de acuerdo a su entorno cambiante como para poder "conocer" en todo momento cómo es el "mundo" que le rodea. Es decir, que el agente debe tener estímulos y actuar de acuerdo con ellos, estos estímulos afectarán las acciones realizadas por el agente para alcanzar sus objetivos.

- **Pro-Activo**

  Es una de las consecuencias de la autonomía de un agente. Este es capaz de elegir en cada momento cuales son las acciones a llevar a cabo para alcanzar sus objetivos, es decir, un agente no solo actúa en función de los estímulos que recibe desde el exterior, sino que puede realizar acciones como resultado de sus propias decisiones.

- **Inteligente**

  Un agente es inteligente si es racional, coherente y adaptable, en mayor o menor medida. Podemos decir que será más inteligente cuanto más desarrolladas tenga estas características:

  - Racional**:** la racionalidad es una característica propia del ser humano. Un agente se considera racional cuando tiene unos conocimientos de su entorno, unos objetivos deseables y unas reglas que determinan cómo alcanzar los objetivos a partir del conocimiento que se tiene del medio. Esta característica de racionalidad le permite a un agente tomar decisiones sin intervención humana. Se está modelando la racionalidad propia del hombre, eso sí, de momento en problemas muy simples.

  - Coherente**:** el conocimiento que un agente tiene de su mundo se almacena en una base de datos de conocimiento interno al propio agente. Todo este conocimiento debe guardar un alto grado de coherencia para que el comportamiento del mismo sea el esperado.

  - Adaptable**:** el aprendizaje o adaptabilidad es una de las características más complejas que se le puede pedir a una entidad de software inteligente. Un agente aprende cuando es capaz de aumentar su base de conocimiento y su base de reglas a partir de las percepciones que recibe del entorno y a partir de sus comportamientos anteriores a la

hora de resolver problemas. Adaptabilidad es una característica bastante deseable debido a que el entorno de un agente suele ser dinámico en el tiempo y es necesario que se adapte al mismo.

- Móvil**:** esta es una característica opcional que pueden poseer los agentes. Un agente móvil es aquel que se puede mover físicamente por los nodos de una red para poder llevar a cabo sus tareas. El objetivo de la movilidad puede ser una mejor distribución de la carga de procesamiento y de recursos, es decir, una lógica distribuida.

1.1.1.1.4 Clasificación de los agentes

Ubicar a un agente en una clase determinada depende del autor que se esté consultando, la mayoría de ellos coincide en la clasificación por la labor que esté desarrollando.

Juan Luis Posada Yagüe[17] contempla la clasificación basada en las características:

> En general un agente puede clasificarse por la característica que más destaque e influya en su comportamiento e implementación. El agente ideal es aquel que reúne todas las características que definen a un agente inteligente, sin embargo, hoy en día esto aún no es posible y es la aspiración de los investigadores en tecnología de agentes.

Pedro Morales Cuesta[23] dice que los agentes de software pueden ser clasificados de acuerdo a los siguientes criterios:

- Según su comportamiento: reflejo simple, basados en metas.
- Según las Propiedades que satisface: son los agentes móviles adaptativos.
- En función de con quien se comunican: interfaces y tareas de información.
- Según las tareas que llevan a cabo: agentes para filtrado de información, agentes de búsqueda.
- Según su arquitectura: agentes reactivos.

Según Hyacinth Nwana, Lyndon Lee, Nick Jennings los agentes se pueden clasificar en [32]:

- Agentes colaborativos (*Collaborative Agents*).

- Agentes interfaz (*Interface Agents*)
- Agentes móviles (*Mobile Agents*)
- Agentes internet o de Información (*Information/Internet Agents*

- *A*gentes reactivos (*Reactive Agents*)
- Agentes híbridos (*Hybrid Agents*)
- Agentes ideales (*Ideals Agents*).

En el artículo *"Mobile Agents"* [31] se clasifican los agentes móviles en:

- **Sólo movimiento de código** *(just code)***:** se realiza el transporte del código necesario para su ejecución a otro lugar, no se transportan los datos (contexto). No es posible interrumpir la ejecución de un objeto en un computador y continuar su ejecución tras realizarse el movimiento a otro computador. Ejemplos: TCL, JAVA con RMI (*Remote Method Invocation*).

- **Movimiento de código y datos (contexto) (*not just code*):** en este caso puede interrumpirse la ejecución de un objeto en un punto concreto y proseguir con la ejecución de dicho objeto desde el punto en el que se interrumpió, tras realizarse el movimiento en otro computador, Ejemplos: TELESCRIPT, OBLIQ, AGLETS.

De acuerdo a su modelo de comunicación los agentes de software móviles se pueden clasificar en:

- **Comunicación local**: para que el agente se comunique con otro agente en otra máquina esté agente tiene que viajar a otra máquina. Ejemplo: TELESCRIPT.

- **Comunicación por red:** el transporte de un agente implica, además del movimiento de código y datos (contexto), el movimiento de las conexiones (no sólo comunicaciones locales). Ejemplo: OBLIQ.

- **Agente de información**: este tipo de agente se concentra en manipular información distribuida. Son agentes dedicados a viajar por servidores de la web y le facilitan la búsqueda de información al usuario. Utilizan los recursos de buscadores como por ejemplo: *ftp, telnet, mail, archie, gopher, netfind*, entre otros.

En resumen:

Un agente puede clasificarse de acuerdo a la labor que desarrolle y al objetivo que este buscando:

- **Agente colaborativo**: agrupan autonomía y cooperación para ejecutar tareas, tiene la posibilidad de negociar con otros agentes basándose en habilidades de cooperación. Pueden ser usados para resolver problemas distribuidos en entornos multi-agentes.

- **Agente de interface**: son autónomos y utilizan aprendizaje para ejecutar tareas para otros usurarios. El objetivo para esta clase de agentes es el de ser un asistente personal que colabora con los usuarios.

- **Agente móvil**: es un sistema de objetos móviles que pueden viajar al lugar donde se requiere y/o donde las condiciones sean más aconsejables para la ejecución: por tiempo en ejecución, por entorno, y/o por velocidad de procesamiento.

- **Agentes de consulta**: agentes especializados en buscar todo tipo de información. Su objetivo es satisfacer una petición humana o automática.

Cuando se tienen plataformas heterogéneas de agentes móviles, cada agente tendrá su clasificación de acuerdo a la labor desarrollada por él, y como resultado tendremos un sistema multiagente de características de agentes híbridos desarrollado de acuerdo a la clasificación de cada autor.

1.1.1.1.5 Movilidad de los agentes

Para lograr la movilidad de los agentes se puede trabajar con dos soluciones [6].

- **Migración:**

  Este modelo es el más complejo ya que requiere la transferencia de código y datos del agente a la plataforma remota. Cuando un agente va a migrar se suspende, se transfiere por la red y se despierta en el otro

extremo siguiendo su ejecución en el mismo punto donde se había suspendido.

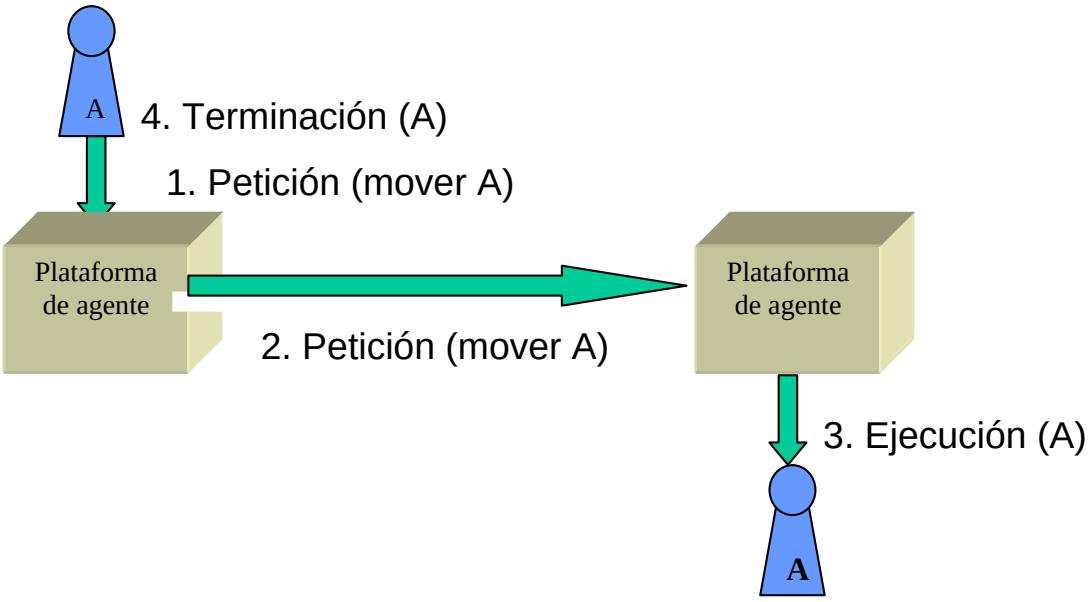
- **Clonación:**

  En este caso se crea una copia del agente en el nodo remoto y éste se encarga de hacer el trabajo solicitado por su versión original en la máquina remota. En este caso no hay una transferencia efectiva de código y datos, sino solo un conjunto de órdenes que el clon debe realizar.

Modelo de movilidad Simple

El AMS (*Agent Management System*) es el responsable de realizar toda la gestión necesaria. El agente solicita a su AMS la transferencia y éste se ocupa de llevarla a cabo (ver Figura 3).

Figura 3. Modelo de Movilidad Simple



Fuente: Organización Internacional FIPA[6]

El protocolo utilizado en este tipo de movilidad es el protocolo de movilidad simple: el agente delega en un protocolo de alto nivel su operación de movilidad a una determinada plataforma (ver Figura 3). El protocolo soporta una única acción (*move*). En este caso, la plataforma en la que se ejecuta el agente tendrá que implementar dicho protocolo necesario para la operación de migración[17].

Las ventajas del protocolo simple son:

1. Reduce la complejidad del desarrollo del agente debido a que la movilidad la proporciona la plataforma.

2. Orientado hacia sistemas existentes de agentes móviles (MASIF) y fácil para la implementación en plataformas ya existentes mediante el lenguaje de comunicación de agentes de FIPA (ACL).

3. El número de interacciones remotas es reducido.

Modelo de movilidad complejo

El agente es el responsable de realizar toda la operatividad que le permita desplazarse a la plataforma remota (protocolo de migración complejo).

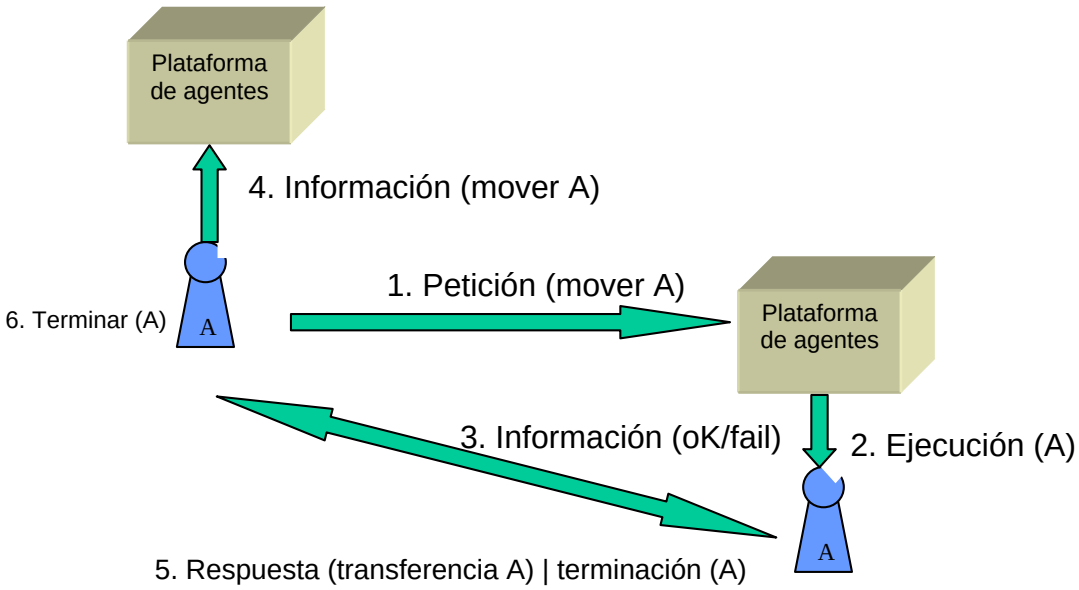
En éste, el agente dirige el protocolo necesario para su operación de movilidad y no delega la responsabilidad en la plataforma (ver Figura 4). Primero el agente mueve su código y estado a la plataforma destino, una vez el nuevo agente ha sido creado con éxito entonces transfiere su identidad y autoridad.

Este protocolo también permite que el agente informe a la plataforma en la cual fue creado *(Home Agent Platform: HAP*) y a las otras plataformas de agentes (A*gents Platforms: AP's*) que se ha movido a una nueva localización.

Las ventajas del protocolo complejo son:

1. Reduce la implementación de la plataforma debido a que son los agentes quienes proporcionan la movilidad.
2. Capacidad del agente para controlar la operación de movilidad.
3. Es la forma más segura de movilidad.

Figura 4. Modelo de Movilidad Complejo.



Fuente: Hayes Roth en architecture for adaptive intelligent systems [5]

Cuando un agente inicie con la acción *move* una operación de movilidad como migración, clonación o invocación, el agente tendrá que indicar el protocolo de movilidad que ha de usarse en la operación, con esta información, el AMS determinará los pasos a realizar para completar la operación, lo cual puede requerir el uso de otras acciones como *transfer*[17].

1.1.1.1.6 Ciclo de vida de un agente de software móvil

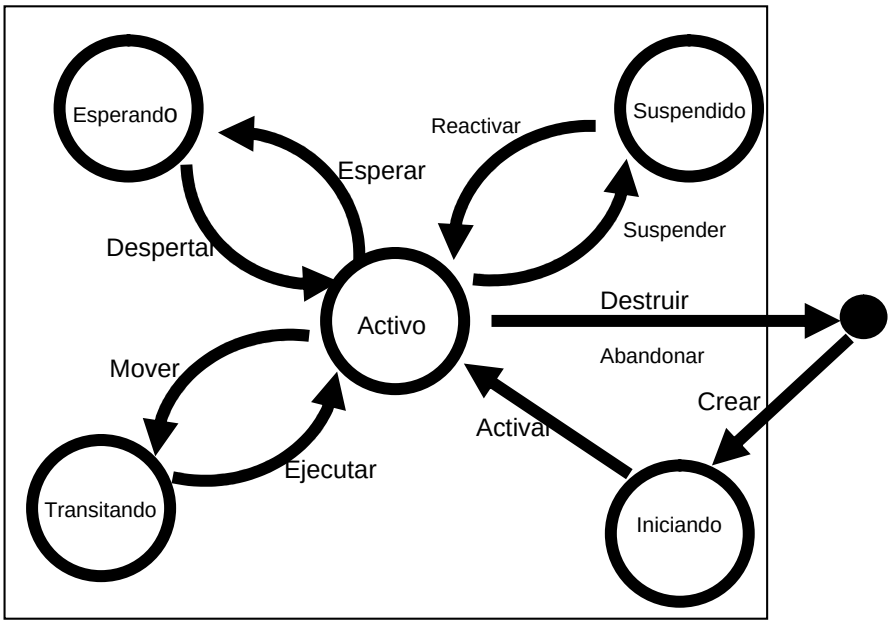
Como se puede ver en la Figura 5, el ciclo de vida representa cada uno de los estados en el cual puede estar un agente de software móvil en el sistema.

Los estados de un agente son:

- **Inicial**: la inicialización de un agente móvil toma lugar en un contexto. Al nuevo agente se le asigna un identificador, se inserta en el contexto, se inicializa y entonces empieza la ejecución. Pasa a este estado con el evento crear.

- **Suspendido**: es la desactivación de un agente. Es la habilidad para detener su ejecución temporalmente y almacenar su estado en memoria secundaria. La activación del agente lo restaura al mismo contexto, queda en estado activo. Pasa a este estado con el evento suspender.

Figura 5. Ciclo de vida de un Agente de software móvil



Fuente: Organización internacional FIPA[6]

- **Activo**: invocación o activación de un nuevo agente una vez inicializado. Pasa a este estado con los eventos activar, reactivar, despertar y ejecutar.

- **Transitando**: es el despacho de un agente de un contexto a otro, se removerá de su contexto origen a otro destino, donde continuará su ejecución y queda en estado activo. Pasa a este estado con el evento mover.

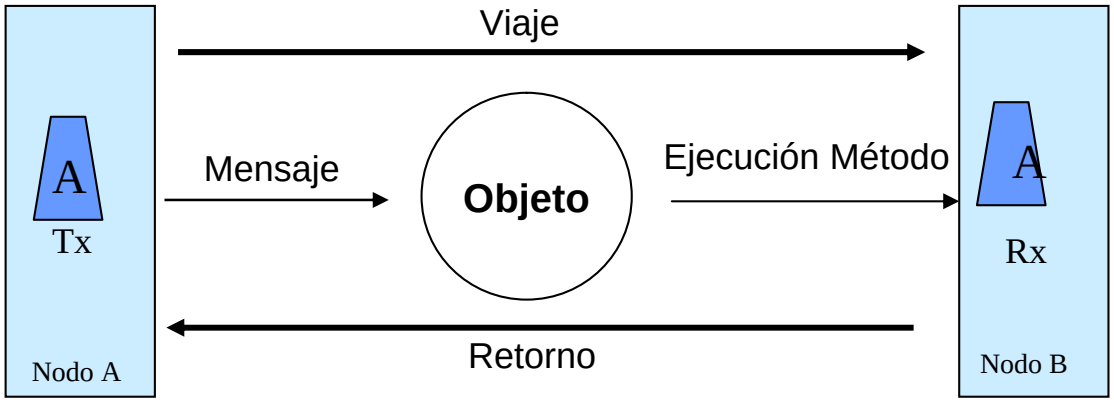
- **Esperando**: pone a un agente en estado de espera, esta acción solo puede inicializarla el agente. Pasa a este estado con el evento esperar.

1.1.1.2 OBJETOS DE SOFTWARE MÓVILES

1.1.1.2.1 *Concepto*

Corresponden a objetos que se mueven entre dos o más aplicaciones locales o remotas en donde el estado y el código se transfieren. Se pueden desplazar por la red si las aplicaciones son distribuidas y normalmente cumple tareas específicas para las que fue programado. Los objetos no tienen inteligencia y no actúan de manera autónoma.

Figura 6. Objeto de Software Móvil



Como se muestra en la Figura 6, el objeto de software móvil, de acuerdo a ciertos parámetros recogidos de su entorno (mensajes) viaja a un sitio remoto a realizar tareas, para luego regresar a un sitio específico

concluyendo así su objetivo.

Sin embargo, “un objeto de software móvil puede ser autónomo en algunos casos: la experiencia ha mostrado que crear objetos que se mueven por si mismos en una red no es generalmente requerido en muchas aplicaciones. En vez de esto, los objetos móviles son parte de una aplicación, como cualquier objeto y son movidos como parte de una comunicación de invocación remota normal”[15].

1.1.1.3 DIFERENCIACIÓN ENTRE AGENTES DE SOFTWARE MÓVILES Y OBJETOS DE SOFTWARE MÓVILES

Los objetos no se ejecutan autónomamente, a diferencia de los agentes que pueden o no ejecutar operaciones autónomamente, dependiendo del tipo de agente, los agentes son capaces de monitorear el estado en el ambiente de ejecución, los objetos no. Los agentes son capaces de usar símbolos y abstracciones como el lenguaje KQML y son capaces de extraer conocimiento de gran cantidad de información, los objetos no.

Dependiendo de la aplicación que se desee construir tanto los objetos como los agentes pueden o no necesitar mensajes. Los objetos no son tolerantes a fallas en los sistemas, mientras que los agentes pueden ser diseñados para que soporten caídas del sistema. Los agentes pueden intercambiar mensajes de conocimiento, los objetos no.

Para permitir que los objetos ejecuten o no la acciones, se requiere más programación, mientras que los agentes no. El agente puede reaccionar de manera diferente al mismo mensaje, mientras el objeto requiere otro mensaje para que actúe de forma diferente, el agente permite programarse con inteligencia dependiendo de la aplicación, se puede diseñar para que actualice conocimiento (aprenda), tenga creencias, deseos e intenciones y tenga autonomía.

Los objetos pueden o no tener su propio hilo de ejecución, los agentes lo tienen por la plataforma de gestión de agentes. Cuando se desea programar un objeto con estas propiedades se deben heredar las propiedades de la clase superior de agentes, teniendo así que ejecutarse en una plataforma de gestión de agentes. Los agentes pueden iniciar acciones sin intervención de otras entidades, ya que el entorno de la plataforma de agentes lo monitorea y lo controla.

Tabla 1. Características de Objetos y Agentes de software Móviles

Característica	Objetos	Agentes
Se ejecutan autónomamente	No	Si/No
Se comunican con otros elementos o usuarios	Si/No	Si/No
Monitorea el estado en su ambiente de ejecución	No	Si/No
Es capaz de usar símbolos y abstracciones	No	Si
Capaz de extraer conocimiento de gran cantidad de información	No	Si/No
Capacidad para adaptarse, comportamiento orientado a objetivos	No	Si/No
Requieren mensajes	Si/No	Si/No
Actúa en respuesta a un mensaje	Si/No	Si/No
Tolerante a fallas inesperadas o entradas erróneas	No	Si/No
Intercambio de mensajes a nivel de conocimiento	No	Si/No
Deciden si ejecutan o no la acción asociada al mensaje	No	Si/No
Pueden reaccionar de diferente manera al mismo mensaje	Si/No	Si/No
El mensaje puede ser solo para actualizar su conocimiento	No	Si/No
Posee estado: creencias, deseos, intenciones	No	Si/No
Grado de autonomía: Tiene control sobre el comportamiento	No	Si/No
Dimensión social: las interacciones se dan dentro de un contexto, trabajan en equipo, coordinados por un elemento central	No	Si
Tiene su propio hilo de ejecución	Si/No	Si
Inician acciones sin intervención de otras entidades	No	Si
Ofrece una serie de servicios que se pueden invocar	Si	Si
Capaz para aprender del ambiente	No	Si
Perciben el entorno y responden de forma oportuna a cambios que ocurren en él	No	Si/No
Están dirigidos por un objetivo	Si/No	Si
Capaz para comunicarse usando lenguaje natural	No	Si

Tanto los objetos y agentes ofrecen servicios que pueden ser invocados (métodos). Los agentes aprenden del entorno, y responden de forma oportuna a cambios que ocurren en él, debido a la plataforma de agentes, ya que ella debe estar presente para gestionar los agentes sea cual fuera la plataforma de sistema operativo.

Por último, los objetos son programados cuando no requieren trabajo de inteligencia, mientras que los agentes se programan cuando se requieren soluciones de inteligencia artificial como: racionalidad, coherencia y adaptabilidad.

1.1.2 ORGANISMOS DE ESTANDARIZACIÓN DE AGENTES MOVILES Y ARQUITECTURAS

El objetivo más importante en la tecnología de agentes móviles es la interoperabilidad entre diferentes sistemas. Para facilitar dicha interoperabilidad es necesario especificar de una forma estándar acciones como la transferencia de agentes, transferencia de clases y operaciones de control de los agentes. Cuando los sistemas de agentes origen y destino son similares (mismo lenguaje de implementación), la estandarización de estas acciones asegura la interoperabilidad. Sin embargo, si los sistemas son diferentes sólo una mínima interoperabilidad puede conseguirse, ya que el código que se transfiriera a un sistema con un lenguaje diferente no podría ejecutarse [17].

1.1.2.1 ORGANISMOS DE ESTANDARIZACIÓN DE AGENTES MOVILES

Entre las organizaciones de estandarización internacional se encuentran las siguientes:

1.1.2.1.1 *OMG*

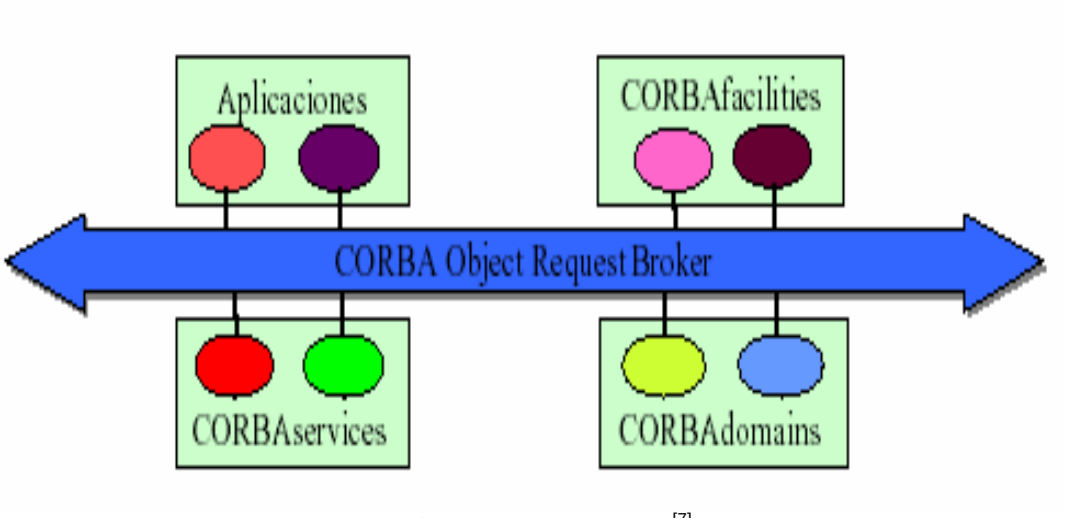
El OMG (*Object Management Group)* es un organismo fundado en 1989 y formado por 800 miembros incluyendo empresas de desarrollo de software, empresas de ventas de software y usuarios.

El OMG ha desarrollado el estándar MASIF (Mobile Agent System Interoperabilities Facility) que es una especificación proporcionada para la interoperabilidad entre sistemas de agentes escritos en el mismo lenguaje, pero desarrollados por diferentes empresas. El OMG ha desarrollado también el estándar CORBA ante la necesidad de ínteroperar diferentes productos de software y hardware.

La arquitectura CORBA (Common Object Request Broker Architecture) es una especificación estándar de un modelo o protocolo de comunicación entre objetos distribuidos, está basada en un gestor de peticiones a objetos comunes y permite la interoperabilidad entre aplicaciones remotas en un entorno completamente heterogéneo (ver Figura 7). La arquitectura CORBA soporta agentes como objetos CORBA que tienen la posibilidad de moverse,

ejecutarse autónoma y asincrónicamente en sistemas de ejecución seguros. La implementación se realiza mediante referencias a los objetos implicados utilizando sus interfaces.

Figura 7. Arquitectura CORBA



Fuente: Organización internacional OMG[7]

CORBA ORB (*Object Request Broker*) es un bus de comunicación que gestiona las peticiones a los objetos, el ORB es independiente del lenguaje empleado, el protocolo que permite comunicar ORB´s es el IIOP *(Internet inter ORB protocol)*.

Los servicios de CORBA son (ver Figura 8):

- Servicio de nombrado de agentes: este servicio asocia un nombre con un objeto *(name binding),* cada nombre es único. Las operaciones utilizan el servicio de nombres para encontrar los objetos a través de sus nombres y para declarar o asignar el nombre del los objetos que comparten.

  Resolver un nombre significa encontrar el objeto asociado con el nombre en un contexto dato. Enlazar un nombre es crear una asociación nombre/objeto para un contexto particular. Cada componente cuenta con dos atributos: un identificador del objeto y una descripción de él.

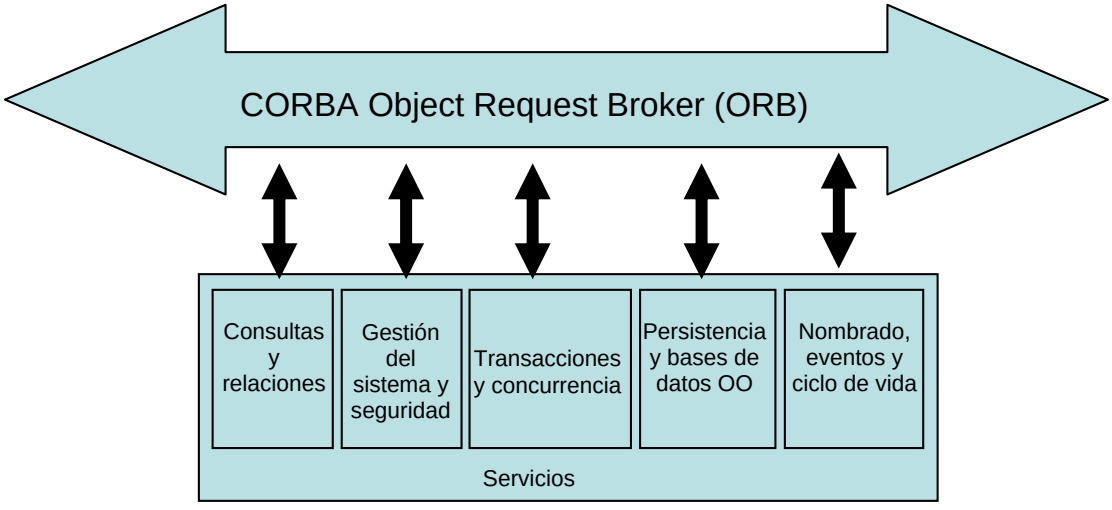
- El servicio de negociación (t*rader):* permite a un objeto localizar el servicio de acuerdo al que más satisfaga sus necesidades. Los clientes se ponen en contacto con el servicio de negociación para averiguar los servicios listados y preguntar por un servicio de acuerdo a su tipo. Este servicio encontrará el más apropiado basado en el contexto del servicio requerido y la oferta de sus proveedores.

- Servicio del ciclo de vida: se definen en CORBA *life cycle service* para crear, borrar, copiar, y mover objetos CORBA. Estas operaciones pueden soportar asociaciones entre grupos de objetos relacionados por referencia y contenido.

  Para crear un nuevo objeto, un cliente debe encontrar una fábrica de objetos, ejecutar una petición de creación y recibir de regreso una referencia a un objeto.

- Servicio de Eventos: permite a los objetos registrar y eliminar dinámicamente eventos. Mantiene la comunicación entre objetos con los eventos *push,* y *pull*. Define quien produce los eventos y quien los va a consumir.

Figura 8. Servicios CORBA



Fuente: Arquitectura de seguridad integral para sistemas de agentes móviles [34]

- Servicio de transacciones de objetos: los ORB´s proveen un entorno para correr los componentes de misión crítica. La transacción es el contrato entre el cliente y el servidor, es la unidad fundamental de recuperación, consistencia y concurrencia en un sistema de objetos distribuidos. Este servicio define cuando inicia y cuando termina una transacción.

- *Servicio* de control de concurrencia: provee las interfaces para fijar y liberar bloques que permitan a clientes múltiples coordinar su acceso a recursos compartidos. El servicio no define un recurso, soporta modos de operación con o sin transacciones y provee los bloqueos a favor de una transacción o a favor de un cliente sin transacciones.

- Servicio de persistencia de objetos: permite a los objetos persistir mas allá de la aplicación que lo creo o el cliente que lo usa, permite ser recuperado cuando se necesite.

- Servicio de consulta: permite encontrar objetos cuyos atributos coincidan con el criterio de búsqueda que se específico en una consulta. También permite manipular los objetos y navegar en una colección de ellos.

- Servicio de colección: permite manipular los objetos como un grupo; se aplican operaciones de grupo y proporciona una forma uniforme de crear y manipular las colecciones comunes.

- Servicio de relación: permite a los componentes y objetos que no saben nada de otro estar relacionados. No se cambian objetos, crea relaciones dinámicas entre objetos inmutables.

- Servicio de externalización: proporciona mecanismos estándar para almacenar el estado de un objeto mediante una secuencia de *byte, object serialization* en java. Define un formato de datos que permite intercambiar flujos entre distintas redes.

- Servicio de seguridad: CORBA permite nombrado, autenticación de agentes, autenticación de los clientes para la creación remota de agentes, autenticación entre sistema de agentes, acceso a los resultados de autenticación, credenciales, delegación y permite definir políticas de seguridad.

- Servicio de sincronización: permite a un ORB mantener los relojes de distintas máquinas sincronizados y dar la noción de tiempo para poder ejecutar eventos que tendrán que ocurrir en un orden específico en un sistema de objetos distribuidos.

CORBA provee facilidades (colecciones de jerarquías de clases bien definidas) que proveen servicios directamente útiles a los objetos de los negocios. Existen las facilidades horizontales que proveen IDLs y marcos en un segmento del mercado y las facilidades verticales encargadas de gestionar los objetos del negocio.

Las facilidades de CORBA son:

- La facilidad común de interfaz de usuario (horizontal): trabaja con la tecnología de documentos compuestos y servicios de edición. Provee un marco de trabajo para compartir y subdividir el despliegue de ventanas de tal forma que diferentes componentes pueden compartir el estado visual de la pantalla.

- Facilidad común de gestión de la información (horizontal): incluye almacenamiento de documentos y facilidades para el intercambio de la información. Define estándares de uso para codificar y representar datos, definir e intercambiar metadatos y para modelar información.

- Facilidad común de gestión del sistema (horizontal): la constituyen interfaces y servicios para la gestión, instrumentación, configuración, instalación, operación y reparación de componentes de objetos distribuidos.

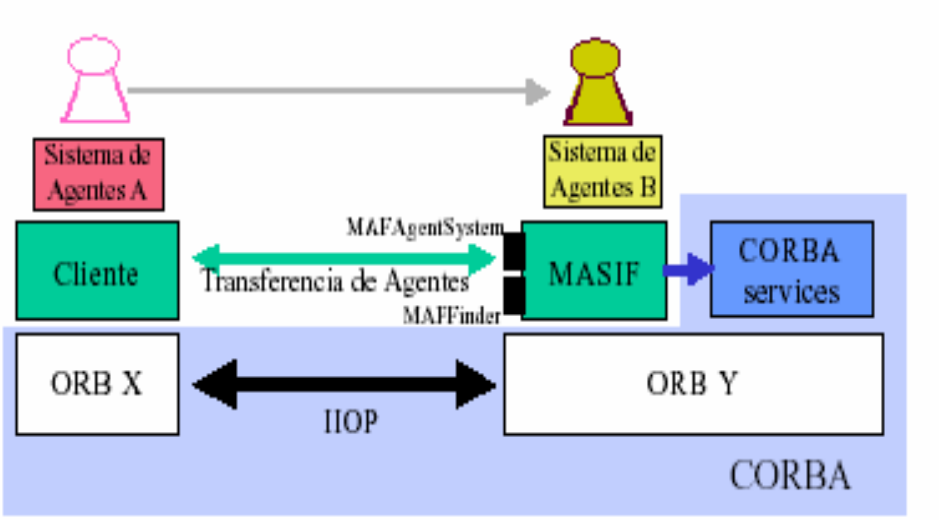
- Facilidad común de gestión de tareas (horizontal): provee un esquema para gestionar el flujo de trabajo, transacciones de agentes, programación, reglas y automatización de tareas. Incluye una facilidad semántica de mensajes para comunicar peticiones orientadas a tareas.

- CORBA para la replica de documentos (vertical): incluye almacenamiento de documentos e imágenes para mover y copiar grandes objetos.

- CORBA para supercarreteras de información (vertical): soporta flujo de información.

- CORBA para la manufactura integrada por computadora (vertical).
- CORBA para la simulación distribuida (vertical): control de tráfico aéreo, escenarios de negocios.
- CORBA para la explotación de gas y petróleo (vertical).
- CORBA para contabilidad (vertical): provee un sistema de contabilidad distribuido que incluye transacciones de negocios, cambios de moneda y ventas, entre otros.

Figura 9. Estandarización de sistemas de agentes en CORBA



Fuente: Organización internacional OMG [7]

La arquitectura *MASIF* es una colección de definiciones de interfaces que permiten la interoperabilidad entre sistemas de agentes móviles, esta especificación esta formada por dos interfaces (Figura 9): la interfaz *MAFAgentSystem* y la interfaz *MAFFinder,* ellas se declaran en el servidor de nombres para que los agentes las puedan tener como referencia.

La interfaz M*AFAgentSystem:* define las operaciones de control de los agentes como creación, terminación, suspensión y recepción de agentes.

La interfaz *MAFFinder:* define las operaciones para registrar y localizar los agentes, lugares y sistemas de agentes.

1.1.2.1.2 *ARPA*

ARPA consorcio centrado en el desarrollo de normas que permiten compartir y reutilizar las bases de conocimiento,

Creo el grupo KSE *(Knowledge Sharing Effort)* en 1990, quien ha desarrollado la especificaciones de lenguaje KQML *(Knowledge Queryng and Manipulation Language)*, el formato KIF *(Knowledge Interchange Format)* y la herramienta ontolingüa que permite la definición de ontologías, estos desarrollos permiten que los agentes tengan la capacidad de comunicación en pragmática, sintaxis y semántica respectivamente.

El lenguaje KQML (se refiere a la pragmática) es un lenguaje y protocolo de comunicación, orientado a mensajes que permite el intercambio de información, es independiente al protocolo de transporte, a la sintaxis del contexto y a la ontología.

KQML esta dividido en tres niveles:

- Nivel de contenido: indica que es lo que el agente quiere comunicar.
- Nivel de mensaje: encapsula el mensaje de intención (existen cerca de 12 tipos de mensajes).
- Nivel de comunicación: este nivel determina el conjunto de parámetros de comunicación a bajo nivel.

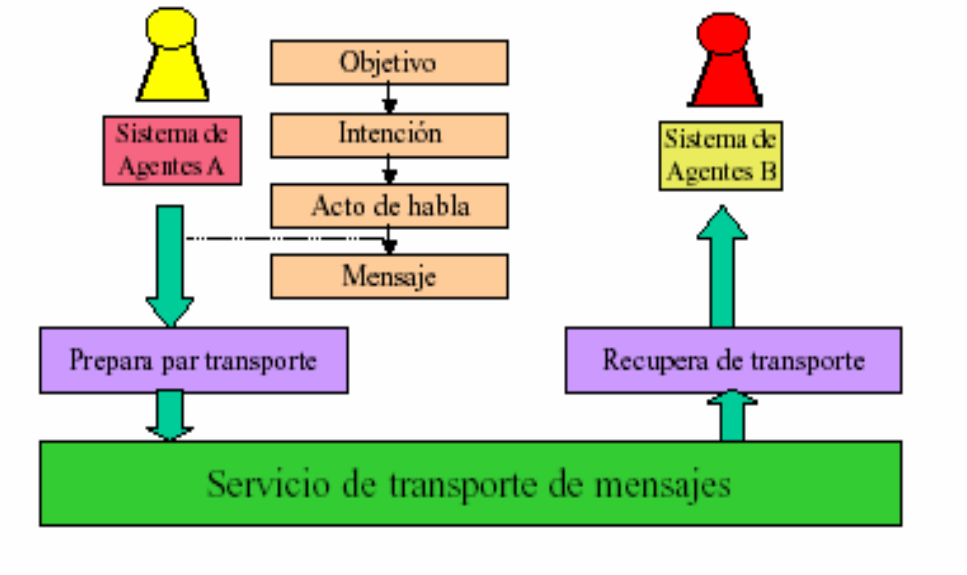
El formato KIF (se refiere a la sintaxis) se encarga de expresar el conocimiento y el meta-conocimiento basado en el cálculo de predicados de primer orden con extensiones que soportan el razonamiento y las definiciones.

La ontolingua (se refiere a la semántica): lenguaje que permite construir, publicar y compartir ontologías que pueden ser traducidos automáticamente a distintos lenguajes de contenido (*KIF, LOOM, prolog,* etc.). La ontología es un vocabulario común que se describe en *KIF* en el que se han acordado significados para describir un dominio, es una conceptualización del mundo en función de objetos, cualidades, distinciones y relaciones. Un agente

satisface una ontología si sus acciones observables son consistentes con las definiciones de la ontología.

Una ontología consiste en un conjunto de términos, los cuales representan el conocimiento, es un diccionario de clases, relaciones, funciones, objetos y constantes que se emplean en el área de conocimiento

Figura 10. Estandarización de sistemas de agentes según KSE



Fuente: Organización internacional ARPA [8]

Como se observa en la Figura 10 el nivel de contenido (mensaje), hace referencia a la información que realmente se está intentando comunicar, es necesario establecer un lenguaje común que represente la información, o que emplee algún tipo de notación para que pueda ser traducible al lenguaje de los agentes. Una vez establecido un lenguaje común son interpretados los mensajes a través de la ontología.

El nivel de intención es el tipo de mensaje. Los tipos de mensajes están clasificados en tres grupos: asertivos, directivos y declarativos. Un mensaje será asertivo cuando comunica un hecho, por ejemplo las primitivas “envía” o “recibe”; directivo cuando indica un comando o acción, por ejemplo una pregunta: “dime todo lo que sepas acerca de ...”, o “te informo que ...”, y declarativo cuando envía información acerca de sus capacidades, como por ejemplo “sé responder cuestiones con el formato siguiente ...”.

La construcción de un conjunto de mensajes, lo suficientemente expresivos como para poder ser utilizados por cualquier tipo de sistema de agentes cooperantes no es una labor sencilla. Investigadores del tema han desarrollado un conjunto de primitivas de comunicación (*performatives*) basadas en la teoría para el modelado de la comunicación humana "*speech act theory"* que describen el conjunto de primitivas necesarias para la coordinación entre agentes.

El nivel de coordinación (acto de habla), se refiere a la forma en el que se establecen los diálogos entre los agentes (Figura 11). ¿Qué convenios han de establecer los agentes para intercambiar mensajes de un modo eficiente?. Existen diferentes protocolos de coordinación: *Contrat Net Protocol*, *AgentTalk*, *COOL*.

Por ejemplo, el protocolo de coordinación *COOL*, es un lenguaje para la coordinación que modela la conversación entre los agentes mediante una máquina de estados finitos. Cada estado representa el punto de la conversación en la que nos encontramos. El paso de un estado a otro vendrá determinado por la llegada del mensaje correcto, de este modo se puede establecer la conducta de los agentes frente a la llegada de los mensajes [17].

1.1.2.1.3 *FIPA*

El consorcio FIPA *(Foundation for Intelligent Physical Agents),* es una industria fundado en 1996, integrada por compañías y universidades relacionadas con telecomunicaciones e informática.

FIPA recoge todas las vistas que tiene un sistema de agentes (gestión, seguridad, movilidad, comunicación, etc.), de tal forma que para cualquiera de esas vistas todos los esfuerzos se centran en una única dirección [17]. Además reune las especificaciones de arquitectura, infraestructura y de aplicación (ver Figura 11).

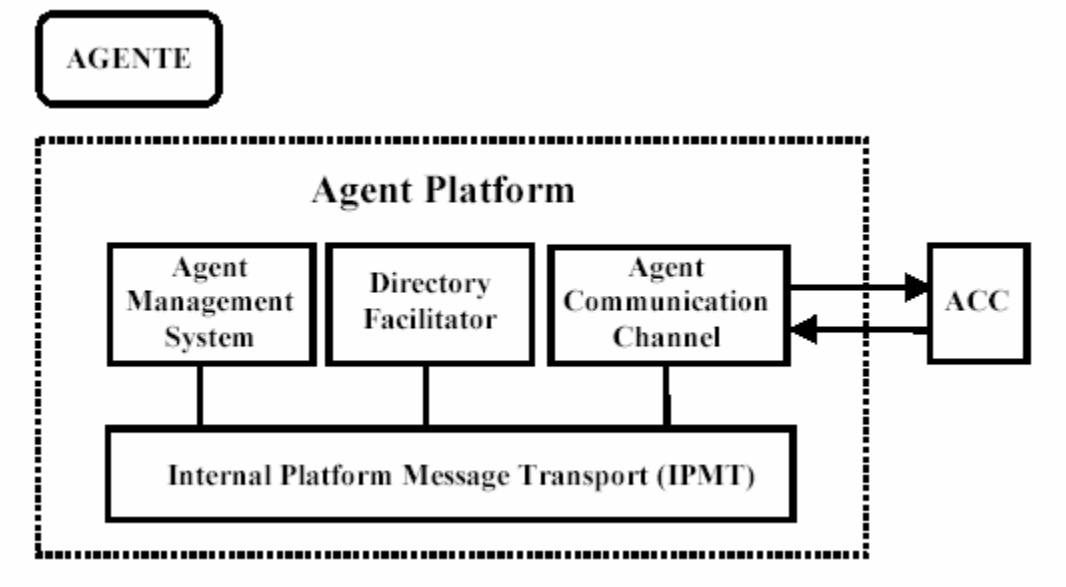
Agent Management proporciona la normativa del entorno donde los agentes FIPA se crean y operan, establece el modelo lógico de referencia para la gestión de agentes (creación, registro, localización, comunicación, migración y terminación de los agentes).

El modelo (ver Figura 11) presenta un conjunto de capacidades lógicas que no implican ninguna configuración física.

El modelo de referencia para la gestión de agentes (*agent management*) está formado por los siguientes componentes lógicos.

Agente: es el componente básico y principal del modelo. Combina una o más capacidades de servicio dentro de un entorno de ejecución integrado y unificado que proporciona servicios de comunicación, acceso a software externo y acceso a los usuarios.

Figura 11. Arquitectura de Agentes FIPA



Fuente: Organización internacional FIPA [6]

Un agente tiene uno o más dueños, debe disponer de una identidad propia proporcionada por un identificador global y único GUID (*Globally Unique Identifier*), el agente puede registrarse con diferentes direcciones para ser contactado remotamente.

Agent Platform (AP)*:* proporciona la infraestructura física y lógica necesaria para que los agentes se puedan ejecutar, la plataforma de agentes está constituida por el hardware (en varios computadores), el sistema operativo, el software de comunicaciones y el software de agentes.

Directory Facilitator (DF): componente que está presente en cualquier plataforma de agentes FIPA, es un agente que proporciona el servicio de páginas amarillas a los demás agentes, un agente puede utilizar el DF para registrar sus servicios o para encontrar servicios ofrecidos por otros agentes.

Agent Management System (AMS): componente presente en cada plataforma, es un agente de gestión que controla el estado y el acceso a la plataforma, proporciona un servicio de páginas blancas que facilita la localización de los agentes a partir de sus nombres.

Agent Communication Channel (ACC)**:** canal de comunicación por defecto entre agentes de diferentes plataformas, soporta el protocolo de comunicación de interoperabilidad IIOP, todos los agentes tienen acceso a él.

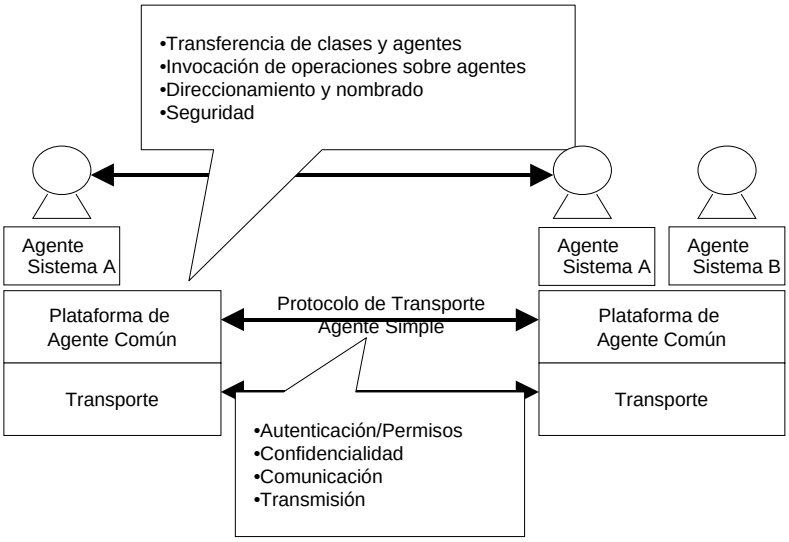
Internal Platform Message Transport (IPMT): técnica de intercambio de mensajes dentro de la misma plataforma. Depende de la implementación del agente.

1.1.2.1.4 *Agent Society*

Organización industrial fundada en 1996, conformada por más de 50 socios, se encarga de facilitar la colaboración y transferencia de tecnología mediante el intercambio de información en el desarrollo de agentes. “Es una arquitectura y protocolo de comunicación genéricos” [10], colabora con FIPA.

Según *Agent Society* (ver Figura 12) presenta la arquitectura de plataforma común de agentes soportando la compatibilidad con *MASIF y FIPA* que permite que los agentes puedan estar desarrollados sobre plataformas, protocolos de transferencia, lenguajes de programación y lenguajes de comunicación diferentes .

Figura 12. Arquitectura de agentes Agent Society



Fuente: Organización internacional Agent Society [10]

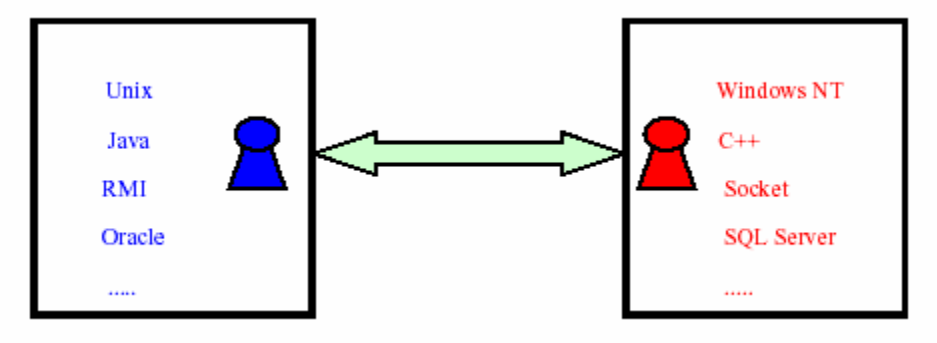
1.1.2.2 CARACTERÍSTICAS GENERALES DE LAS ARQUITECTURAS DE AGENTES

La plataforma de agentes como núcleo de modelo de referencia, proporciona la infraestructura para que los agentes puedan ser desarrollados y usados. La plataforma de agentes está soportada por el hardware y el software (S.O., software de comunicaciones, *middleware* y software de agente).

La características permiten que cada implementación lleve consigo unas decisiones para que los componentes usados mantengan siempre una imagen hacia el exterior[18] como se aprecia en la Figura 13.

El problema clave al que se enfrenta un agente es el de decidir cual de sus acciones debe ejecutar para satisfacer mejor sus objetivos de diseño. La toma de decisiones debe basarse también en el sistema operativo sobre el que este soportado y el entorno donde se este desenvolviendo

Figura 13. Plataforma de Agentes de software Móviles



Fuente: Morales Cuesta Pedro en sistemas multiagentes [23]

Las propiedades del entorno de ejecución son[23]:

1.1.2.2.1 *Accesible o inaccesible*

En un entorno accesible el agente puede obtener información completa y actualizada de su estado. La mayoría de los entornos complejos (como Internet) son inaccesibles.

1.1.2.2.2 *Determinista o no determinista*

En un entorno determinista cualquier acción tiene un único efecto garantizado, no hay incertidumbre sobre el estado que resulte de ejecutar una acción.

1.1.2.2.3 *Episódico o no episódico*

En un entorno por episodios la actuación del agente depende de un número finito de episodios discretos y no hay enlace entre la actuación del agente en escenarios diferentes (por ejemplo: sistemas de clasificación de correo, el

agente decide para el episodio actual, no necesita razonar sobre las interacciones entre este y futuros episodios).

1.1.2.2.4 *Estático o dinámico*

Un entorno estático puede asumirse que permanece inalterado a excepción de que un agente actué sobre él. Un entorno dinámico es aquel que tiene otros procesos operando en él y sufre cambios fuera del control del agente.

1.1.2.2.5 *Discreto o Continuo*

Un entorno es discreto si tiene un número fijo y finito de acciones y percepciones. (Por ejemplo: Juego de ajedrez). Un entorno continuo tiene un número infinito de acciones y percepciones. (Por ejemplo: Partido de fútbol).

1.1.2.3 COMPARATIVO DE LOS ORGANISMOS DE ESTANDARIZACIÓN DE AGENTES

De acuerdo al cuadro comparativo de los organismos internacionales se observa que las características, protocolos y tecnologías son propias de cada una. Ello indica que existen diferencias sustanciales en las especificaciones técnicas de las arquitecturas, el diseño, los lenguajes de desarrollo y los ambientes de sistema operativo. Cabe resaltar que para lograr el intercambio de información entre plataformas de diferente organización es necesario encontrar los puntos de acceso a la plataforma que generalmente se encuentran en los protocolos de comunicación, pues es allí, donde se establecen las reglas y estructuras concretas de la comunicación.

Tabla 2. Organismos de Estandarización Internacional de Agentes Móviles

Organización	Arquitectura	Protocolo	Características	Tecnología
OMG (1989)	CORBA MASIF	IIOP(Internet inter-ORB protocol) ATP (Aglet Transfer Protocol)	Sistemas heterogéneos Sistemas distribuidos Seguridad Movilidad Provee Servicios y facilidades	Aglets MOA
ARPA KSE(1990)	KQML	Contrat Net Protocol. COOL	Reutilización de bases de conocimiento Sintaxis (KIF) Semántica(Ontolingua) Pragmática(KQML)	JatLitle JafMas Jackal
FIPA(1996)	Arquitectura FIPA para la gestión de agentes	ACL (Agent Communication Language) ACC (Agent communication channel)	Gestión Seguridad Movilidad Comunicación	HAP (Home Agent Platform) FIPA-ACL
Agent Society(1996)	Common Agent Platform	SATP (simple agent transport protocol)	Transferencia de clases Direccionamiento Seguridad	CAP (common agent platform)

No obstante, para lograr la comunicación es necesario construir una interfase de software o de hardware que hable uno de los dos idiomas “Si emisor, receptor o intermediario conocen dos idiomas, estos, están manipulando perfectamente las reglas que gobiernan dicha comunicación. Lo que se conoce como estándar de comunicación para los componentes de una red que deseen intercambiar información de diferente tecnología y diferente proveedor” [42].

1.1.3 SISTEMAS DE AGENTES DE SOFTWARE MOVILES

1.1.3.1 Aglets

Un *aglets* es un objeto escrito en java capaz de visitar máquinas que soporten el entorno de ejecución *Aglets WorkBench* (AWB). Fue desarrollado por IBM[19] en Tokio *Research Labs* (1996).

Un *aglet* se puede ejecutar en un *host*, puede suspender la ejecución, ser enviado remotamente a otro e inmediatamente iniciar la ejecución. Cuando en un agente se mueve el código (serialización ó flujo de bits) junto con el estado de todos los objetos, un mecanismo de seguridad hace que este proceso sea confiable.

Las herramientas de desarrollo de *aglets* son:

1. *Tahiti*: Servidor de *aglets*.
2. *Tazza*: Constructor visual de *aglets*.
3. *Fiji*: Lanzador de *aglets* para la *www*.
4. *Agletsd*: Demonio de *aglets*. Se puede ejecutar como *applets* o como aplicación.

1.1.3.1.1 *Características de aglets*

- Provee un modelo de programación de agentes móviles, sin necesidad de modificar la maquina virtual de java (VM).
- Soporta una comunicación dinámica que habilita los agentes a comunicarse con otros.
- Permite reutilizar el código y extender la arquitectura.
- Permite diseño arquitectónico con integración a tecnología *web* y Java.

- Provee mecanismos de seguridad simples y suficientes para los agentes.

- Cifra el código y los datos de un *aglet* utilizando el método de seriación de Java (JOS)

- Traslada agentes utilizando el protocolo para el transporte de a*glets* (ATP).

- Ofrece una Interfaz de programación para *Aglets* (A-API).

- Permite interconexión e intercambio de información entre *aglets* y otros objetos, mediante paso de mensajes.

- El ciclo de vida de un *aglet* puede tratarse por métodos basados en captura de eventos. Los eventos definidos son: creación, clonación, expedición, retractación, eliminación, activación, desactivación y paso de mensaje.

- Ofrece transparencia respecto a la localización del agente.

- Los *aglets* se pueden referenciar remotamente mediante RMI.

- La administración del espacio de datos se realiza por copia, eliminación y referencias remotas.

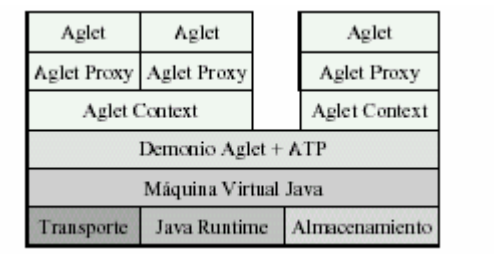
- Permite controlar la seguridad mediante definición de autoridades, privilegios y preferencias[20].

- Desarrollado en Java, herramienta con las siguientes características: seguro, orientado a objetos, robusto, neutral de arquitectura, interpretado e interoperable[40].

1.1.3.1.2 Arquitectura *de aglets*

El entorno computacional (EC) es implementado en el servidor de *Aglets* o *AgletContext*. El *framework* soporta movilidad débil basada en transferencia *(dispatch)* y recuperación *(arrived)* de código. Los *aglets* pueden comunicarse mediante intercambio de mensajes (sincrónico o asincrónico, *unicast o multicast*) o invocación de métodos de otros *aglets*. El AgletProxy actúa de *escudo* protegiendo al *aglet* de operaciones maliciosas (ver Figura 14).

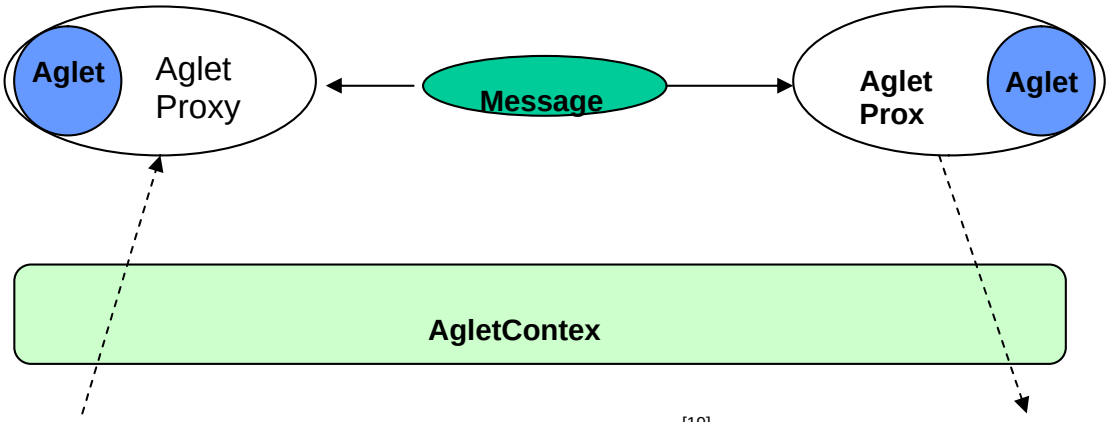
Figura 14.Arquitectura de aglets



Fuente: Mobile Agents - An overview [20]

La Figura 15 muestra las características básicas de la *API,* la funcionalidad de un agente móvil dentro del contexto: *Aglet*, *Aglet Proxy*, *Message* y *Aglet Context*,. Las interfaces, las clases y las relaciones entre ellas se detallan en el Anexo C.

Figura 15. Visión general de la *API* de *Aglets*

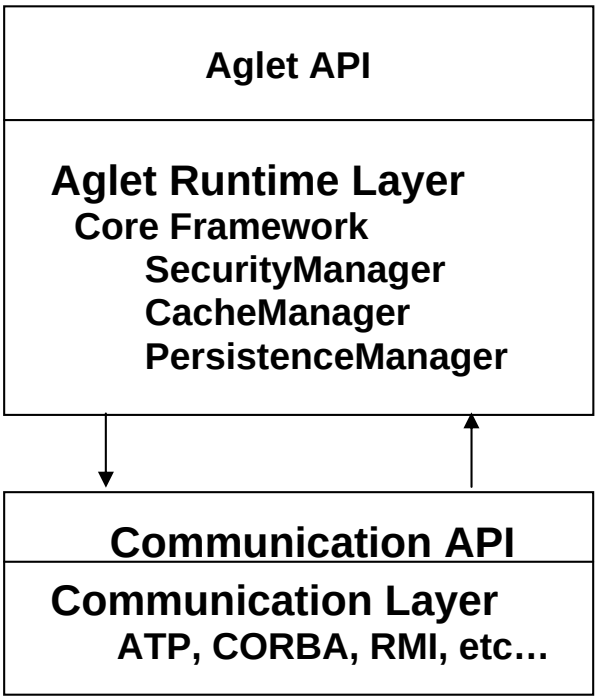


Fuente: http:www.*ibm.com*[19]

- **Arquitectura de la API**

Consiste en dos capas de implementación y dos *APIs* que definen la interfase para acceder a sus funciones (ver Figura 16). La capa de ejecución de *Aglet* es la implementación del *Aglet API*, y la capa de comunicaciones. Ellas se explican más adelante.

Figura 16. Arquitectura de la API de Aglets



Fuente: http:www.*ibm.com*[19]

a. Capa de Ejecución

La capa de ejecución de un *aglet* implementa la interface de cada *Aglet* con *AgletContext. AgletContext* que es el núcleo del sistema y de los subcomponentes, además provee los siguientes mecanismos fundamentales para la ejecución de un *aglet*:

- Serialización y deserialización de los *aglets*
- Llamado y transferencia de clases
- Gestión de referencias y recolección de basura

Los subcomponentes son diseñados para ser extendidos, personalizados y ofrecer varios servicios dependiendo de los requerimientos del ambiente; por ejemplo el manejo de la persistencia para los *applets* puede activar el almacenamiento de un solo *aglet* en memoria. En otro caso puede tener valores por defecto para el manejo de la seguridad en el *web browser.*

Los componentes de la capa de ejecución (ver Figura 16) son:

- ***PersistenceManager*:** Es el responsable de almacenar y serializar los agentes. El código del agente es almacenado junto con el estado.

- ***CacheManager*:** Es el responsable del manejo del *bytecode* usado por el *aglet*, porque al llegar el *bytecode* de un agente necesita transferirse cuando el agente se mueve al destino.

- ***SecurityManager*:** Es el responsable de proteger el *host* de agentes y entidades maliciosas. Este chequea todas las llamadas y ejecuta las permitidas.

b. Capa de Comunicaciones

La capa de comunicaciones es fundamental para la transferencia y serialización de los agentes, es la responsable de recibir el agente en el destino, soporta la comunicación agente-agente y facilita la gestión de cada uno de ellos.

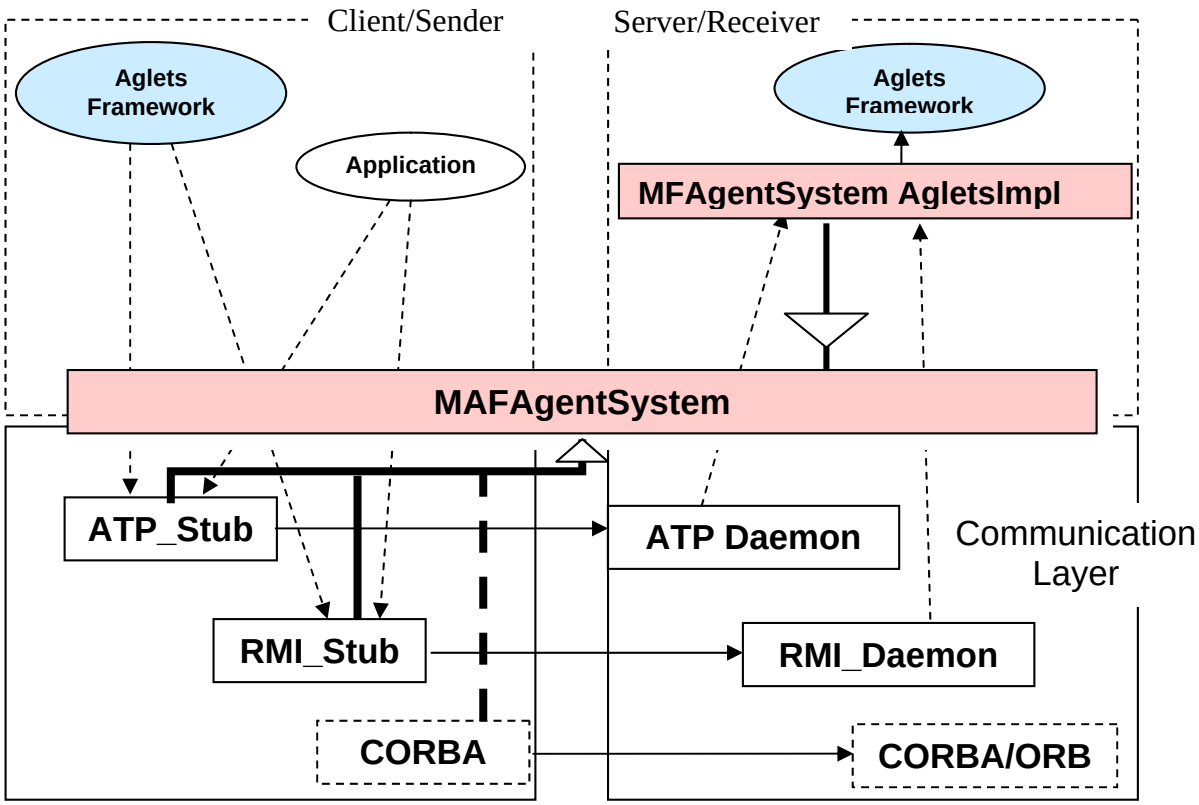
El API define los métodos para crear, transferir, rastrear y gestionar agentes en un sistema de agentes y utiliza el ATP como protocolo de forma independiente. ATP es modelado por el protocolo *HTTP* y es un protocolo que se encuentra en el nivel de aplicación para transmitir agentes móviles, soporta paso de mensajes.

- Arquitectura de la Capa de Comunicaciones

La clase *com.ibm.maf.MAFAgentSystem* (Figura 17) define el conjunto de métodos utilizados para soportar la comunicación. La clase hereda las características de la clase de implementación para proveer las facilidades del sistema de agentes.

El sistema de agentes tiene instanciado un objeto de la clase *stub* y un demonio de comunicaciones por cada protocolo (ATP, RMI), éste envia una solicitud para que en el otro extremo un *aglet (*que implementa a *com.ibm.aglets.MAFAgentSystemImpl)* actualice las solicitudes remotas. El demonio acepta la solicitud y retorna la respuesta a través de *MAFAgentSystem_AgletsImpl.*

Figura 17. Arquitectura de la Capa de Comunicaciones



Fuente: http:www.*ibm.com*[19]

1.1.3.1.3 Ciclo de Vida

El ciclo de vida de un agente móvil[20], representa el recorrido completo de un *aglet*, iniciando por la creación hasta la eliminación (ver Figura 18). Los *aglets* heredan las características de la clase *aglet.Aglet* para obtener los atributos y los métodos (Tabla No. 3) básicos del ciclo de vida.

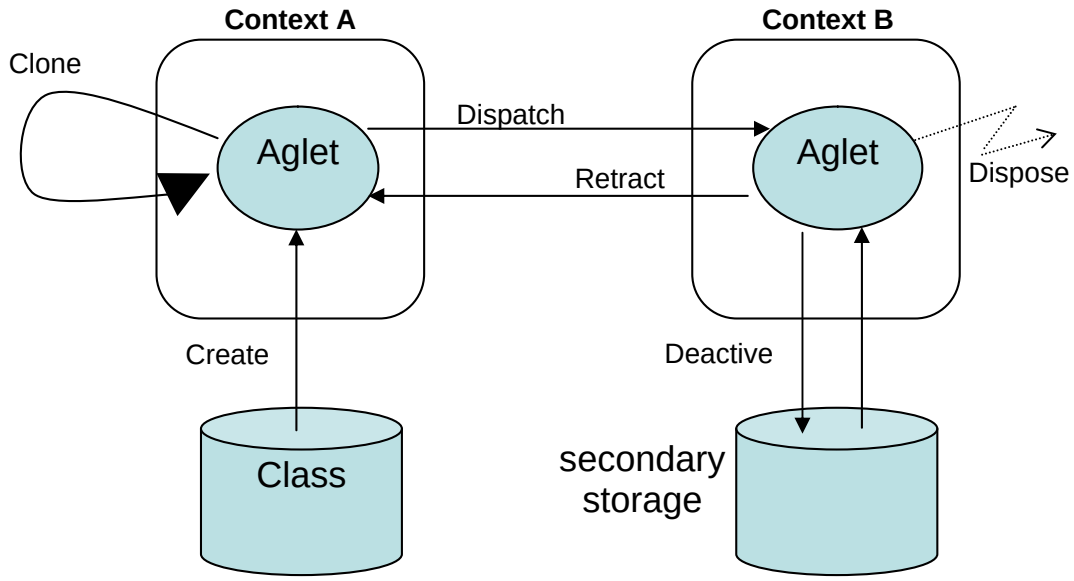
Existen dos formas de crear una nueva instancia de un *aglet*:

1. Instanciar un *aglet* completamente nuevo a partir de las definiciones de una clase, haciendo uso de *AgletContext.crearAglet(URL, codebase, String name, Object init)*. Esta primitiva crea una nueva instancia dentro del contexto especificado. Se puede inicializar el

aglet también con *Aglet.onCreation(Object.init)* del objeto creado pasándole también como parámetro el objeto inicializador que se le pasó a la primitiva *create.Aglet*.

2. Crear una copia de un *aglet* ya existente usando la primitiva *Aglet.clone()*. El *aglet* clonado posee el mismo estado que el *aglet* original. No obstante tiene un objeto *AgletID* diferente y por lo tanto una identidad única.

Figura 18. Ciclo de vida de un Aglet



Fuente: http:www.*ibm.com*[19]

Una vez creado el objeto *aglet* puede ser enviado a una máquina remota o traído de ésta, posteriormente ser desactivado y almacenado en memoria secundaria y por último activado. El objeto puede ser creado, puede ser despachado o retractado desde un *host* remoto, desactivado y almacenado, la activación ocurre después.

Un *aglet* puede despacharse a un servidor remoto por un llamado de *dispatch(URL dest),* el servidor puede tener más de una plataforma de agentes, pero es necesario especificar por cada una la dirección IP, el número de puerto escucha y el nombre del contexto del servidor.

Tabla 3. Operaciones con Aglets

Operaciones con aglets
Creación : *onCreation()* *onDisposal()*
Movilidad *onArrival()* *onDispatching(URL)* *onReverting()*
Clonación *onClone()*
Persistencia *deactivate(long duration)*

Al enviar un *aglet* se suspende su ejecución, se serializa su estado y su código a una forma estándar, luego es transportado a su destino. En el extremo del receptor, el objeto java se reconstruye de acuerdo con los datos recibidos desde el origen, se asigna un nuevo hilo y se reanuda su ejecución.

Los *aglets* soportan la persistencia de los objetos *aglet*. Todos los objetos *aglet* pueden ser persistentes por naturaleza, puesto que es posible que se conviertan en un flujo de *bits*, pudiendo así que el flujo se almacene en memoria secundaria. La primitiva *Aglet.deactivate(long timeout)* permite que un *aglet* sea almacenado en memoria secundaria donde permanecerá dormido durante un número específico de milisegundos, una vez que dicho tiempo ha transcurrido o que otro programa ha solicitado su activación, el *aglet* será reactivado dentro del mismo contexto donde fue desactivado.

A diferencia de los objetos normales en java que son desechados por el recolector de basura, un objeto *aglet* activo puede decidir si desea terminar o no con el método *dispose()* y *onDisposing*() se llamará a continuación para realizar tareas de finalización.

- **Los eventos de los *Aglets* y el modelo de delegación de eventos**

Es imposible hacer migrar un objeto hilo a un servidor remoto o salvarlo a memoria secundaria al mismo tiempo que se guarda su estado de ejecución, sin embargo los *aglets* usan un modelo de eventos para posibilitar a los programadores la implementación de una acción que permita su movilidad.

Antes que un *aglet* se envie a si mismo a un host remoto con el método *dispatch(URL)* se llama el método *OnDispatching()* del objeto *MobilityListener.* Cuando el *aglets* llega a su destino el método *onArrival()* se ejecuta. De esta manera se puede implementar una acción que se realiza como respuesta a estos eventos independiente de cuando se envió o de quien fue el encargado de enviarlo.

La migración se realiza con la función *dispatch()*, especificando el servidor destino mediante un URL que depende de la localización, cuando un agente es reactivado en su destino, se ejecuta su método *run(),* y cuando se permite volver a un agente a su destino su primitiva es *retract()*, se llaman eventos a el conjunto de acciones de un *aglet* (Tabla 4).

Tabla 4. Métodos y eventos de *Aglet*

Método que se ejecuta	Evento	Evento asociado	Método llamado cuando ocurre el evento	Método llamado después de ocurrir el evento	Listener requerido
	Creation		-	*OnCreation() Run()*	
clone()	Cloning	Evento de clonación CloneEvent	OnCloning() OnClone()	*OnCloned()*	*CloningListener*
dispatch()	Dispatching	Evento de Movilidad MobilityEvent	OnDispatching()	*OnArrival() run()*	*MobilityListener*
retract()	Retraction	Evento de Movilidad MobilityEvent	OnReverting()	*OnArrival() run()*	*MobilityListener*
dispose()	Disposal	-	OnDisposing()	-	-
deactivate()	Deactivation	Evento de persistencia PercistencyEvent	OnDeactivating()	-	*PersistencyListener*
activate()	Activation	-	-	*OnActivation() run()*	*PersistencyListener*
sendMessage()	Message Received	-	-	*HandleMessage(Message)*	-

Fuente: Basado en: *http:www.ibm.com*[19] - MARINE Agent Platforms [45]

Los objetos *listener* pueden conectarse ó desconectarse y por lo tanto pueden añadirse a un *aglet,* ademas tambien permiten ser borrados en tiempo de ejecución, es por ello que la clase *listener* puede usarse como una librería de *aglets,* además puede usarse la clase ***LimitadorClones*** para limitar el número máximo de clones.

1.1.3.1.4 Tipos de serialización en la transferencia de aglets

- **Serialización de un objeto *aglet*:** cuando un *aglet* se despacha, clona o desactiva, se convierte en un flujo de bits, para poder recomponerse posteriormente. Los *aglet* usan *ObjectSerialization* de java para transportar el estado de los agentes, todos los objetos que van a serializarse deben implementar *java.io.Serializable* o *java.io.Externalizable,* además si tiene objeto no serializable al que hace referencia de forma directa o indirectamente, debe declararse la referencia como ***transient*.**

- **Serialización de objetos no *Proxy:*** un objeto no *proxy* que sea visible desde un *aglet*, migra haciendo uso de *copy*. Esto quiere decir que una vez serializado el objeto compartido se copia y no volverá a compartirse.

- **Serializacion de objetos *proxy*:** cuando un objeto *proxy* migra, mantiene el indicador de *aglet*, su dirección y reestablece la referencia al objeto *aglet* original. El objeto *AgletProxy* puede mantener la referencia al *aglet* actual, incluso si el *proxy* se transfiere a una máquina remota o se desactiva, siempre que el *aglet* resida en la misma ubicación que el *AgletProxy*.

- **Uso de RMI con objetos remotos:** el interface remoto se define en RMI y se usa para identificar los objetos remotos que poseen métodos que puedan invocarse de forma remota. El programa del cliente RMI funciona con la librería de *Aglet* sin necesidad de tener que hacer ninguna modificación.

- **Objetos *RemoteServer* con RMI:** Estos objetos son estacionarios, por ello no pueden transferirse con el *aglet* (posiblemente se solucione en versiones futuras).

- **Objeto *proxy* de un *aglet* que ha sido despachado:** Actualmente el objeto *AgletProxy* no puede guardar constancia de los *aglet* en

movimiento. Una vez que un *aglet* se despacha, el *proxy* que antes referenciaba al *aglet* ya no es válido [20].

1.1.3.1.5 Mensajes con *Aglets*

Una aplicación o un *aglet* puede comunicarse con otro objeto *aglet* mediante el paso de mensajes. Un *aglet* que desee intercambiar información con otro *aglet* tiene que crear inicialmente un objeto ***message*** y enviárselo al *aglet* con el que desea comunicarse. Un objeto ***message*** tiene como argumentos un atributo ***kind*** que indica el tipo de mensaje. El *aglet* receptor determina que hacer en respuesta al recibido por ***Aglethandle.Message()*** en el campo ***kind***. Un *aglet* tiene un objeto *MessageManager* el cual gobierna todos los mensajes entrantes enviados al agente. Este objeto gestiona y controla el orden y la concurrencia de los mensajes.

Ejemplo de un mensaje entrante a un agente:

```
MyAglet extends Aglet {
    public boolean handleMessage(Message msg) {
        if (msg.sameKind("doJob")) {
            doJob();
        } else if (msg.sameKind("shutdown")) {
            deactivate(0);
        }
    }
```

Tipos de Mensajes: *Aglet* soporta los siguientes tipos de mensajes:

- ***Now-Type:*** *AgletProxy.sendMessage(Message msg).* Para mensajes sincronos, bloquea el receptor hasta que haya finalizado el tratamiento del mensaje.

  ```
  String answer = proxy.sendMessage(new Message("question"));
  System.out.println(answer);
  ```

- ***Future-Type:*** *AgletProxy.sendAsyncMessage(Message msg)* para mensajes asíncronos no permite bloquear la ejecución actual. El método retorna un ***FutureReply*** que puede usarse para obtener o esperar la respuesta.

```
FutureReply future =
        proxy.sendAsyncMessage(new Message("question"));
    int num_task = 10;
    // trabajo privado mientras el tiempo este en 10
    while(future.isAvailable() == false && num_task-- >0) {
        doPrivateJob();
    }
System.out.println( (String)future.getReply() );
```

- ***Oneway-type*:** *AgletProxy.sendOnewayMessage(Message msg)* para mensajes asíncronos, no permite bloquear la ejecución actual. Se diferencia al *future-type* en que es colocado al final de la cola de mensajes incluso si el mensaje se envía al propio *aglet,* no retorna ningún valor.

```
proxy.sendOnewayMessage(new Message("question"));
```

- ***Delegation-type:*** *AgletProxy.delegateMessage(Message msg), en este caso* un mensaje a un objeto es pasado como argumento en *handleMessage(Messagemsg),* y puede ser usado para la ejecución de acuerdo a la clase de mensaje. El método retorna un valor boolean que indica si ha sido manipulado o no.

```
public boolean handleMessage(Message msg) {
      if (msg.sameKind("sayHello")) {
          System.out.println("Hello");
          return true; // i know this message...
      }
      return false; // false, otherwise
       }
```

- **Paso de mensajes de forma Remota**

Los *aglets* dan soporte al paso de mensajes de forma remota, permitiendo que los objetos puedan comunicarse con objetos locales o remotos. Los parámetros y objetos devueltos por mensajes pueden ser de cualquier tipo

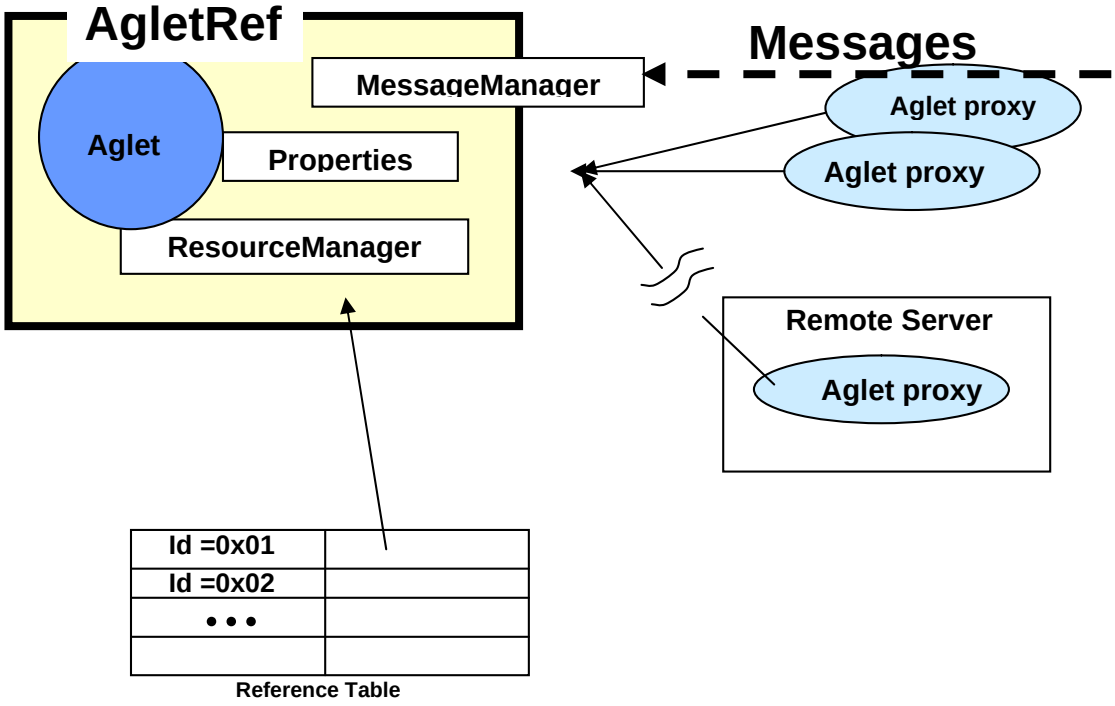
de java que implemente ***java.io.Serializable***, realizando la serialización y deserialización mediante *ObjectSerialization*.

Enviar un mensaje remoto se diferencia de enviar un *aglet* porque al enviar un mensaje remoto no se realiza ninguna transferencia de código binario. El paso de mensajes de forma remota suele usarse para comunicar *aglets* que se encuentran en máquinas diferentes.

1.1.3.1.6 Estructura de un Objeto Aglet

Un objeto *aglet* tiene una representación interna *AgletRef,* el cual tiene los componentes que controlan la llegada de mensajes y gestiona los recursos de seguridad y de información *(AgletInfo).*

Figura 19. Estructura de un Objeto Aglet



Fuente: http:www.*ibm.com*[19]

Cuando es transferido un *aglet*, los mensajes llegan al esqueleto del agente ó *AgletRef,* donde son registrados en la tabla de referencia por el manejador de

recursos, quien a su vez mapea el agente para que posteriormente sean liberados los recursos en el *host* local (Ver Figura 19).

1.1.3.1.7 Carga y movilidad de clases

Cuando se transfiere un agente, el sistema carga la clase en el sitio remoto en tiempo de creación (onCreation()) para el agente que esta en ejecución en el sitio local.

Una clase que soporta movilidad tiene las siguientes reglas:

1. Cargar la clase: como escoger la clase y cargarla.
2. Transferir la clase: que clase es transferida y cuando.
3. Reanudación y actualización de la clase: como una clase es escogida para cargarse y reanudarse.

- **Proceso de carga de la clase**

En la carga de clases el sistema de agentes crea un *aglet* en forma dinámica en tiempo de ejecución, por ello las definiciones de las clases *(bytecode)* del *aglet se* tienen que cargar en tiempo de ejecución.

El cargador (*AgletClassLoader)* carga primero una clase y luego todas las interfaces y clases faltantes, éste seleccionará de acuerdo con el código base, el parámetro que se pasa al método *AgletContext.CreateAglet(URL, códigobase, String nombre, Object init).* El código base es un atributo inmutable de un *aglet* y nunca cambia durante su ciclo de vida[35].

Al pasar como argumento el valor nulo a la primitiva *AgletContextcreateAglet(URL codigobase, String nombre, Object init),* se buscarán las clases especificadas en los directorios contenidos en la variable de entorno *AGLET_PATH* y *CLASSPATH*, el *AgletClassLoader* delega la solicitud al cargador de clases del sistema, la clase usada de forma directa por la clase también se cargará. Si la definición del *aglet* se encuentra en *CLASSPATH* el *aglet* no tendrá un cargador propio y se cargará usando un cargador del sistema.

- **Transferencia de clases**

  Cuando se despacha o desactiva un *aglet*, el código base del *aglet* se almacena en un flujo y se usará para crear un nuevo cargador de clases para el *aglet*. Si ya existe un cargador de clases con el mismo código base en la *cache*, se usará éste en lugar de crear uno nuevo para definir las clases que se reciban. Todas las clases usadas directamente se cargarán usando el mismo cargador de clases o el cargador del sistema.

  El *AgletClassLoader* dispone de una *caché* de clases que almacena todas las clases a partir de su código base, si la clase solicita un *aglet* que ya se encuentre en la *caché*, se usará dicha clase a partir del código base, de esta forma se reduce la carga en la red y se mejora el rendimiento. En ocasiones los *bytecode* de un *aglet* se encuentran desactualizados y entran en conflicto con una clase almacenada en la *caché*.

  Cuando un *aglet* envía el estado con un valor igual a *runtime* el sistema comprueba que las definiciones de las clases recibidas son completamente compatibles con las ya almacenadas en la *caché*, creándose un nuevo cargador si existe al menos una clase que entre en conflicto.

  En la movilidad las definiciones de las clases se transmiten junto con el estado del agente en tiempo real al momento de transferir el agente. Los *aglets* clasifican las clases en tres categorías según el código base, como son[34]:

  a. Clases del sistema: Son las clases que se cargan a partir del *CLASSPATH* y por lo tanto no tendrán un código base.

  b. Clases con código Base: Es una clase que se carga a partir del código base del *aglet*.

  c. Otras Clases: Un *aglet* podrá hacer referencia a otras clases que se cargan por otros *aglets* desde diferentes códigos base, esto sucede cuando un *aglet* recibe mensajes que tienen como argumento un objeto perteneciente a una clase que se cargó en el código base del emisor.

- **Reanudación y actualización de clases**

Cuando un *aglet* se activa o desactiva, el *bytecode* de la clase del *aglet* se almacena en forma de flujo durante el proceso *ObjectSerialization*, posteriormente se envía a su destino o se guarda en memoria secundaria. Las clases del sistema no se almacenan en el flujo. En consecuencia nunca se enviará al destino. Si algunas clases no se encuentran disponibles en el destino, la operación de *dispatch* fallará y se lanzará una excepción.

Si una clase local tiene el mismo nombre que otra clase en el flujo recibido, la clase local se selecciona y se usa para reconstruir el *aglet*. Si la clase local no es compatible con la clase original en términos de serialización se lanzará una excepción. Cuando se envía código binario, el *runtime* consulta al *SecurityManager* para determinar si el *aglet* tiene permisos para ser enviado.

Es necesario tener en cuenta que una solicitud enviada desde el servidor para buscar un archivo de clase debe referenciarse en el *SecurityManager,* de lo contrario no se permite traer el fichero especificado, el *SecurityManager* rechazará la solicitud.

1.1.3.1.8 Protocolo de Transferencia de Agentes (ATP)

El protocolo de transferencia de agentes ATP *(Agent Trasfer Protocol*) (ver Figura 21), es un protocolo a nivel de aplicación diseñado para transmitir un agente de una forma que sea independiente al sistema de agentes. El formato es definido por una línea de petición, campos de cabecera y un contenido. La línea de petición específica el método de petición a ejecutar, mientras los campos de cabecera contienen los parámetros de la petición.

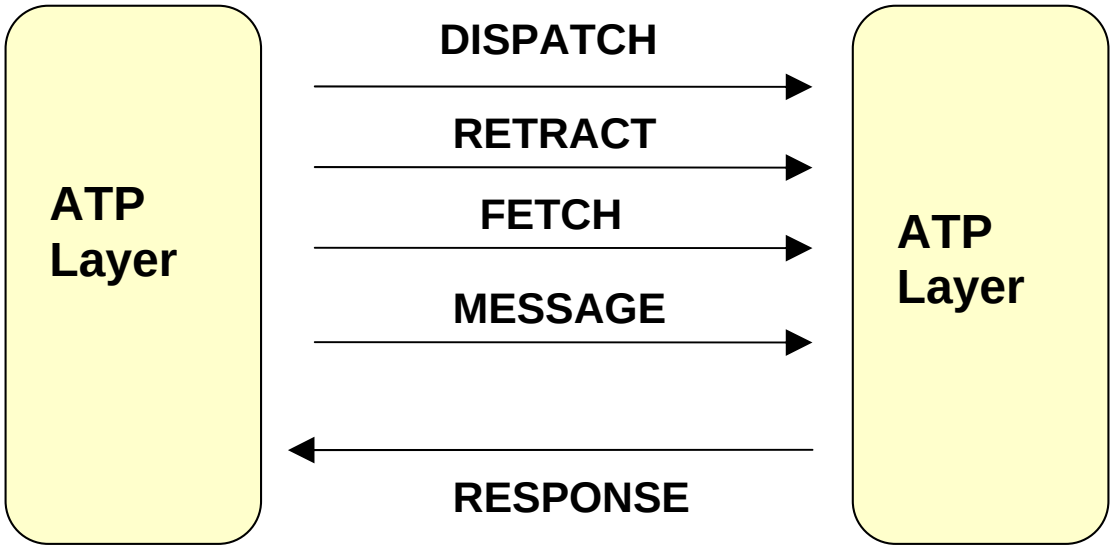
La comunicación se realiza a través de los siguientes métodos (ver Figura 20):

- ***Dispatch*:** este método solicita al sistema de agentes destino que reconstruya un agente a partir del contenido de la petición y que reinicie la ejecución del agente. Si la petición ha tenido éxito, el emisor debe finalizar la ejecución del agente y devolver todos los recursos utilizados por él.

- ***Retract***: este método solicita a un sistema de agentes destino que devuelva un agente específico al emisor. El receptor será el responsable de reconstruir el agente así como de reanudar su ejecución. Si el *aglet* se transmite con éxito, el receptor pondrá fin a la ejecución del agente y devolverá los recursos consumidos por éste.

- ***Fetch***: es similar a *GET* en *HTTP*, este método solicita al receptor que recopile y envié cualquier información que se le especifique (normalmente archivos .*class*).

Figura 20. Protocolo de Transferencia ATP



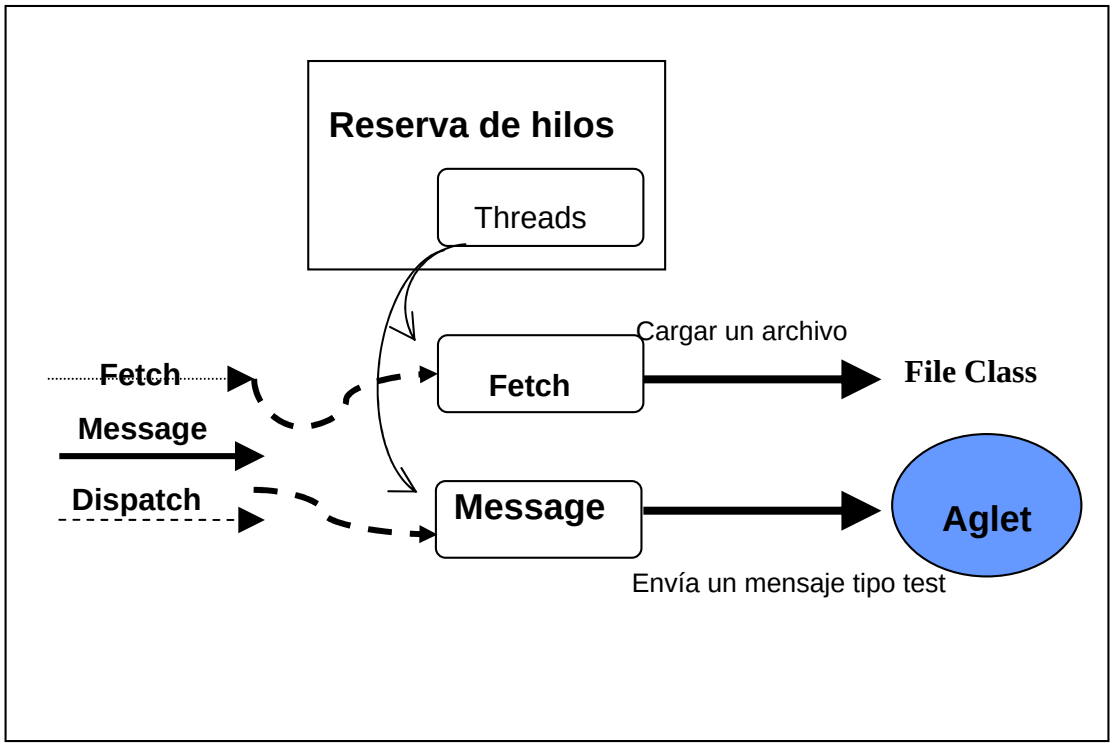
Fuente: http:www.*ibm.com*[19]

- ***Message***: este método se utiliza para enviar un mensaje a un agente especificado por el identificador del agente, devolviendo un valor como respuesta. Aunque el protocolo adopta una forma de solicitud/respuesta, no establece ningún tipo de reglas para un esquema de comunicación entre agentes.

1.1.3.1.9 Uso de Hilos en ATP

La implementación ATP no requiere de una creación de un nuevo hilo para manejar una solicitud de una nueva conexión, ATP dispone de una reserva de hilos y lo asigna para gestionar la solicitud, debido a ello es posible manejar diferentes solicitudes de forma eficiente, ya que un *aglet* posee sus propios hilos de control que garantiza que todos los hilos usados pertenezcan al grupo de hilos del *aglet* y que no coincidan con los hilos de reserva ver Figura 21.

Figura 21. Asignación de hilos en ATP



Fuente: http:www.*ibm.com*[19]

1.1.3.1.10 Seguridad de agentes

La seguridad en un sistema de agentes es esencial, porque acepta o rechaza agentes hostiles en los destinos que pueden estropear la privacidad. *Aglets* provee un sin número de mecanismos de seguridad. La seguridad inter-EC

(Entorno Computacional) provee encriptación, autenticación, calidad de servicio, entre otros. La seguridad intra-EC está basada en listas de permisos (a través del administrador de seguridad Java) y autenticación [21].

La plataforma provee los siguientes servicios de seguridad:

- **Autenticación de remitente, creador y propietario del agente:**

  a. Quien es el responsable de este agente.
  b. Quien es el responsable del código del agente.
  c. Quien puede obligar a forzar la seguridad con el agente (código y estado)

- **Autorización para el agente ó propietario:**

  a. Que puede hacer el Agente (ejemplo: puede este agente acceder a los archivos)

- **Seguridad de comunicación entre sistemas de Agentes:**

  a. Puede el agente proteger la privacidad

- **No- repudiación y Auditoría:**

  a. Como puede asegurarse que la caída del sistema es real
  b. Actividades sensibles de seguridad, recordar, administrar, capaz de formar grupos de auditoría

- **Autenticación de dominio**

Todos los servidores disponen de autenticación siempre y cuando tengan un dominio. Todos los servidores que pertenezcan a un dominio compartido tiene una llave secreta (*secret key*) y usan el MAC *(Message Authentication Cod)* que es un valor obtenido por técnica *hash*.

- **Chequeo de la integridad de la comunicación**

El sistema envía el MIC (*Message Integrity Code)* de la misma forma que el MAC. El MIC es un valor del contenido y de la llave secreta y es enviado junto con el contenido. El MIC controla el contenido de la información y la llave secreta.

- **Autenticación de agentes:**

La plataforma de agentes móviles de *Aglets* tiene una interfaz gráfica y una base de datos que permite manipular las políticas de seguridad. Entre ellas están: acceso de lectura/escritura a los archivos y librerías, instanciación de objetos, advertencia por uso de ventanas.

Un ejemplo es:

```
java.io.FilePermission        : File read/write/execute
java.net.SocketPermission     : Socket resolve/connect/listen/accept
java.awt.AWTPermission        :
                        showWindowWithoutWarningBanner, accessClipboard
java.util.PropertyPermission  : Java property
java.lang.RuntimePermission   : queuePrintJob, load library
java.security.SecurityPermission  : getPolicy, setSystemScope
java.security.AllPermission       : all other permissions

com.ibm.aglets.security.ContextPermission  : context property,
                                                        start, shutdown
com.ibm.aglets.security.AgletPermission    : dispatch, deactivate,
                                                        etc.
com.ibm.aglets.security.MessagePermission  : messaging
```

La siguiente configuración da al propietario del *aglet* “oshima” que tiene el code base en *"http://trusted.com"* con permisos de lectura y para el directorio "C:\temp”, tiene permisos de escritura.

```
grant codeBase "http://trusted.com", ownedBy "oshima"
{
  permission java.io.FilePermission  "C:\\temp", "read","write";
  permission com.ibm.aglets.security.ContextPermission  "property.*", "write";
  permission com.ibm.aglets.security.AgletPermission  "oshima", "dispose";
  permission com.ibm.aglets.security.MessagePermission  "oshima",
"message.getResult";
}

//El archivo de seguridad se encuentra en $HOME/.aglets/security/aglets.policy.
```

1.1.3.1.11 Ventajas de *aglets*

- La plataforma *Aglet* soporta diferentes sistemas de transferencia de información *ATP, CORBA, RMI*.

- La existencia de lugares y dominios proporciona la posibilidad de modularizar un sistema de agentes, así como definir políticas de seguridad propias de cada lugar.

- Como java es portable, seguro y autónomo tiene una gran aceptación en el desarrollo de aplicaciones.

- Se pueden utilizar herramientas gráficas y de bases de datos para intercambiar información.

- El diseño del sistema permite que los agentes puedan viajar por Internet y proporcionen la posibilidad de enlazar *aglet* con navegadores.

- *Aglet* proporciona primitivas sencillas y flexibles de migración y comunicación.

- El modelo de seguridad es robusto.

1.1.3.1.12 Desventajas de *aglets*

- No se conserva el estado de ejecución de agentes, debido a que la máquina virtual de java no soporta accesos a su pila. Este problema no es posible mejorarlo hasta que se modifique la máquina virtual.

1.1.3.2 MOA (Agentes y Objetos Móviles)

Esta arquitectura la ofrece el *Open Group*[3], en la actualidad se encuentra en fase de diseño y se han definido las especificaciones funcionales y las de implementación del modelo de objetos. MOA como arquitectura utiliza java y permite crear objetos y agentes móviles, un objeto móvil contiene tareas y por lo tanto está activo, es independiente y actúa en nombre de un usuario [22].

1.1.3.2.1 Características de MOA

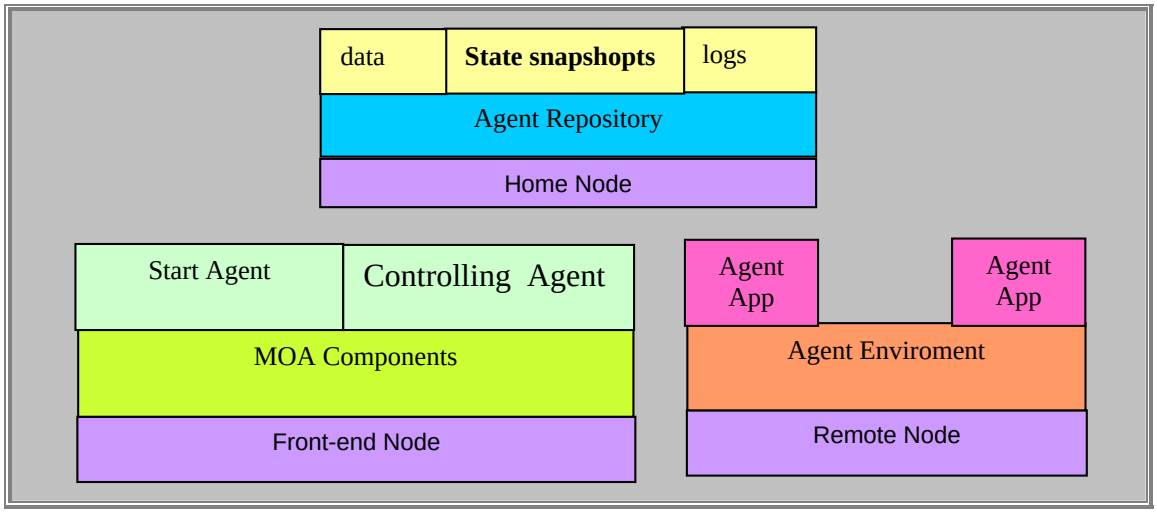
- Proveer puntos de comprobación para poder reiniciar las clases Java.
- Permite que los objetos y agentes móviles escritos en Java sean traducidos al lenguaje *Telescrip*t de *General Magic*.
- Ofrece un esquema de nombres apropiado.
- Conserva los canales de comunicación tras cada migración de un objeto.
- Soporta comunicación Asincrónica (*one-way messages*) o sincrónica (*RPC-like*).
- Define una interfaz para el control de agentes, ofreciendo suspensión, reanudación, eliminación, localización y monitorización.
- Soportar control histórico de la actividad de los agentes.

1.1.3.2.2 Arquitectura MOA

Los tres tipos de nodos que integra el sistema MOA (ver Figura 22): *Front-end Node* permite usuarios para el control y monitoreo del agente. El *Home Node* es usado como repositorio para datos de los agentes y el *Remote Node* es donde típicamente completa su ciclo de vida el agente.

La arquitectura MOA tiene como base el modelo *Telescript-like* (aunque tiene diferencias para no infringir la patente). El *Agent Environments (AE)* soporta el agente viajando por diferentes lugares. Cada *Place* (lugar) acepta agentes y almacena información. El *Host Environment* contiene varios objetos. El nombre del servidor tiene pistas de localización de agentes y otros objetos mientras un *monitor Server* controla y monitorea los objetos.

Figura 22. Arquitectura MOA



Fuente: http:www.usenix.org/publications[46]

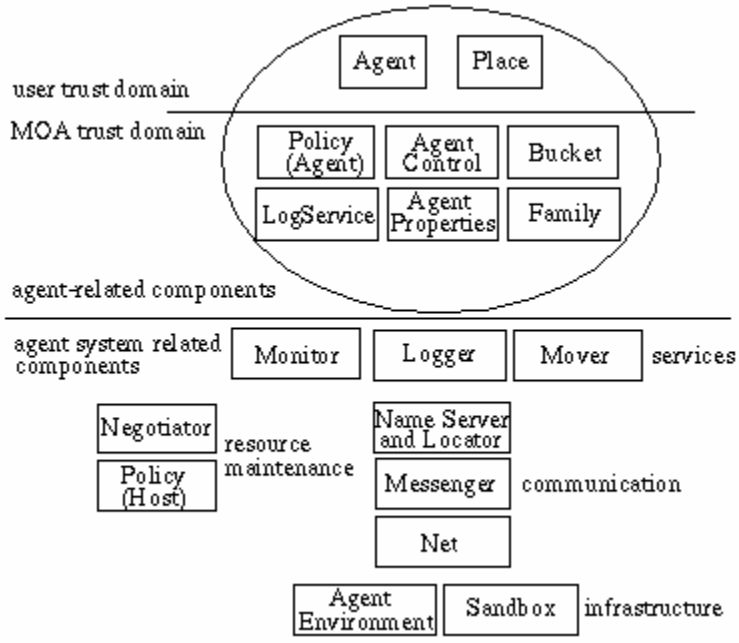
Los objetos MOA en el *Remote Node* pueden ser clasificados como agentes, el agente puede migrar. Agente y Lugar están dentro de un dominio de usuario, mientras que otros componentes pertenecen a un dominio MOA (ver Figura 23).

Los componentes son:

- Del sistema:
  a. De servicio: *monitor, logger y mover*.
  b. De mantenimiento: n*egociator, policy host*.
  c. De comunicación: *net, Messeger, Name server, y locator*
  d. De infraestructura: agent environment, sandox.

- De un agente: agentes y lugar, *agent pocicy, controlagent, bucket, logservice* y un *agent propierties y family:*

Figura 23. Objetos MOA



Fuente: http:www.usenix.org/publications[46[

1.1.3.2.3 Ciclo de Vida

Contempla creación, suspensión, reanudación, eliminación, localización y monitoreo (Tabla 5). Estas operaciones son similares a la de la arquitectura de *Aglets*.

Tabla 5. Operaciones MOA

Operación MOA
Creación
Suspensión
Reanudación
Eliminación
Localización
Monitorización

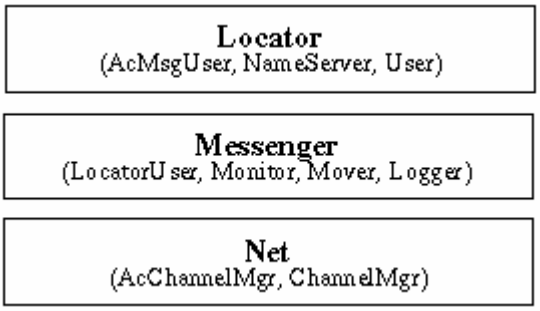
1.1.3.2.4 Capa de Comunicaciones MOA

La capa de comunicaciones se encuentra soportada por *agent environment* y *sandbox* que son los componentes de infraestructura (ver Figura 23) y está conformada por tres niveles :

Net: Soporta tramas que son la base de la comunicación.
Messenger: Es el mensajero de los objetos de acuerdo a la localización y se encarga de la migración de objetos.
Locator: Introducen transparencia en la localización y migración de objetos.

El sistema *MOA* se construye sobre la *JVM (Java Virtual Machine)* con *sockets*, que proporciona un nivel más alto de abstracción, tal como el paso de mensajes entre objetos *MOA* (agentes, lugares y servidores). El canal de comunicaciones soporta tramas de objetos. El paso de mensajes se especifica de acuerdo a la aplicación.

Figura 24. Capa de Comunicaciones



Fuente: http:www.usenix.org/publications[46]

El empaquetamiento de la información sobre la red soporta varios canales abiertos con retransmisión automática. El sistema destino puede opcionalmente rechazar o pedir un canal de acuerdo a los recursos del sistema. Esto ocurre en el sistema del agente. Para ello solamente requiere el nombre del agente. El sistema de agentes resuelve la localización (Figura 24) con la ayuda del localizador de objetos de una manera distribuida.

1.1.3.3 ARA (Agentes para la Acción Remota)

El sistema *ARA (Agents for Remote Actions)* desarrollado en la Universidad de *Kaiserslautern* Alemania en el año de 1997. Tecnología que se encarga de agregar movilidad a los lenguajes de programación mediante un *middleware* entre el sistema operativo y las aplicaciones.

1.1.3.3.1 Características ARA

Las principales características de ARA son:

- Permite agentes móviles interpretados en Tcl y en C/C++ usando MACE (Mobile Agent Code Environment).
- Los agentes pueden transportarse mediante TCP o SMTP.
- Utiliza permisos para todos los recursos relevantes del sistema (ficheros, conexiones, ancho de banda, espacio en disco, entre otros) mediante listas de control de acceso.
- Permite la codificación y certificación de los agentes móviles.
- Los agentes ARA están soportados por un servidor independiente del lenguaje y el intérprete del lenguaje en el que fue escrito.
- ARA implementa la movilidad a través de migración proactiva.
- La comunicación entre entidades ARA, se realiza mediante intercambio de mensajes asíncronos.

1.1.3.3.2 Arquitectura ARA

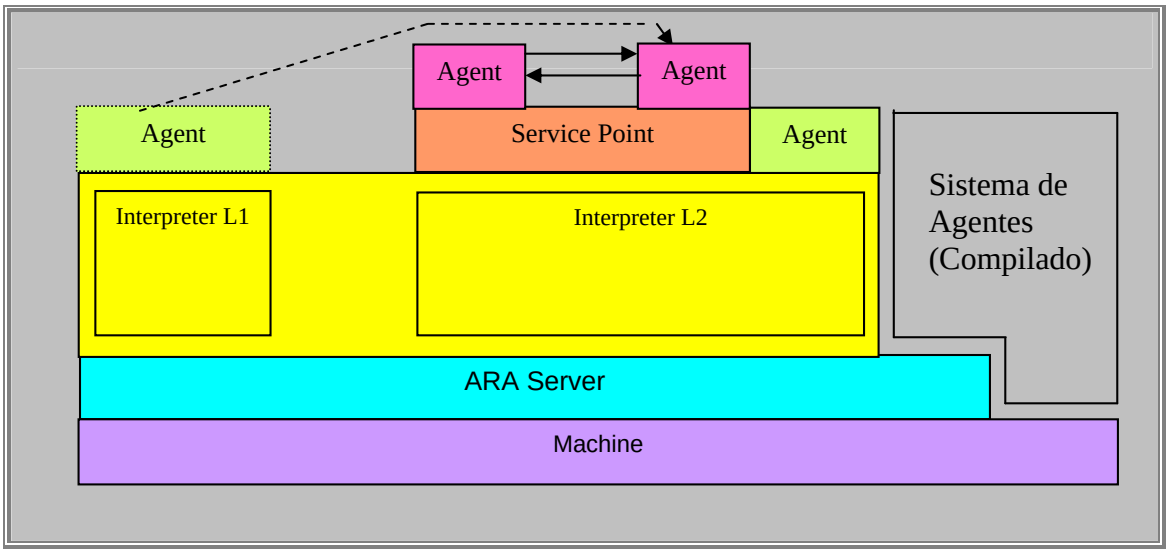
La arquitectura ARA se centra en los siguientes conceptos[20]:

- *Machine(sistema Operativo)*: representa el mecanismo de soporte ubicado en cada uno de los *hosts*.

- *Place*: es el dominio de servicios relacionados lógicamente que se manejan bajo la misma política de seguridad y se ejecutan en un *host* dado.

- *service point*: es una instancia de un agente ofreciendo un servicio particular a los otros agentes. Cuando se crea un punto de servicio, el agente que anuncia el servicio se transforma en el servidor de ese servicio y los agentes que se subscriben a ese punto de servicio serán los clientes.

Figura 25. Arquitectura ARA



Fuente: Pavón Maestras Juan [20].

- *Allowance*: representan los permisos que poseen los agentes para acceder a los recursos. Los agentes pueden compartir permisos debido a que se pueden agrupar en distintos niveles. Los permisos pueden ser globales o locales. Un permiso global se le asigna al agente en el momento de su creación mientras que los permisos locales son asignados por un *place* para restringir el acceso a sus recursos.

- *Interpreter*: para soportar compatibilidad con los lenguajes de programación y no restringir a los agentes a ser escritos en un lenguaje determinado, la arquitectura ARA provee una interfaz para acoplar

distintos tipos de lenguajes. Por lo tanto esta arquitectura está compuesta por diferentes intérpretes al mismo tiempo sobre el *server* o *core* (ver Figura 26).

1.1.3.3.3 Ciclo de Vida

La creación y el borrado de un agente son funciones básicas ofrecidas por el núcleo. A cada agente le es asignado un identificador único, el cual no puede ser modificado. En ARA los agentes pueden reproducirse, suspender y reanudar temporalmente su ejecución, dormir por un tiempo e incluso terminar su ejecución.

Estas funciones de control de agentes pueden ser usadas para formar un equipo de trabajo sobre una tarea común. Los agentes pueden hablar uno con otro, intercambiar información, ofrecer, requerir y negociar servicios entre ellos.

ARA permite la interacción entre agentes locales, existen alternativas para la interacción, incluyendo archivos, áreas compartidas de memoria, intercambio de mensajes y llamadas a procedimientos especiales. El núcleo proporciona un punto de servicio para la interacción llamado punto de encuentro *(meeting place)* en donde los agentes interaccionan como clientes y/o servidores.

El servidor establece límites para los agentes visitantes que se ejecutan, los agentes requieren recursos (*allowances*) y ellos registran la cantidad de recursos que pueden consumir en el futuro.

El sistema asegura que un agente nunca sobrepase su asignación de recursos. Un grupo de agentes puede compartir una asignación de otro agente. Los agentes pueden indagar sobre el estado actual de su asignación o lo puede transmitir a otro.

1.1.3.3.4 Operaciones ARA

Los principales comandos para gestionar agentes son (Tabla 6):

ara_agent:	crea un agente.
ara_exit:	permite la terminación voluntaria del agente.
ara_kill:	elimina un agente, siempre y cuando haya

privilegios.

ara_supend: suspende el agente y lo envía a su estado de espera.

Ara_active: activa un agente.

ara_sleep: suspende a un agente por un cierto período de tiempo.

ara_go: permite a un agente migrar.

Tabla 6. Operaciones ARA

Operación ARA
Creación
Borrado
Reproducción
Suspensión
Reanudación
Terminación

Los agentes ARA pueden moverse e interferir en su ejecución. Esta migración implica que el estado de la ejecución tiene que ser extraído del sistema origen y reinstalado en el sistema destino. En esta arquitectura el agente es transformado a una forma portable antes de ser movido. El estado de ejecución de un agente está compuesto por el estado del intérprete específico por un lado y por el estado de su proceso general en el núcleo de ARA sobre el que reside.

1.1.3.4 Grasshopper

Desarrollado por la empresa IKV++ (Berlín, Alemania)[43] en 1998. Basada en la plataforma de agentes móviles de java, normalizado con los estándares MASIFde OMG y FIPA.

1.1.3.4.1 Características Grasshopper

- Soporta varios protocolos de comunicación: Java RMI, Sockets, SSL, CORBA IIOP.

- Los modos de comunicación son síncrono, asíncrono y multidifusión.
- Provee un conjunto de herramientas de desarrollo de interfaces GUI.
- Provee una interface de agencia y de registro.

1.1.3.4.2 Arquitectura Grasshopper

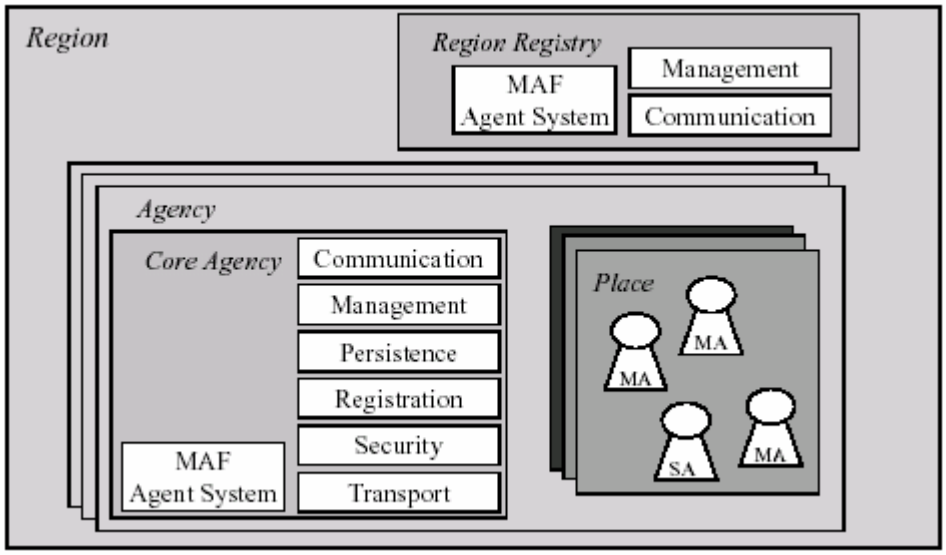
Los componentes de la arquitectura G*rasshopper* (ver Figura 26) son:

- *Region*: compone todo el sistema:
  - *Region Registry.*
  - *Agency.*

- *Region Registry*: compuesta por:
  - *MAFAgentSystem*
  - *Management*
  - *Communication.*

- *Agency*: compuesta por:
  - *Core Agency.*
  - *Place*

Figura 26. Arquitectura Grasshopper



Fuente: http://www.informatica.us.es/~ramon/tesis/CORBA/Seminario-MASIF/ [43]

- *Core Agency* tiene las capas:
  - *Communication*
  - *Management*
  - *Persistence*
  - *Registration*
  - *Security*
  - *Transport*
  - *MafAgentSystem*

- Place:
  - Contiene los agentes

1.1.3.5 Odyssey

La implementación en java de la arquitectura *TeleScript* se conoce como Odyssey. Telescript[36], es un producto comercial de *General Magic Incorporated* para desarrollar agentes móviles. *Telescript* [37] es un lenguaje orientado a objetos desarrollado para construir aplicaciones distribuidas seguras.

1.1.3.5.1 Características Odyssey

- Cuenta con un conjunto de bibliotecas Java que proporcionan la funcionalidad de agentes móviles. *Odyssey* no permite que sus agentes sean persistentes.

- La clase *worker* de *Odyssey* es una subclase de la clase agente que soporta una tarea por destino. Un *worker* esta estructurado como un conjunto de tareas y un conjunto de destinos.

- El lenguaje *Telescript* provee movilidad fuerte a través de un mecanismo de migración proactiva.

- Los entornos computacionales de Telescript son intérpretes del lenguaje *Low Telescript. Low Telescript* es una representación portable y compacta que resulta de la compilación de *Telescript*.

- El *place* es una unidad de ejecución que puede contener otras unidades de ejecución y puede definir otros mecanismos de seguridad.

- *Telescript* utiliza el concepto de propiedad para administrar el espacio de datos. Cada recurso tiene una unidad de ejecución propietaria.

- *Telescript* ofrece una variedad de mecanismos de seguridad integrados en el lenguaje. Es así que los objetos *thread* tienen atributos que describen la autoridad o dueño del mismo, los permisos y cuotas de recursos.

- Provee integridad y encriptación de mensajes y agentes.

- *Odyssey* corre en cualquier plataforma que soporte JDK 1.1.

- Está enteramente escrito en Java, y no requiere modificaciones de Java VM.

- Usa Remote *Method-Invocation (RMI)* de Java y mecanismos de reflexión, también usa Java AWT.

- *Odyssey* puede usar cualquier mecanismo de transporte confiable para mover un agente de un sistema de agentes a otro.

Además en *RMI* se tienen dos mecanismos de transporte, *Microsoft Distributed Component Object Model (DCOM)* y el Internet- *Orb Protocol (IIOP). RMI* es usado para dos diferentes propósitos en *Odyssey*. Es el mecanismo de transporte de los agentes (aunque puede ser reemplazado por *DCOM* o *IIOP*) y es también usado como un mecanismo *Finder*, para determinar en que lugar se localiza un sistema de agentes *Odyssey.*

1.1.3.5.2 Arquitectura Odyssey

La arquitectura incluye una división de módulos que favorece el comercio electrónico, así se incluyen: lugares, puntos de reunión, autoridades, agentes, conexiones y permisos que son necesarios para que un sistema de agentes móviles pueda resolver dicha tarea (ver figura 27).

La arquitectura Odyssey cuenta con:

Place: Plataformas hacia y desde donde se pueden mover los agentes son componentes encargados en manejar autorizaciones de agentes.

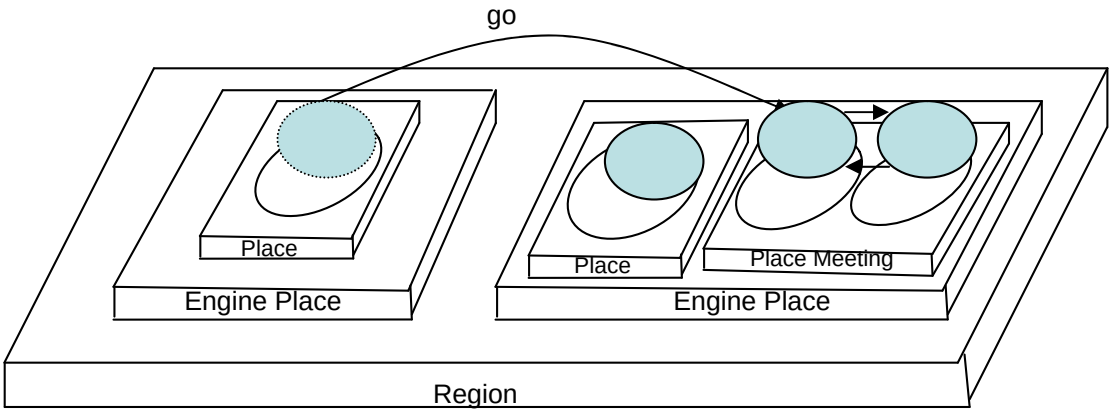
Engine place: es el componente que se encuentra en contacto directo con los places de los agentes y que soporta a los *place*.

Regions: estos componentes son los puntos de entrada a una red *Telescript*. Una región puede componerse de un número de motores que pueden ser ejecutados en diferentes *hosts*. Los *engine places* corren en las regiones.

Ticket: es un argumento que indica una posición de un *place* y el tiempo en el cuál un agente debe comenzar la migración.

Purgatory:: cuando un agente es rechazado de un *place* es enviado al purgatorio, en donde se alojan por un corto tiempo en un ambiente limitado.

Figura 27. Arquitectura Odyssey



Fuente:*Telescript technology* [37]

Meeting place: es un *place* en el cual dos agentes se encuentran para comunicarse. Este componente prioriza la comunicación entre agentes y la utilización de otros recursos.

Allowance: representa un conjunto de permisos de acceso otorgados por un *place* para restringir el acceso a sus recursos.

1.1.3.6 JATLite

El *framework* de la Universidad de *Stanford* (1997) para la construcción de sistemas multi-agente llamado *JATLite* está implementado en Java y provee un conjunto de paquetes Java[38].

En general el término "agente" en *JATLite* es usado para significar una comunidad de "entidades" que intercambian mensajes a fin de desarrollar una tarea (agentes reactivos). Esto significa que cada entidad coopera con las otras a fin de solucionar un problema, aún así *JATLite* no cuenta con estructuras definidas que le permitan representar el problema en su estado interno, aprender, o decidir que acciones llevar a cabo a fin de lograr autonomía.

1.1.3.6.1 Características JATLite

JATLite provee la funcionalidad básica para construir aplicaciones de sistemas multiagente.

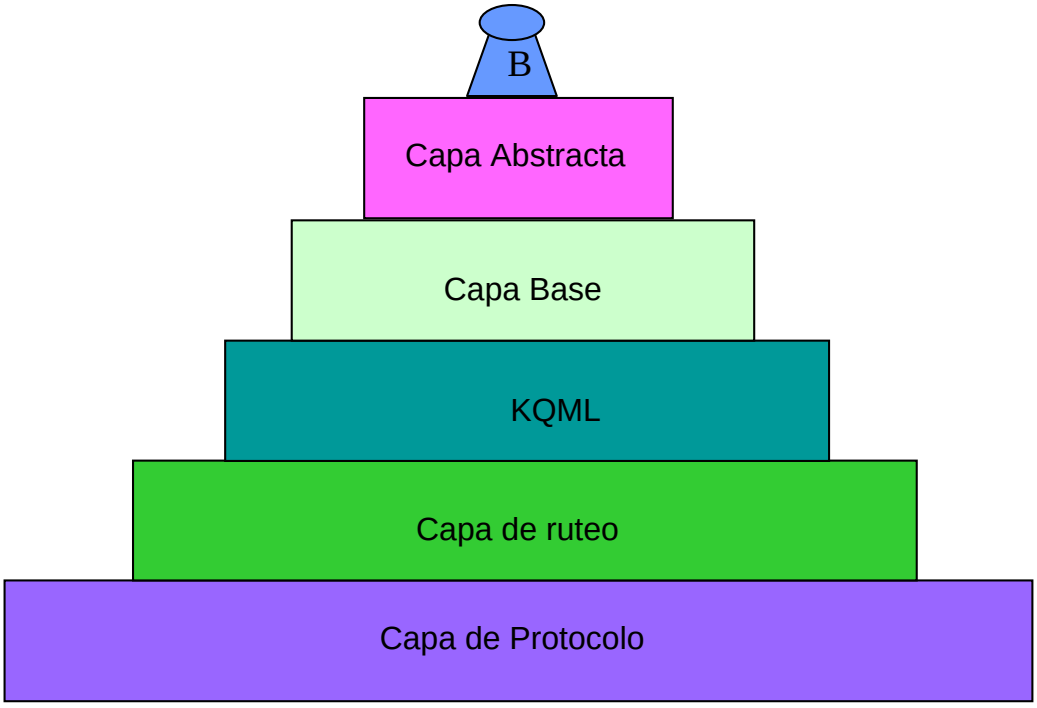
- Posee herramientas para la comunicación, a través del intercambio de mensajes *KQML* usando *TCP/IP*.
- Tiene un servidor de nombres de agentes, con la información de todos los nombres y direcciones de los agentes existentes.
- Permite la funcionalidad para que los *Applets* Java intercambien mensajes con cualquier agente registrado en Internet.
- *JATLite* no facilita el uso de herramientas de razonamiento ni establece ninguna estructura para autonomía y percepción.

1.1.3.6.2 Arquitectura JATLite

Existen cinco capas en la arquitectura de *JATLite*, de las cuales el desarrollador puede elegir la más apropiada para que a partir de ésta inicie la construcción de sus sistemas. Cada capa de un nivel superior impone nuevas restricciones a las aplicaciones de agentes. La arquitectura por niveles de *JATLite* está formada por los niveles (de mayor a menor nivel de abstracción). Las cinco capas, descritas a continuación, son[39] (Figura 28):

1. La capa abstracta es una colección de clases abstractas necesarias para la implementación de *JATLite*. Esta capa supone que las conexiones se hacen siguiendo el protocolo TCP/IP, aunque es posible extenderla para soportar otros protocolos tales como UDP.
2. La capa base provee comunicación elemental basándose en el protocolo TCP/IP y la clase abstracta. En esta capa no hay restricción en el lenguaje de los mensajes o el protocolo.

Figura 28. Arquitectura JATLite



Fuente: interpretado de la lectura de JATLite

3. La capa *KQML* hace el análisis gramatical de mensajes *KQML* y permite su almacenamiento. Esta capa también implementa las extensiones a *KQML* propuestas por el centro para la investigación de diseño de la Universidad de *Stanford* el cual provee un protocolo para registrarse, conectarse y desconectarse.

4. La capa de ruteo provee los servicios de registro por nombre, de ruteo y almacenamiento de mensajes *KQML*. Todos los agentes envían y reciben mensajes mediante el ruteador el cual los reenvía a los destinos nombrados. Cuando un agente intencionalmente se desconecta o accidentalmente termina, el ruteador almacena sus mensajes hasta que el agente se vuelve a conectar. El ruteador es particularmente importante para agentes que son *applets*, ya que éstos sólo pueden iniciar conexiones con el servidor que los creó debido restricciones de seguridad del WWW y Java.

5. La capa de protocolo la cual provee servicios tales como *SMTP (Simple Mail Transfer Protocol)* y *FTP (File Transfer Protocol),* aunque puede ser extendida para soportar otros protocolos. Esta capa es útil cuando los agentes requieren intercambiar archivos o enviar mensajes por correo electrónico.

1.1.3.7 COMPARATIVO

A continuación se presenta el comparativo soportado en las fuentes y referencias relacionadas por cada una de las tecnologías de agentes móviles.

Tabla 7. Comparativo plataformas de agentes móviles

PLATAFORMA	COMPAÑÍA	HERRAMIENTA / DESARROLLO	Sistema de Comunicación	Portabilidad	Seguridad	Operaciones
Aglets	IBM's (1996) Tokio	Java	Sincrono y asincronico. Protocolo ATP, mensajes	Todas las plataformas	Definición de autoridades, privilegios y preferencias	Creación, clonación, expedición, retractación, eliminación, activación, desactivación y paso de mensaje
Odyssey y/ó TeleScript	General Magic's Incorporated 1994, 1997	Telescript, Java	Sincrono, asincronico y multipunto. Independiente del protocolo de transporte. CORBA, RMI, IIOP, DCOM.	Unix, Windows sobre JDK	Administra el espacio de datos, encriptación de mensajes y permisos.	Creación de agentes y lugares, migración, eliminación, no permite persistencia.
Grasshopper	IKV++'s 1998 (Belin Alemania)	Plataforma basada en java	Sincrono, asincronico y multipunto. CORBA, RMI o sockets.	Todas las plataformas	Cada agencia mantiene servicios de seguridad interna y externa.	Creación, borrado, suspensión, reanudación, clonación, copia, migración, almacenamiento e invocación de acciones.
MOA	Mobile Object and Agents (The Open Group's)	Java	Sincrono (RPC-like) Asíncrono (one-way messages).	Todas las plataformas	Soportar control histórico de la actividad de los agentes.	suspensión, reanudación, eliminación, localización y monitorización.
Ara	Universidad Kaiserslauten Alemania (1997).	Tcl y C/C++ con MACE(Lenguaje Intermedio)	Asincrono, Protocolo TCP ó SMTP. Mensajes	Unix	Provee utilidades básicas de seguridad. Utiliza permisos	Interacción, acceso a la interfaz de usuario, al sistema de archivos o a otros programas y migración a nodos

					para todos los recursos.	remotos
JatLite	Stanford University 1997	Java	Sockets sobre TCP/IP, UDP, usa KQML	Todas las plataformas	Sistema Centralizado, posee servidor de nombres y direcciones, no utiliza razonamiento, no utiliza autonomía ni percepción. Permite almacenar mensajes.	Operaciones: Creación, Registro, conexión, desconexión, ruteador.

1.1.4 PLAN DE ORDENAMIENTO TERRITORIAL (POT)

Mediante el desarrollo del prototipo se desea integrar la tecnología informática en áreas de gestión administrativa con entidades nacionales y municipales para la consulta de información de forma automática referente al plan de ordenamiento territorial (P.O.T).

Desde el punto de vista del dominio del problema para analizar y enmarcar el desarrollo de la aplicación prototípica se han tenido en cuenta tres aspectos:

- El plan de ordenamiento territorial y los objetivos que debe cumplir.
- El modelo general para la gestión y ejecución del plan de ordenamiento territorial.
- Los Aspectos generales del plan de ordenamiento territorial.

1.1.4.1 DEFINICIÓN

Se define como el conjunto de objetivos, directrices, políticas, estrategias, metas, programas, actuaciones y normas adoptadas para orientar y administrar el desarrollo físico del territorio y la utilización del suelo [26].

Además es el instrumento de planeación y gestión del desarrollo territorial el cual incorpora las políticas nacionales, departamentales y los compromisos adquiridos por los mandatarios locales en su programa de gobierno. Estos planes y programas a su vez reflejan los intereses de la comunidad.

1.1.4.2 OBJETIVOS DEL PLAN DE ORDENAMIENTO TERRITORIAL

1. Armonizar y actualizar las disposiciones contenidas en la Ley 9a. de 1989 con las nuevas normas establecidas en la Constitución Política, la Ley Orgánica del Plan de Desarrollo, la Ley Orgánica de Áreas Metropolitanas y la Ley por la que se crea el Sistema Nacional Ambiental.

2. Establecer los mecanismos que permitan al municipio en ejercicio de su autonomía a promover el ordenamiento de su territorio a usar equitativa y racionalmente el suelo, a preservar y defender el

patrimonio ecológico y cultural localizado en su ámbito territorial y a prever desastres en asentamientos de alto riesgo, así como la ejecución de acciones urbanísticas eficientes.

3. Garantizar que la utilización del suelo por parte de sus propietarios se ajuste a la función social de la propiedad y permita hacer efectivos los derechos constitucionales a la vivienda y a los servicios públicos domiciliarios, y velar por la creación y la defensa del espacio público, así como por la protección del medio ambiente y la prevención de desastres.

4. Promover la armoniosa concurrencia de la Nación, las entidades territoriales, las autoridades ambientales y las instancias y autoridades administrativas y de planificación, para el cumplimiento de las obligaciones constitucionales y legales que prescriben al Estado en el ordenamiento del territorio y lograr el mejoramiento de la calidad de vida de sus habitantes.

5. Facilitar la ejecución de actuaciones urbanas integrales, en las cuales confluyan en forma coordinada la iniciativa, la organización y la gestión municipales con la política urbana nacional, así como con los esfuerzos y recursos de las entidades encargadas del desarrollo de dicha política.

1.1.4.3 TIPOS DE PLANES DE ORDENAMIENTO TERRITORIAL

Los tipos de planes de ordenamiento territorial según la ley 388 de 1997 (ver anexo A) son[26]:

1. Plan de ordenamiento territorial.
2. Plan básico de ordenamiento territorial.
3. Esquema de ordenamiento territorial.

Cada uno de ellos se utiliza en los municipios de acuerdo a la cantidad de habitantes y en ellos se establecen los principios básicos que se deben cumplir. El cuadro presenta sucintamente la diferencia entre estos tipos de planes de ordenamiento territorial (**Ver cuadro comparativo siguientes Páginas)**

1.1.4.4 MODELO GENERAL PARA LA GESTION DEL PLAN DE ORDENAMIENTO TERRITORIAL

En el modelo general para la gestión del plan de ordenamiento territorial, uno de los pilares son los **sistemas de información municipal** quienes dan el soporte técnico para la toma de decisiones. (ver esquema 1)

El modelo general pretende dar un marco de referencia que enmarca la consulta de información con los objetos de software móviles en el **sistema de información municipal**. El modelo define los pasos para construir el plan de ordenamiento territorial en un municipio.

Se definen cuatro etapas dentro del modelo:

1. **Etapa Inicial:** plantea las estrategias, políticas y objetivos para el desarrollo del P.O.T. teniendo en cuenta la legislación vigente. Se elabora un documento que contiene los términos de referencia, equipos técnicos y el plan operativo a seguir.

2. **Etapa de Diagnostico**: una vez elaborado el documento en la etapa inicial se desarrolla un análisis de caracterización, clasificación y especialización del sistema administrativo, sistema biofísico, sistema social, sistema económico y del sistema funcional, con ello se logra la evaluación integral del territorio.

   En estas dos estapas es primordial la participación ciudadana puesto que la comunidad es un instrumento clave para la formulación del POT

3. **Etapa de Prospectiva**: se diseñan los diferentes escenarios (tendenciales y alternativos) donde participan los diferentes estamentos de la comunidad. Esta etapa culmina con la propuesta del plan de ordenamiento territorial.

4. **Etapa de Instrumentación y Ejecución:** en donde se elaboran los artículos que contiene el plan de ordenamiento territorial para entrar a discutirlos y aprobarlos. Una vez aprobados quedan establecidos como normas que se deben cumplir en el municipio.

Las entidades administradoras y fiscalizadoras del municipio realizaran seguimiento, actualización, evaluación y control al Plan de Ordenamiento Territorial aprobado.

Esquema 1. Modelo General para la Gestión del P.O.T

Fuente: Cartilla IGAC [49] (Ver esquema siguiente página)

1.1.4.5 ASPECTOS DEL PLAN DE ORDENAMIENTO TERRITORIAL

Otro elemento que debe ser tenido en cuenta para el desarrollo de la aplicación, son los Componentes del Plan de Ordenamiento Territorial como muestra el Esquema No. 2[44].

Se definen cinco aspectos dentro del plan de Ordenamiento Territorial:

1. Aspecto Social: que debe tener en cuenta el bienestar de la comunidad, la salud, la seguridad, la educación y la vivienda.

2. Aspecto Cultural: contempla las festividades, costumbres, personajes, sitios turísticos, leyendas, grupos culturales, religiones, indígenas que la comunidad apropia como bien cultural, , así como las áreas de conservación y protección del patrimonio histórico, cultural y arquitectónico. De acuerdo a la ley 99 de 1993 (ver Anexo B).

3. Aspecto Ambiental: enmarca los componentes biofísicos, geológicos, hidrología y clima. que posee el municipio. Además señala las áreas de reserva, las medidas para la protección del medio ambiente, conservación de los recursos naturales, defensa del paisaje[27] y el tipo de suelo.

4. Aspecto Económico: son aquellos aspectos que le generan recursos económicos al municipio como son: la industria, el comercio, las empresas, los megaproyectos, los microproyectos y la infraestructura.

5. Aspecto General: contempla la localización del municipio, la superficie, el número de habitantes, el número de veredas, barrios y los límites de la población.

Esquema 2. Aspectos del Plan de Ordenamiento Territorial

Fuente: Herrera Luz Helena [44]. (Ver esquema siguiente página)

1.2. MARCO REFERENCIAL METODOLÓGICO

1.2.1. UML (Lenguaje Unificado de Modelado)

Metodología utilizada para la construcción del prototipo OSM-CONDABAPOT, UML (Unified Modelling Language) es un lenguaje visual para especificar, construir y comunicar artefactos de un sistema basado en software.

1.2.1.1. ORIGEN

El proceso unificado esta equilibrado por ser el producto final de tres décadas de desarrollo y uso practico, inicia con el proceso Objectory (1987) pasando por el proceso Objectory de Rational (1997) hasta el proceso unificado de Rational en 1998. Metodología influenciada y soportada con los productos de Ericsson y de Rational[47].

Método Ericsson:

Creado en 1967, este proceso se encarga de modelar el sistema como un conjunto de bloques interconectados para ensamblarlos posteriormente desde el más bajo hasta el más alto nivel.

Se identifican los bloques identificando los casos del negocio (en UML casos de uso), se relacionan los bloques de acuerdo a las responsabilidades. Las actividades de diseño producían un conjunto de diagramas de bloques estáticos con sus interfaces agrupando subsistemas.

Los diagramas de bloques corresponden hoy a una versión simplificada de diagramas de clases o paquetes en UML.

En esencia el método utilizado es el que se conoce hoy por desarrollo de componentes creado por Ivar Jacobson.

Método Rational:

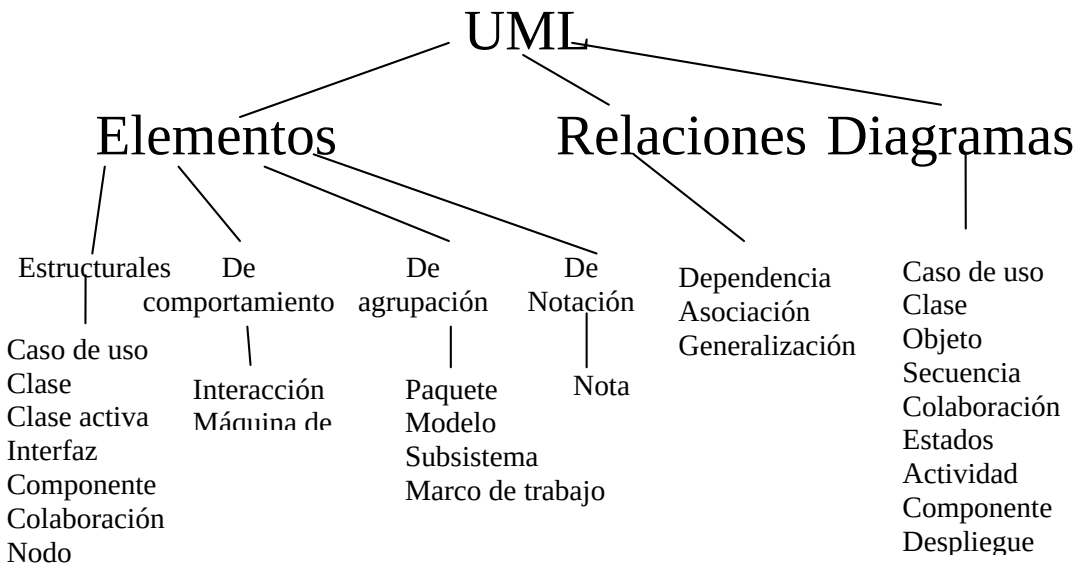
Creado en 1981. Rational software Coporation compró a Object AB a fines de 1995 para unificar los principios básicos subyacentes en los procesos de desarrollo.

Este proceso hace énfasis en la arquitectura y el desarrollo interactivo. La arquitectura contempla: la vista lógica, la vista de procesos, la vista física, la vista de desarrollo y una vista adicional que contempla las primeras cuatro vistas mediante casos de uso o escenarios. El desarrollo interactivo contenía cuatro fases (comienzo, elaboración, construcción y transición). Estos dos procesos creados por Grady Booch se encargan de estructurar y controlar el proceso durante las interacciones.

1.2.1.2. CARACTERISTICAS

UML cuenta con diferentes elementos (glosario) que se desarrollan durante las cuatro etapas de desarrollo del software (inicio, elaboración, construcción y transición), la metodología proporciona un vocabulario que incluye las categorías, elementos, relaciones y diagramas (ver Figura 29).

Figura 29. Elementos de UML



Fuente: El proceso Unificado de Desarrollo de Software[45]

UML modela orientado a objetos, las clases y objetos junto a sus relaciones son los principales elementos. El modelo de clases y objetos muestra el sistema que se trata de describir. Los casos de uso corresponden a una descripción de un conjunto de secuencias de acciones, incluyendo variaciones que un sistema lleva a cabo y que conduce a un resultado observable de interés para un actor determinado (ver glosario).

En general:

- UML esta basado en componentes.
- UML Interconecta componentes a través de interfaces bien definidas.
- UML representa los esquemas de software.
- UML esta dirigido por los casos de uso.
- UML se centra en la arquitectura
- UML es Interactivo e incremental.
- UML representa a los usuarios como algo que interactúa con el sistema.

1.2.1.3. UML EN EL PROTOTIPO OSM-CONDABAPOT

Las fases de UML constituyeron una herramienta de ingeniería fundamental para lograr el desarrollo del prototipo.

Cada módulo tiene una funcionalidad y una forma, “la funcionalidad son los casos de uso y la forma es la arquitectura”[48].

Durante la construcción el prototipo fue dirigido por los casos de uso, ya que se definieron las funcionalidades del sistema de acuerdo a los usuarios o actores.

El proceso fue centrado en la arquitectura ya que el sistema se diseño desde diferentes puntos de vista contemplando también aspectos estáticos y dinámicos y el proceso fue interactivo e incremental

Los casos de uso y la arquitectura son importante, porque: “Los arquitectos moldean el sistema para darle una forma. En esta forma la arquitectura es la que debe diseñarse para permitir que el sistema evolucione no solo en su desarrollo inicial, sino a lo largo de futuras generaciones. Para encontrar esa forma los arquitectos deben trabajar sobre la comprensión general de las funciones clave, es decir sobre los casos de uso claves del sistema. Estos casos de uso pueden ser el 5% o el 10% del total [48].

2. ANALISIS Y DISEÑO DEL PROTOTIPO “OSM-CONDABAPOT”

2.1. CONCEPCIÓN

2.1.1. Especificación Del Problema

Una entidad Central (ej: Planeación Nacional) podrá requerir un sistema automatizado que consulte información de un conjunto de localizaciones específicas (municipios) de una región, este sistema debe consultar información referente al Plan de Ordenamiento Territorial.

Los tipos de consulta que abordará el prototipo obedecen a un análisis de la información presentada referente al P.O.T.. Los tipos de consulta son:

- Datos generales de la localización específica
- Datos del Uso del Suelo de la localización específica
- Datos de Infraestructura de la localización específica.
- Datos de las entidades existentes en la localización específica.
- Datos del Clima de una región específica del Municipio

El solicitante haciendo uso del sistema seleccionará los sitios específicos y el tipo de consulta deseada para que los objetos de software móviles automáticamente viajen y regresen con la información solicitada.

Como solución se propone:

Un sistema que combine la plataforma de Agentes de Software Móviles (ASM) con Objetos de software Móviles (OSM) y Agentes Humanos para lograr la consulta de información remota de diferentes localizaciones específicas de una región.

El sistema se denominará “OSM-CONDABAPOT: Objetos de software móviles para la consulta de información en bases de datos tipo plan de ordenamiento territorial.”

Para satisfacer la consulta el prototipo del sistema simulará estar instalado en la entidad central (ej: planeación nacional) y en cada una de las localizaciones específicas (ej: municipios) que deseen comunicarse usando la red de redes.

A continuación se describe el sistema desde la perspectiva de cada uno de los sitios geográficos:

Entidad Central:

El administrador debe iniciar la plataforma de agentes móviles validando su entrada al sistema con "*login*" o registro de nombre de usuario y contraseña.

La entidad central tendrá los siguientes módulos, cada uno de ellos validará mediante *"login" y* contraseña la entrada al usuario.

- **Módulo de seguridad de acceso:** le permitirá al administrador registrar los datos de los usuarios (nombre, cargo, *login*, *password).*

- **Módulo de registro localizaciones específicas:** le permitirá al administrador registrar los datos de las localizaciones específicas (País, región, nombre localización específica, dirección IP) para que el solicitante las pueda seleccionar antes de lanzar un objeto de software móvil tipo consulta P.O.T..

- **Módulo de recolección de datos:**

  - **Submódulo definir itinerario y tipo de consulta:** le permitirá al solicitante definir las localizaciones específicas de viaje y de regreso, además permite definir el tipo de consulta a ser realizada.

  - .**Submódulo lanzar objetos de software móviles a localizaciones específicas:** lanzará los objetos tipo consulta P.O.T. a cada una de las localizaciones específicas predefinidas.

    La respuesta a la solicitud depende de:

    El éxito de búsqueda en la localización específica.
    La existencia del lugar.
    El tráfico de información sobre la red o

El establecimiento de la comunicación entre la localización específica y la entidad central.

En el momento de llegar el objeto de software móvil a la localización específica se registrará la fecha y hora de llegada del objeto. Cuando el objeto obtenga la información de la consulta solicitada en las localizaciones específicas, éste retornará a la entidad central en donde se registrará la consulta en una base de datos histórica.

- **Módulo Visualización de consultas:** le permitirá al solicitante visualizar las consultas realizadas en las localizaciones específicas.

Localización Específica:

En la localización específica el administrador debe iniciar la plataforma de agentes móviles validando su entrada al sistema con "*login*" o registro de nombre de usuario y contraseña.

A continuación la plataforma estará apta para recibir objetos de software móviles. En el momento de llegar el objeto de software móvil a la localización específica se registrará el tipo de consulta, la fecha y hora de llegada del objeto, posteriormente los objetos consultarán de manera automática la base de datos tipo plan de ordenamiento territorial y a medida que se vayan encontrando datos se cargará la información en un contenedor. Cuando termine el cargado de datos se transmitirá automática y autónomamente el objeto a la entidad central junto con la consulta.

El éxito de la consulta depende de:

- El éxito de búsqueda en la localización específica,
- La existencia del lugar,
- El tráfico de información sobre la red o
- El establecimiento de la comunicación entre la localización específica y la entidad central.

De otro lado, durante la transmisión y la recepción de la consulta el canal de comunicaciones permanecerá disponible para cualquier otro tipo de comunicación.

Los procesos generales que debe realizar el sistema son:

- Despachar objetos de software móviles especializados (consultas ya mencionadas) a localizaciones específicas.

- Registrar en la base de datos de la localización específica la llegada del OSM.

- Consultar la base de datos tipo P.O.T. y cargar los datos en un contenedor.

- Transmitir a la entidad central el OSM con el resultado de la consulta.

- Registrar la consulta realizada por el OSM en la base de datos de la entidad central.

Los tipos de consulta son aproximaciones a la realidad de lo que podría manipularse desde la entidad central y la(s) localización(es) específica(s) referentes al P.O.T. en Colombia.

2.1.2. Distribución de Funcionalidades

La Figura 30 muestra dos nodos: la entidad central y la localización específica. Cada nodo contiene el software necesario del sistema "OSM-CONDABAPOT".

Figura 30. Distribución de Funcionalidades

2.2. MODELO FUNCIONAL Y DE COMPORTAMIENTO DEL SISTEMA

2.2.1. Actores

Todos los roles encontrados en la especificación del sistema son candidatos a ser actores, ellos son:

El Administrador: tendrá a cargo las siguientes responsabilidades:

- Iniciar la plataforma de agentes de software móviles en la entidad central y en las localizaciones específicas.
- Registra los usuarios que pueden acceder al sistema.
- Registra los datos de todas las localizaciones específicas y sus direcciones IP a donde pueden viajar y regresar los objetos de software móviles.

El Solicitante: tendrá a cargo las siguientes responsabilidades:

- Define por cada localización específica el itinerario de viaje, el tipo de consulta a realizar y el sitio de regreso.
- Lanza los objetos de software móviles tipo consulta P.O.T. a las localizaciones específicas.
- Consulta en la entidad central los datos referentes al P.O.T. de cada una de las localizaciones específicas que han visitado los OSM.

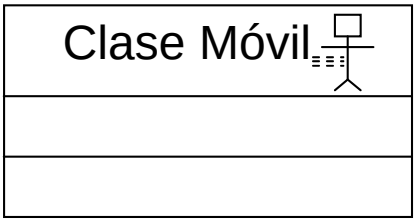
2.2.2. Estereotipos

En el desarrollo del prototipo OSM-CONDABAPOT se construyeron dos estereotipos, los cuales representan aspectos propios del sistema como la móvilidad.

Los estereotipos son:

Clase móvil: Corresponde a una clase que instancia objetos de software móviles (ver Figura 31).

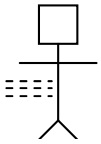
Figura 31. Estereotipo Clase Móvil



Estereotipo necesario para identificar las clases móviles de las clases estáticas.

Objeto de software móvil: Corresponde a un objeto (software) que se desplaza a través de diferentes nodos de una red (ver Figura 32).

Figura 32. Estereotipo Objeto de Software Móvil



Estereotipo necesario para identificar en los diagramas UML la movilidad de objetos.

2.2.3. Casos de uso

Codificación de los casos de uso del sistema OSM-CONDABAPOT, ver Tabla 8, en donde se presenta el nombre del caso de uso, el módulo al cual pertenece y el código asignado a él.

Tabla 8. Codificación Casos de Uso

#	Nombre Caso de Uso	Módulo	Actor	Codificación
1	Validar *login* y *password*	Todos los módulos	S - A	cu1
2	Registrar usuario	Seguridad de Acceso	A	cuA1
3	Seleccionar módulo	Seguridad de Acceso	A	cuA1.1
4	Incluir Locallzación específica	Registro Localizaciones Específicas	A	cuA2
5	Registrar País	Registro Localizaciones Específicas	A	cuA2.1
6	Registrar Región	Registro Localizaciones Específicas	A	cuA2.2
7	Registrar localización específica	Registro Localizaciones Específicas	A	cuA2.3
8	Definir itinerario de viaje	Recolección de Datos Submodulo definir itinerario de viaje y tipo de consulta	S	cuS1
9	Seleccionar Tipo de Consulta	Recolección de Datos Submodulo definir itinerario de viaje y tipo de consulta	S	cuS1.1

10	Seleccionar Localización Específica	Recolección de Datos Submódulo definir itinerario de viaje y tipo de consulta	S	cuS1.2
11	Seleccionar Regreso	Recolección de Datos Submódulo definir itinerario de viaje y tipo de consulta	S	cuS1.3
12	Lanzar OSM a localización Específica	Recolección de Datos Submódulo lanzar OSM a localización específica	S	cuS2
13	Cargar Direcciones IP	Recolección de Datos Submódulo lanzar OSM a localización específica	S	cuS2.1
14	Carga Datos consulta tipo P.O.T.	Recolección de Datos Submódulo lanzar OSM a localización específica	S	cuS2.2
15	Registra Datos Base Datos DBHisVis	Recolección de Datos Submódulo lanzar OSM a localización específica	S	cuS2.3
16	Consultar Datos Localización Específica	Consultar Datos Localización Específica	S	cuS3
17	Consultar uso del suelo	Consultar Datos Localización Específica	S	cuS3.1
18	Consultar infraestructura	Consultar Datos Localización Específica	S	cuS3.2
19	Consultar datos generales	Consultar Datos Localización Específica	S	CuS3.3
20	Consultar entidades	Consultar Datos Localización Específica	S	cuS3.4
21	Consultar clima	Consultar Datos Localización Específica	S	cuS3.5

Los símbolos para la tabla 8 corresponden a:

Los actores:
Administrador (A)
Solicitante (S).

La codificación general es: **cu[A]#1.[#2.]** en donde:

cu **:** casos de uso
[A] **:** es opcional la inicial del Actor
#1 **:** primer nivel
[#2.] **:** es opcional segundo nivel.

Cuando en los diagramas UML se presente el símbolo XXXXXX indica que corresponde a un tipo de consulta definida por el sistema las cuales corresponden a:

- **DatosGeneralesLocalizacionEspecifica:** Correspondientes a datos generales localización especifica.
- **DatosInfraestructuraLocalizacionEspecifica:** Correspondientes a infraestructura de la localización específica
- **DatosUsoSueloLocalizacionEspecifica:** Correspondientes a Uso del Suelo de la localización específica
- **DatosEntidadLocalizacionEspecifica:** Correspondientes a Entidades de la localización específica
- **DatosClimaLocalizacionEspecifica:** Correspondientes al clima de la localización específica

2.2.3.1. Diagrama Arquitectónico Funcional para los Actores Administrador y Solicitante

Se presenta en la figura 33 el diagrama arquitectónico para los actores administrador y solicitante. Cada uno de ellos tiene asociado los casos de uso correspondientes y los diagramas de secuencia.

Los Diagramas de Secuencia para cada caso de uso se presentan en el Anexo D.

Figura 33. Diagrama Arquitectónico Funcional para los Actores Administrador y Solicitante

2.3. MODELO ESTRUCTURAL

2.3.1. Modelo de persistencia

Dos modelos entidad relación se elaboraron.

2.3.1.1. Modelo entidad relación Localización Específica

Modela el manejo de los datos para el Plan de Ordenamiento Territorial (POT) en los municipios. (ver Figura 34).

Son las abstracciones de los datos para los tipos de consulta de los casos de uso presentados anteriormente.

Este modelo cuenta con una tabla (RegistroHistoricoConsulta) para registrar los tipos de consulta (nombre, fecha y hora) realizados por los OSM y que fueron lanzados desde la entidad central.

Las abstracciones del Plan del Ordenamiento Territorial están almacenadas en el modulo de persistencia DBTPOT.

En el ANEXO D (Diccionario de datos) se describen detalladamente las características de cada una de las entidades.

Figura 34. Modelo Entidad Relación Localización Específica (DBTPOT)

2.3.1.2. Modelo entidad relación Entidad Central:

Modelo similar al de la localización específica ya que se utiliza para almacenar todos los tipos de consulta que se realizan a las localizaciones específicas (ver Figura 35).

Tiene en cuenta una tabla (RegistroHistoricoConsulta) para registrar los tipos de consulta (nombre, fecha y hora) realizados por los OSM remotamente a la base de datos de la localización específica.

Todos los tipos de consulta tienen un secuencial asociado a la tabla RegistroHistoricoConsulta para identificar la hora, fecha y tipo de consulta de llegada del OSM desde la localización específica.

Las abstracciones están almacenadas en el modulo de persistencia DBHisVis.

El modelo que almacena los permisos de los usuarios para tener acceso al sistema “OSM-DONDABAPOT” se presenta en la Figura 36.

El modelo que almacena el itinerario de viaje y el sitio de regreso para los OSM que lo define el solicitante se presenta en la Figura 36.

Estas abstracciones son almacenadas en el modulo de persistencia DBAdmin.

En el ANEXO D (Diccionario de datos) se describen detalladamente las características de cada una de las entidades.

Figura 35. Modelo Entidad Relación Entidad Central (DBHisVis)

Figura 36. Modelo Entidad Relación Entidad Central (DBAdmin)

2.3.2. Clases del Dominio del Problema

En la Figura 37 se presenta el diagrama de clases del dominio del problema y la instanciación de objetos para el Módulo Lanzar OSM-a Localización Específica

La instanciación de objetos se muestra por cada una de las consultas realizadas por el sistema.

Esta instanciación representa los objetos de tipo diccionario que se deben despachar desde la entidad central a la localización específica.

Una vez el OSM ubicado en la localización específica se inicia el proceso de cargado de datos. Por cada tabla consultada de la base de datos DBTPOT se carga un objeto tipo diccionario.

Cuando finalice el cargado de datos (todas las tablas que involucra cada consulta) el OSM se despacha a la entidad central junto con todos los objetos tipo diccionario. Estando el OSM en la entidad central cada objeto tipo diccionario es descargado en cada una de las tablas de la base de datos DBHisVis.

En el ANEXO D (Diccionario de clases) se describen detalladamente las características de cada una de las clases que constituyen el prototipo.

Figura 37. Diagrama de Clases del Dominio del Problema

2.4. INTERFAZ GRAFICA

2.4.1. DIGRAMA DE NAVEGACIÓN

2.4.1.1. Diagrama de Navegación

El sistema prototipo OSM-CONDABAPOT tiene el siguiente diagrama de navegación (Ver Figura 38).

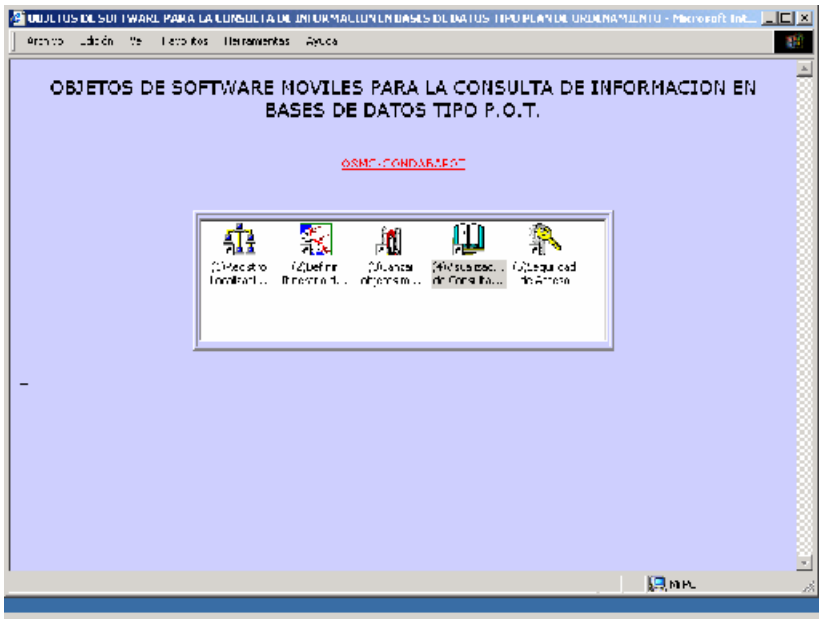
Figura 38. Diagrama de Navegación Sistema OSM-CONDABAPOT

2.4.2. DIAGRAMA DE LAYOUTS DE INTERFAZ GRAFICA

Cuando el usuario hace doble *click* sobre el icono OSM-CONDABAPOT (Ver Figura 39) que esta sobre el escritorio se despliega la interfaz que contiene todos los módulos del sistema (Ver Figura 40).

Figura 39. Icono del Sistema

Figura 40. Ventana Opciones del Sistema OSM-CONDABAPOT

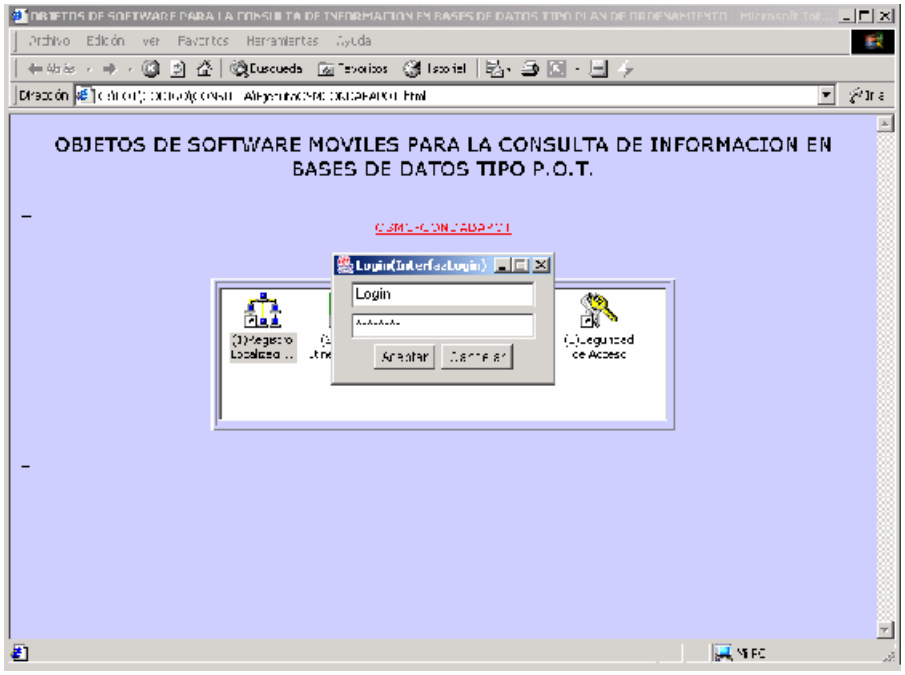


El actor puede seleccionar cualquiera de los módulos del sistema haciendo doble *click* sobre uno de ellos.

2.4.2.1. Ventana Validación de Acceso

En la ejecución de todos los módulos se despliega la interfaz de *Login* que valida la entrada al sistema mediante nombre de usuario y contraseña (Ver Figura 41).

Figura 41. Ventana Validación Acceso a Módulo

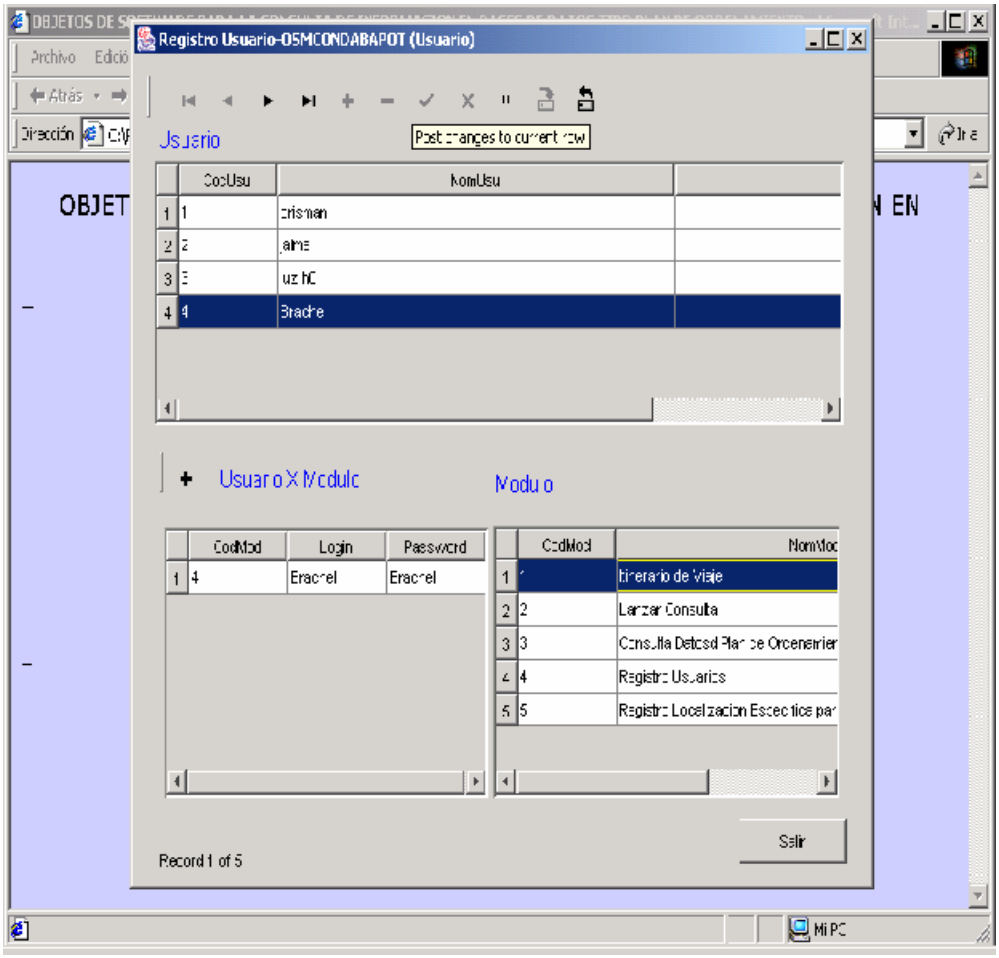


El Administrador ó solicitante debe digitar el *Login* y el *password* para entrar a un módulo del sistema. Posteriormente debe hacer *click* en el botón Aceptar.

2.4.2.2. Ventana Seguridad de Acceso

Cuando el actor Administrador ingrese al módulo seguridad de acceso aparecerá la interfaz que le permite registrar los datos del usuario y los datos de seguridad para cada módulo del sistema junto con el *Login* y el *password* (Ver Figura 42):

Figura 42. Ventana Seguridad de Acceso

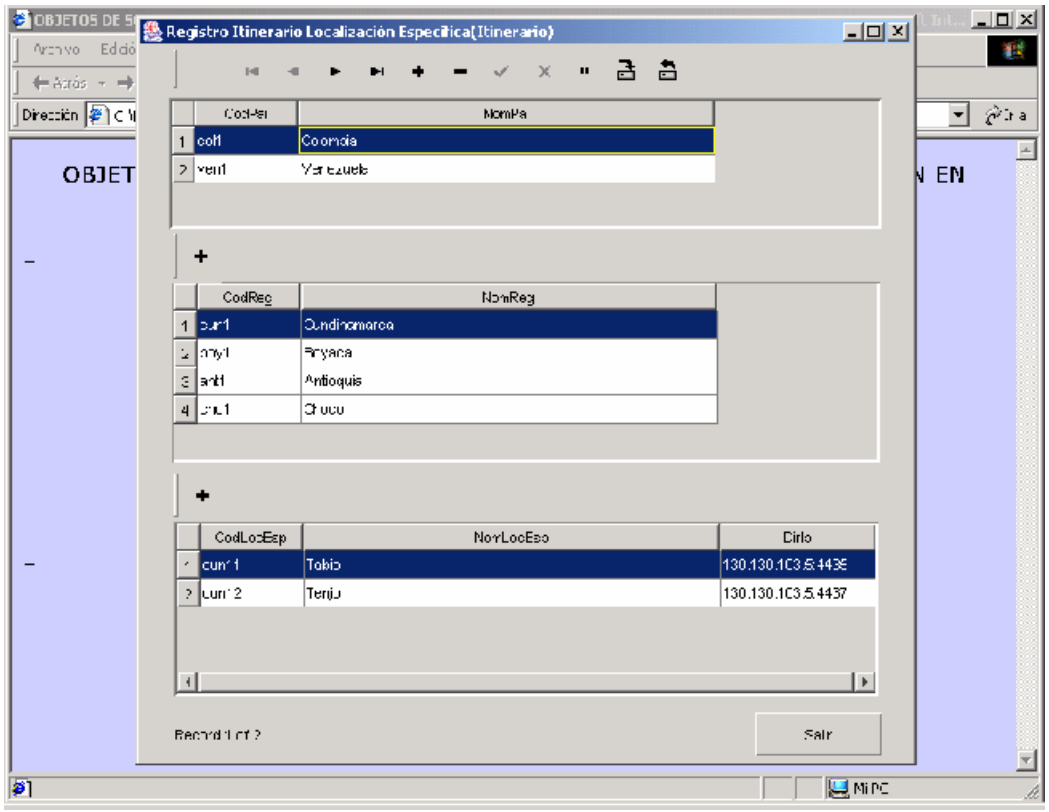


El Administrador debe hacer *click* en el botón adicionar de la barra de herramientas de la forma usuario para registrar los datos, dará *click* en salvar cambios. Posteriormente el administrador hará *click* en el botón adicionar de la barra de herramientas de la forma Usuario X Modulo para registrar *Login* y *password* de un determinado módulo y dará *click* en salvar cambios.

2.4.2.3. Ventana Registro Localización Específica

En el momento de ingresar el administrador al módulo **Registro Localización Específica** el sistema le permite registrar los datos de un país, las regiones pertenecientes a ese país y las Localizaciones específicas de cada región. Aquí se registra la dirección IP de la localización Específica donde se encuentra configurada la plataforma de agentes (Ver Figura 43).

Figura 43. Registro Localización Específica

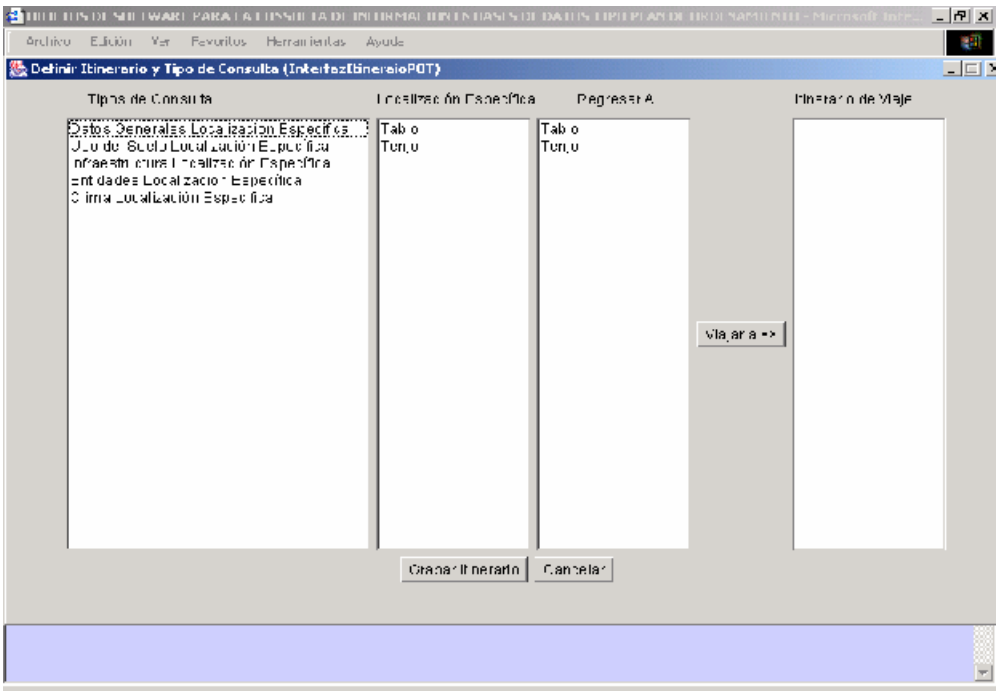


El usuario administrador podrá adicionar, modificar, eliminar y actualizar los datos de un País, de las Regiones o de las Localizaciones Específicas.

2.4.2.4. Ventana Definir Itinerario y Tipo de Consulta

Cuando el actor solicitante ingrese al submódulo Definir Itinerario y Tipo de Consulta el sistema automáticamente cargará los tipos de consulta y todas las localizaciones específicas que haya registrado el administrador en el módulo registro localización Específica (Ver Figura 44).

Figura 44. Ventana Definir Itinerario y Tipo de Consulta

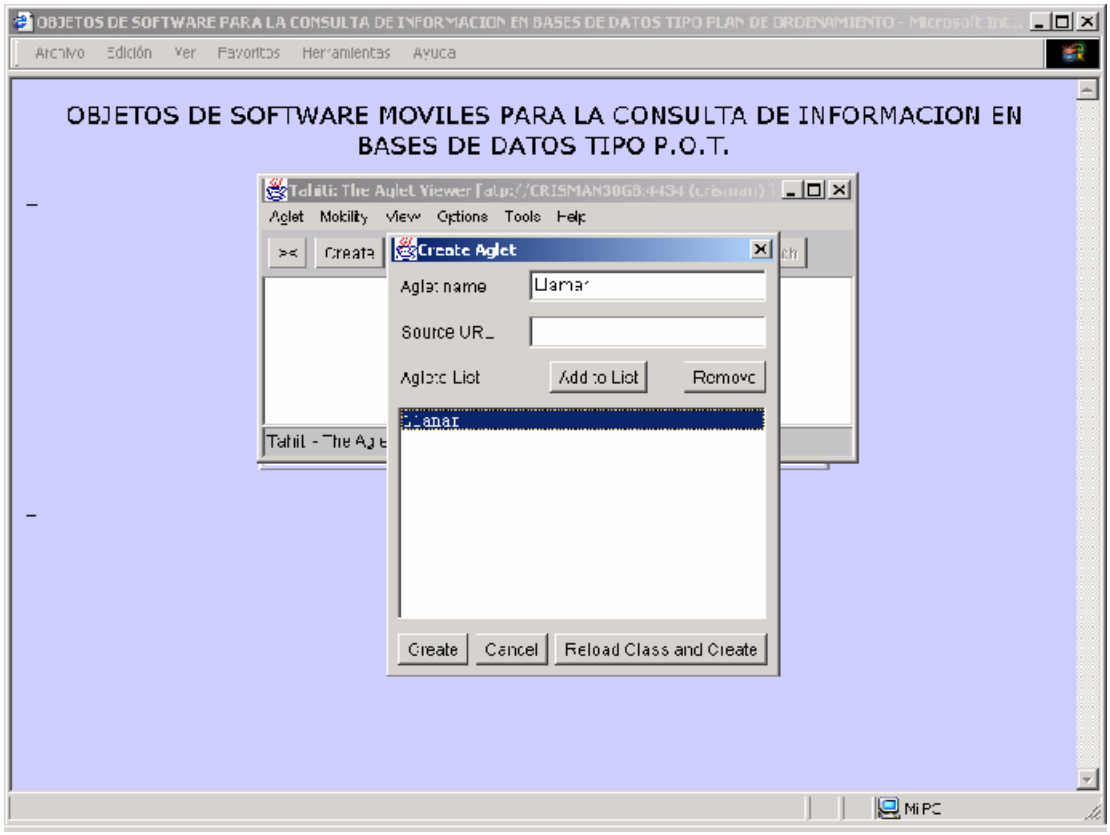


El solicitante podrá elegir el tipo de consulta que desea realizar en una determinada localización específica o diferentes localizaciones específicas. Debe elegir un tipo de consulta, una localización específica y el sitio de regreso del OSM, posteriormente debe hacer *click* en el botón **viajar a >>.** Por último debe hacer *click* en el botón **Grabar Itinerario.**

2.4.2.5. Ventana Lanzar OSM a Localización Específica

Le permite al solicitante Lanzar los OSM a las localizaciones específicas de acuerdo a lo seleccionado en el submódulo Definir Itinerario y Tipo de Consulta (Ver Figura 45).

Figura 45. Lanzar OSM a Localización Específica



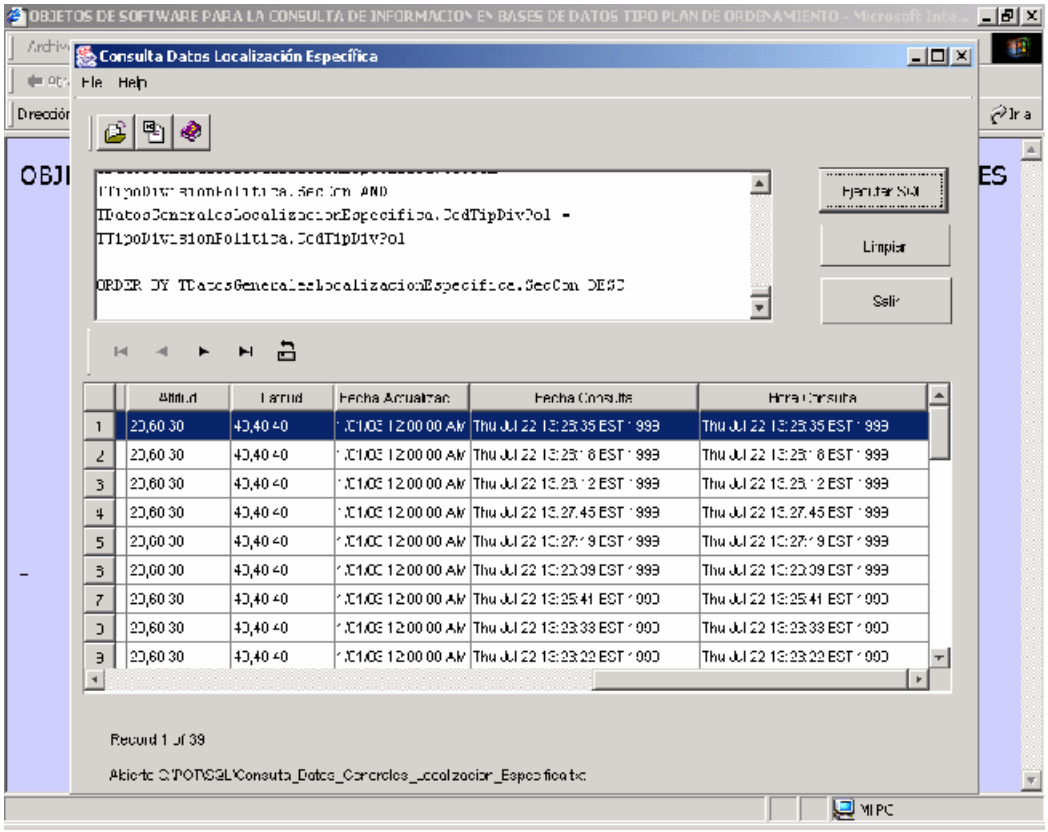
Los OSM viajarán a las localizaciones específicas y regresarán con la información tipo Plan de Ordenamiento Territorial para que posteriormente se registre automáticamente en la base de datos Histórica de la entidad central.

El solicitante debe hacer *click* en el botón ***Create*** y seleccionar el OSM **Llamar** para que la plataforma de agentes despache a las localizaciones específicas los Objetos de Software Móviles. (Se debe esperar hasta que los OSM arriben a la entidad central). Por último hará *click* en el botón ***Aglet*** y seleccionara **Exit**.

2.4.2.6. Ventana Visualizar Consultas POT

Cuando el solicitante escoja este módulo, le permitirá elegir un tipo de consulta que deseé visualizar. Se visualizarán todas las consultas realizadas del tipo de consulta seleccionada (Ver Figura 46).

Figura 46. Visualizar Consultas POT



El solicitante debe hacer *click* en el botón **Abrir Archivo** seleccionar un tipo de consulta definida en el sistema directorio **POT/SQL**, hacer click en el botón **Abrir** y posteriormente hacer *click* en el botón **Ejecutar SQL.** El solicitante también puede introducir directamente sentencias SQL en el área de Texto para consultar una o varias tablas de la Base de datos DBHisVis.

2.4.3. CLASES UTILIZADAS EN LA INTERFAZ GRAFICA

Las clases de la Figura 47 son las utilizadas en la construcción de la interfaz grafica para todos los módulos del sistema OSM-CONDABAPOT. En el anexo E se presenta el diagrama de paquetes para cada uno de los módulos gráficos del sistema.

Figura 47. Clases de la Interfaz Gráfica

3. ASPECTOS DE IMPLEMENTACION

3.1. ARQUITECTURA

El sistema de agentes de software móviles “OSM-CONDABAPOT” puede estar conformado por “Un conjunto de computadores autónomos unidos por una red de comunicaciones y equipados con software de sistemas distribuidos[35]”. Para el prototipo: la plataforma *aglets* soporta las comunicaciones entre entidad central y localización(es) específica(s).

La arquitectura que integra el prototipo OSM-CONDABAPOT se representa en el nodo de la Entidad Central y en el nodo de la Localización Específica (Ver Figura 48).

Figura 48. Arquitectura del Sistema “OSM-CONDABAPOT”

3.2. HERRAMIENTAS Y VERSIONES

3.2.1. Aglets

Aglets Software Development Kit (ASDK) de IBM corporation. Es una herramienta para desarrollar aplicaciones de agentes móviles escritos en código java.

ASDK esta disponible para los sistemas operativos SPARC/Solaris 2.5, x86/Solaris, Windows 95/NT, AIX 4.1.4, y OS/2, Unix, entre otros.

ASDK (*kit:* conjunto de clases) como herramienta de desarrollo es necesaria (junto con JDK) para compilar las clases construidas en java que heredan de *Aglet*.

ASDK (*kit: Tahiti*) como plataforma es necesaria para lanzar y recibir los objetos de software móviles. Plataforma presente en la entidad central y en las localizaciones específicas.

La versión libre [50] utilizada en el prototipo es la ASDK versión 1.1 Beta 3.

Limitaciones de la versión de *aglets*: Por ser una versión libre, la fecha configurada en el sistema debe ser anterior al año 2001.

3.2.2. JDK

Java development Kit (JDK) de sun Microsystem Inc. "es una herramienta de desarrollo para construir programas seguros, portátiles, robustos, orientados a objetos, multihilo e interactivos"[40].

JDK esta disponible para los sistemas operativos SPARC/Solaris 2.5, x86/Solaris, Windows 95/NT, AIX 4.1.4, y OS/2, Unix, entre otros.

JDK como herramienta de desarrollo es necesaria para compilar el código Java de los módulos del sistema "OSM-CONDABAPOT". Además es necesario JDK y ASDK para compilar las clases java que heredan de *Aglets.*

La versión libre [51] utilizada en el prototipo es la JDK versión 1.1.8.

3.2.3. Access

Access de la compañía *Microsoft Corporation*, es un sistema de administración de datos a través de bases de datos.

Con *Access* se implanto tecnológicamente los modelos relacionales de la Entidad Central y la Localización Específica.

La versión preinstalada [52] utilizada en el prototipo es *Access* 97.

3.2.4. Jbuilder

JBuilder de la compañía *Borland Software Corporation* es una herramienta para construir aplicaciones visuales en java con acceso a base de datos.

JBuilder proporciona acceso directo a los datos mediante la API JDBC de *sun*, además proporciona componentes *DataExpress* adicionales que simplifican en gran medida el desarrollo de aplicaciones de bases de datos.

La versión para pruebas [53] utilizada en el prototipo es *JBuilder Enterprise* 8.0.140.0.

3.2.5. ODBC

Es una interfaz de programación que permite a las aplicaciones tener acceso a datos en sistemas de administración de bases de datos que utilizan SQL como estándar.

El *ODBC* lo provee cada fabricante de bases de datos. En este caso es *Microsoft Corporation.*

La versión preinstalada [54] utilizada en el prototipo es *ODBC Desktop Database Drivers 4.0*

3.2.6. Explorer

Explorer es un navegador para la *web* desarrollado por *Microsoft Corporation.*

La versión preinstalada utilizada en el prototipo es *Internet Explorer* 5.00.3315.1000

3.3.7. Uso de Herramientas

La Tabla 9 muestra el uso de las herramientas en el sistema prototipo “OSM-CONDABAPOT”.

Tabla 9. Uso de Herramientas

Herramienta	Uso en OSM-CONDABAPOT
ASDK 1.1b3 y JDK1.1.8	Compilar el código Java con funcionalidad móvil
ASDK 1.1b3 (Tahiti)	Lanzar y recibir los objetos de software móviles
Access 97	Construir las bases de datos DBHisVis, DBAdmin y DBTPOT.
JBuilder 8.0.140.0	Construir el entorno visual (Interfaz Gráfica) del sistema que gestiona los datos de las bases de datos.
ODBC – JDBC	Permitir a las aplicaciones el acceso a los datos del sistema manejador de bases de datos.
Internet Explorer 5.0	Presentar al usuario los módulo del sistema “OSM-CONDABAPOT”

3.4. PAQUETIZACION

La paquetización corresponde a un mecanismo para organizar elementos en un grupo. La organización por paquetes para el sistema “OSM-CONDABAPOT” se presenta en el anexo E.

3.5. REQUISITOS DE HARDWARE Y SOFTWARE

La configuración de hardware y software requerida para el sistema "OSM-CONDABAPOT" es la siguiente:

- Computador *Pentium* III
- Tarjeta de Red
- 64 MB RAM
- Disco duro de 70 MB de espacio libre
- Monitor SVGA
- Mouse o puntero compatible con *Windows*
- Teclado estándar 101/102 teclas.
- Unidad *CD-ROM*
- Sistema operativo *Windows* 98
- ASDK versión 1.1b3
- JDK 1.1.8
- *Internet Explorer* 5.0
- *ODBC Desktop Database Drivers* 4.0
- El sistema OSM-CONDABAPOT para la Entidad Central y para la Localización Específica)

Esta configuración debe corresponder para el nodo en la Entidad Central y el (los) nodo(s) de la (s) Localización(es) Específica(s).

3.6. INSTALACION

3.6.1. Instalación JDK Entidad Central

1. Instale JDK 1.1.8 en el directorio ***c:\jdk1.1.8***

2. Configure las variable de entorno en el archivo *autoexec.bat*:

```
SET PATH=%PATH%;C:\jdk1.1.8
SET PATH=%PATH%;C:\jdk1.1.8\bin
SET JDK=c:\jdk1.1.8
SET JDK_HOME=c:\jdk1.1.8
```

3. Las librerías dinámicas: MSVCRT20.DLL y MFC30.DLL deben ser copiadas en el directorio ***c:\jdk1.1.8\bin***

4. Para hacer una prueba de la instalación de JDK digite el siguiente programa :

```
// Aplicación HolaMundo de ejemplo
class HolaMundoApp
{
        public static void main( String args[] )
        {
        System.out.println( "agosto: Hola Mundo!" ) ;
        }
}
```

5. Guárdelo como *HolaMundoApp.java* en el directorio ***c:\prueba***

6. Reinicie el sistema.

7. Compile el programa desde el directorio ***c:\prueba*** con el comando:

   Javac HolaMundoApp.java

8. El sistema debe generar en el directorio ***c:\prueba*** el archivo:

   HolaMundoApp.class

9. Ejecute el programa desde el directorio ***c:\prueba*** con el comando:

   Java HolaMundoApp

10. El sistema debe mostrar en la consola: "***agosto: Hola Mundo!"***

3.6.2. Instalación JDK Localización Específica

Realizar el mismo procedimiento que en la Entidad Central.

3.6.3. Instalación de ASDK Entidad Central

1. Instale *Aglets1.1b3* en el directorio **c:\aglets1.1.b3**

2. Configure las variables de entorno en el archivo *autoexec.bat*:

```
SET ASDK=C:\aglets1.1b3
SET AGLET_PATH=%ASDK%\public
SET AGLET_EXPORT_PATH=%AGLET_PATH%
SET AGLET_HOME=C:\aglets1.1b3
SET PATH=%AGLET_HOME%\bin;%PATH%
SET HOME=C:\
```

3. Si anteriormente estaba instalado *aglets* debe eliminar el archivo *tahiti.properties* para configurar el *login* y el *password* de la plataforma

4. Para hacer una prueba de la instalación de ASDK digite el siguiente programa :

```
// Aplicación MiAglet de ejemplo
package trial;
import com.ibm.aglet.*;

    public class MiAglet extends Aglet
        {
        public void run()
        {
        System.out.println("Mi aglets. Hola Mundo!");
        }
    }
```

5. Guárdelo como *MiAglet.java* en el directorio **c:\prueba**

6. Reinicie el sistema

7. Compile el programa desde el directorio ***c:\prueba*** con el comando:

javac -d c:\Aglets1.1b3\public -classpath c:\jdk1.1.8\lib\classes.zip;c:\Aglets1.1b3\lib\aglets.jar c:\prueba\MiAglet.java

8. El sistema debe generar en el directorio ***c:\aglets1.1b3\trial*** el archivo:

MiAglet.class

9. Verifique que el archivo *my_aglets.props* en el directorio ***c:\aglets1.1b3\bin*** tenga el siguiente contenido:

aglets.home=c:\\aglets1.1b3
maf.finder.host=localhost
maf.finder.port=4434
maf.finder.name=MAFFinder

10. Inicie *Tahiti* desde el directorio ***c:\aglets1.1b3\bin*** con el comando:

agletsd -f my_aglets.props

11. Registre el nombre del *aglet* desde la plataforma *tahiti con el* botón *Create y* digite en la casilla *aglet name* ***trial.MiAglet*** oprima el Botón ***add to List*** y después seleccione en el ***pane*** el agente ***trial.MiAglet*** para posteriormente oprimir el botón ***create.***

12. El sistema debe mostrar en la consola: "*Mi aglets. Hola Mundo!***"**

13. Salga de la plataforma *aglets* con el botón ***Aglet*** -> ***Exit*** (después de que regresen los OSM de las localizaciones específicas).

3.6.4. Instalación de ASDK Localización Específica

Realizar el mismo procedimiento que en la Entidad Central a excepción de los pasos No. 9 y No. 13.

Paso 9. Verifique que el archivo *my_aglets.props* en el directorio ***c:\aglets1.1b3\bin*** tenga el siguiente contenido:

aglets.home=c:\\aglets1.1b3
maf.finder.host=localhost
maf.finder.port=4435
maf.finder.name=MAFFinder

Paso 13. No lo haga.

3.6.5. Instalación sistema OSM-CONDABAPOT Entidad Central

1. Cree el directorio ***c:\POT*** en su sistema.

2. Descomprima el contenido del archivo **OSM-CONDABAPOTENTIDADCENTRAL.ZIP** en el directorio ***c:\POT*** (archivo existente en el CD del sistema OSM-CONDABAPOT).

3. Cree desde su escritorio un acceso directo a:
   C:\POT\CODIGO\CONSULTA\EjecutaOSMCONDABAPOT.html

4. Cambie el icono y seleccione uno apropiado (lo puede elegir del archivo 400icons.dll en el directorio ***c:\POT\CODIGO\CONSULTA\iconos).***

Los archivos instalados en la entidad central se relacionan en el archivo ***LeerEntidadCental.txt*** que esta en el CD de instalación del sistema “OSM-CONDABAPOT”

3.6.6. Instalación sistema OSM-CONDABAPOT Localización Específica

1. Cree el directorio ***c:\POT*** en su sistema.

5. 2. Descomprima el contenido del archivo **OSM-CONDABAPOTLOCALIZACIONESPECIFICA.ZIP** en el directorio ***c:\POT*** (archivo existente en el CD del sistema OSM-CONDABAPOT).

Los archivos instalados en la localización específica se relacionan en el archivo ***LeerLocalizacionEspecifica.txt*** que esta en el CD de instalación del sistema "OSM-CONDABAPOT"

3.6.7. Variables ODBC sistema OSM-CONDABAPOT

A continuación se presentan las bases de datos del sistema "OSM-CONDABAPOT" junto con las variables *ODBC* que deben ser configuradas (ver Tabla 10).

Tabla 10. Variables ODBC

Bases de datos	Ubicación	**Variable ODBC**
DBHisVis	Entidad Central	ENTIDADCENTRALHISVIS
DBAdmin	Entidad Central	CENTRAL_O_ESPECIFICA
DBTPOT	Localización Especifica	LOCALIZACIONPECIFICADBTPOT

3.6.8. Configuración Variables ODBC Entidad Central

Las variables *ODBC* permiten que los módulos instalados en la entidad central del sistema "OSM-CONDABATPO" puedan tener acceso a los datos de las bases de datos.

1. En el panel de control haga *click* sobre el icono **Herramientas administrativas** y *click* en el icono **Origen de Datos *ODBC*.**

2. Seleccione la pestaña ***DNS* de Usuario**

3. Haga *click* en el botón **Agregar*.***

4. Seleccione en la lista el *ODBC* perteneciente a Microsoft Access (*.mdb)

5. Seleccione el botón **Finalizar.**

6. Digite en **Nombre origen de Datos*:* ENTIDADCENTRALHISVIS**

7. Digite en **descripción*:* Base de Datos Registro Histórico consulta Remota POT**

8. Oprima el botón **Seleccionar**

9. Seleccione la base de datos presente en su directorio ***POT\DBCENTRAL\DBHisVis.mdb***

10. Nuevamente haga *Click* en el botón **Aceptar**

11. Haga *click* en el botón **Agregar*.***

12. Seleccione en la lista el *ODBC* perteneciente a Microsoft Access (*.mdb)

13. Seleccione el botón **Finalizar*.***

14. Digite en **Nombre origen de Datos*:* CENTRAL_O_ESPECIFICA**

15. Digite en **descripción: Base de Datos Administrativa POT**

16. Oprima el botón **Seleccionar**

17. Seleccione la base de datos presente en su directorio ***POT\DBADMIN\DBAdmin.mdb***

18. Haga *Click* en el botón **Aceptar**

19. Haga *Click* en el botón **Cancelar.**

3.6.9. Configuración Variable *ODBC* Localización Específica

Las variables *ODBC* permiten que los OSM del sistema "OSM-CONDABATPO" que visiten a la localización específica puedan tener acceso a los datos de la base de datos.

1. En el panel de control haga *click* sobre el icono **Herramientas administrativas** y *click* en el icono **Origen de Datos ODBC.**

2. Seleccione la pestaña ***DNS* de Usuario**

3. Haga *click* en el botón **Agregar.**

4. Seleccione en la lista el *ODBC* perteneciente a Microsoft Access (*.mdb)

5. Seleccione el botón **Finalizar.**

6. Digite en **Nombre origen de Datos: LOCALIZACIONPECIFICADBTPOT**

7. Digite en **descripción: Base de Datos POT**

8. Oprima el botón **Seleccionar**

9. Seleccione la base de datos presente en su directorio ***POT\DBESPECI\DBTPOT.mdb***

10. haga *Click* en el botón **Cancelar**

4. CONCLUSIONES

- El prototipo “OSM-CONDABAPOT” es un sistema de objetos de software móviles que se desplaza desde una entidad central a una localización específica para consultar la base de datos tipo Plan de Ordenamiento Territorial. Los objetos no son construidos con principios de inteligencia artificial.

- En la transferencia de los objetos móviles desde la entidad central a la localización específica el uso del canal es mínimo, debido al establecimiento de la comunicación cuando se despachan y cuando regresan.

- La comunicación es asincrónica en el prototipo, el uso del canal de comunicaciones es eficiente.

- La plataforma de agentes móviles *Aglets* es una plataforma de software que se superpone al sistema operativo y utiliza los protocolos de comunicación de este.

- Para la transferencia de los objetos móviles la plataforma de agentes *aglets* utiliza el protocolo de comunicaciones *ATP (Aglets Transfer Protocol)*.

- Java es un lenguaje que soporta diferentes sistemas operativos debido a su maquina virtual que se superpone al hardware y al S.O.. La mayoría de plataformas de agentes móviles lo utilizan.

- La evolución de los sistemas para desarrollar software de comunicación ha sido: *Pipeline, RCP, sockets*, objetos móviles, agentes móviles y plataformas multiagentes.

- Para desarrollar sistemas abiertos, se debe estandarizar los procesos tanto en el transmisor como en el receptor, para ello se deben construir procesos homologables en ambas arquitecturas. Por ello los fabricantes de cada plataforma deben llegar a acuerdos en que hacer, como hacerlo y donde usar la conectividad de los sistema.

- Al implementar los objetos de software móviles para la consulta de información en bases de datos tipo plan de ordenamiento territorial, se construyo una jerarquía de clases para consultar la información. Cada objeto fue especializado en un tipo de consulta.

- En el desarrollo del proyecto, se construyo un único objeto para consultar cualquier tabla en la base de datos tipo POT, pero se agruparon varios de ellos para realizar un tipo de consulta.

- La metodología UML soporta estereotipos, los cuales son muy importante debido a que permite modelar especificaciones propias de cada sistema y con ello se pueden proponer nuevos modelos de diseño. La investigación propone dos estereotipos.

- La metodología UML ha sido fundamental para el desarrollo del prototipo, debido a su flexibilidad en las etapas de construcción de software y al carácter estándar internacional que ofrece: Visualización, especificación, construcción, documentación y comunicación entre elementos.

- El plan de ordenamiento territorial de una localización específica (municipio, ciudad, corregimiento, vereda, barrio, etc.) se define como el conjunto de objetivos, directrices, políticas, estrategias, metas, programas, actuaciones y normas adoptadas para orientar y administrar el desarrollo físico del territorio y la utilización del suelo.

- El P.O.T. Es el instrumento de planeación y gestión del desarrollo territorial el cual incorpora las políticas nacionales, departamentales y los compromisos adquiridos por los mandatarios locales en su programa de gobierno. Estos planes y programas a su vez reflejan los intereses de la comunidad.

- Se desarrollo un prototipo innovador con tecnología de objetos de software móviles y aplicabilidad al Plan de ordenamiento territorial en Colombia.

- Las bases de datos que soportan *ODBC* son muy importantes, ya que cualquier aplicación de desarrollo la pueden consultar.

- Java integra *JDBC* y con ello permitió desarrollar el modulo de visualización de datos para consultar cualquier tabla en la base de datos. Se desarrollo una interfaz genérica para consulta. Esta interfaz grafica le permite al usuario lanzar cualquier SQL valido para ver información.

- Se utilizo JBuilder como herramienta para construir el modulo gráfico de OSM-CONDABAPOT, este desarrollador permite integrar componentes visuales con acceso a los datos de las bases de datos.

- JBuider permite construir automáticamente los diagramas de clases de acuerdo a la metodología de desarrollo UML.

- Con la jerarquía de clases propuesta por la investigación se puede consultar cualquier base de datos haciendo la instanciación correspondiente de diccionario. Soporta la consulta de cualquier tabla de la base de datos.

- La formulación y desarrollo de la investigación OSM-CONDABAPOT fue pertinente para formarme como investigador, ya que con ella desarrolle el hábito de escribir artículos.

- Se escribió el articulo “Agentes se Software móviles” en la revista del Nómadas No. 15 del departamento de investigaciones de la Universidad Central en 2001.

- Se escribió el articulo “Tecnología CORBA” en la revista Nómadas No. 17 del departamento de investigaciones de la Universidad Central en 2002.

- Se escribió el articulo en Julio de 2003 “Modelo de Referencia para Construir Software de Procesamiento Distribuido” propuesto para ser publicado en una de las revistas de la Asociación Colombiana para el Avance de la Ciencia (ACAC), actualmente en estudio por el comité de esta asociación.

5. SUGERENCIAS Y RECOMENDACIONES

- Se puede implementar Objetos de software móviles en la entidad central para que en caso de fallar la comunicación entre entidad central y localización específica; el objeto móvil intente el desplazamiento cuando se reestablezca la comunicación después de un determinado tiempo.

- Se puede estudiar la posibilidad de establecer las llamadas telefónicas o envió de fax con el método asincrónico para que en el momento de no haber transmisión se cierre el canal, y en el momento de transferir datos se establezca la comunicación. Ahorraría costos y uso del canal.

- La seguridad es un aspecto que el prototipo “OSM-CONDABAPOT” únicamente se contemplada en el marco teórico de las plataformas de agentes. Se recomienda que un grupo de investigación aborde el tema de seguridad.

- El control de concurrencia sobre la base de datos en la localización específica: simultaneidad en la consulta a través de la plataforma de agentes y la simultaneidad en el registro y/o la consulta de información en forma local. Son dos aspectos que no cubre el proyecto, pero que otro grupo de investigación lo podría tener en cuenta.

- JBuider JBuilder 8.0.140.0 como herramienta de desarrollo de software podría construir los diagramas de secuencia para el código Java. Actualmente construye los diagramas de paquetes UML.

- Un grupo de investigación podría abordar el tema de agentes ARB (Agent Resource Broker) y desarrollar software para intercambiar información des sistemas que soportan agentes y sistemas no lo soportan.

- Un área de estudio son los protocolos de comunicación entre agentes, de ahí podrían surgir otros proyectos de investigación para la la maestría

6. GLOSARIO

Acción (*action*):
Elemento básico que representa una actividad que un agente puede realizar. Una clase especial de acción es un acto de comunicación

AE:
Agent Environments. Ambiente de Agentes de la arquitectura MOA.

Agente ARB (*ARB Agent*):
Agente que proporciona el servicio ARB (Agent Resource Broker). Este servicio permite a un agente usar otros servicios fuera del mundo de los agentes (sistemas propietarios o "*legacy systems").*

Agent Communication Language (ACL):
Lenguaje con una sintaxis, semántica y pragmática formalmente definidas. Es la base de las comunicaciones entre agentes de una naturaleza heterogénea.

Agent Communication Channel (ACC) Router:
Es un agente que usa la información proporcionada por el AMS para encaminar los mensajes entre agentes dentro de la misma plataforma o de plataformas diferentes.

Agent Management System (AMS):
Es un agente que gestiona la creación, destrucción, suspensión, autentificación y migración de los agentes de la plataforma. Además proporciona un servicio de nombres para todos los agentes que residen en ella. Guarda la correspondencia entre GUID (*Globally Unique Identifier*) del agente y la dirección de transporte donde se encuentra.

Agent Platform (AP):
Proporciona la infraestructura necesaria para que los agentes puedan se utilizados. Un agente debe ser registrado en una plataforma para poder interactuar con otros agentes de esa plataforma. Un AP contiene tres componentes básicos : ACC, AMS y DF.

ARA:
Agents for Remote Actions. Agentes para la acción remota. Arquitectura para desarrollar agentes de software móviles.

Arquitectura Agente:
Define los mecanismos que permiten interconectar los componentes tanto de software como de hardware, que hacen que el agente se comporte como tal.

ASDK:
Aglets software development kit. Conjunto de herramientas de software para el desarrollo de agentes móviles.

ASM:
Agente de software móvil. Los agentes móviles son programas de software inteligentes que realizan un objetivo y pueden estar construidos con herramientas de desarrollo estructuradas, orientadas a objetos y/o concurrentes. Normalmente involucran desarrollos soportados con técnicas de Inteligencia Artificial.

Asociación (UML):
Relación estructural que describe un conjunto de enlaces, donde un enlace es una conexión entre objetos; la relación semántica entre dos o más clasificadores que implican las conexiones entre sus instancias.

ATP:
Aglet Transfer Protocol. Protocolo de transferencia de agentes. Es el protocolo de la capa de comunicaciones que utiliza el sistema de agentes *Aglets.*

API:
Programa de aplicación de interfaces. Son programas construidos y puestos para que el desarrollador de aplicaciones las use y acceda más fácilmente a otros componentes.

Caso de uso (UML):
Corresponden a una descripción de un conjunto de secuencias de acciones, incluyendo variaciones que un sistema lleva a cabo y que conduce a un resultado observable de interés para un actor determinado.

Colaboración (UML):
Sociedad de clases, interfaces y otros elementos que trabajan juntos para proporcionar algún comportamiento cooperativo que es mayor que la suma de todos los elementos; la especificación de como un elemento, como un caso de uso o una operación, es llevada a cabo por un conjunto de clasificadores y asociaciones desempeñando los roles específicos utilizados de una forma específica.

CORBA:
Common Object Request Broker Architecture, un estándar bien establecido que permite la comunicación de sistemas de objetos distribuidos.

COOL:
Es un lenguaje para la coordinación que modela la conversación entre los agentes mediante una máquina de estados finitos. Cada estado representa el punto de la conversación en la que nos encontramos.

Componente (UML):
Parte física y reemplazable de un sistema que se ajusta a, y que proporciona la realización de un conjunto de interfaces.

Clonar:
Termino usando en desarrollo de objetos, la clonación de un objeto es una replica exacta de otro objeto y que posee las mismas características.

Clase:
Es una entidad que existe por si sola y que la componen los atributos y los métodos. Los atributos son los elementos que la identifican y los métodos son las operaciones que se pueden realizar. Es un molde o plantilla. Es la abstracción de todos los objetos de un mismo tipo.

Clase activa (UML):
Una clase cuya instancia son objetos activos.

Directory Facilitator (DF):
Es un agente que proporciona un servicio de páginas amarillas. Guarda información de los agentes y de los servicios que estos ofrecen.

DCOM:
Microsoft Distributed Component Object Model. Componente para desarrollar aplicaciones distribuidas en Microsoft, arquitectura similar a CORBA.

Dependencia (UML):
Relación semántica entre dos elementos, en la cual un cambio en un elemento (la cosa independiente) puede afectar la semántica del otro elemento (la cosa dependiente).

Diagrama (UML):
Representación grafica de un conjunto de elementos, usualmente presentado como grafo conectado de vértices (elementos) y arcos (relaciones).

Diagrama de actividad (UML):
Muestra el flujo de actividad a actividad, los diagramas de actividad tratan la vista dinámica de un sistema. Un caso especial de diagramas de estados en el cual todos o casi todos los estados son estados de acción y en el cual todas o casi todas las transiciones son disparadas por la terminación de las acciones en los estados origen.

Diagrama de casos de uso (UML):
Muestra un conjunto de casos de uso y de actores y sus relaciones. Muestran el sistema desde un punto de vista estático.

Diagrama colaboración (UML):
Un diagrama de interacción que enfatiza la organización estructural de los objetos que envían y reciben mensajes. Muestra las interacciones organizadas alrededor de instancias y de los enlaces entre ellas.

Diagrama de componentes (UML):
Muestra un conjunto de componentes y sus relaciones. Muestran los componentes desde el punto de vista estático.

Diagrama de clases (UML):
Muestra un conjunto de clases, interfaces y colaboraciones y las relaciones entre ellos. Muestran el diseño de un sistema desde el punto de vista estático.

Diagrama de despliegue (UML):
Muestra un conjunto de nodos y sus relaciones, muestra el despliegue de un sistema desde un punto de vista estático.

Diagrama de estados (UML):
Muestra una máquina de estados, trata la vista dinámica de un sistema.

Diagrama de objetos (UML):
Muestra un conjunto de objetos y sus relaciones en un momento determinado, muestra el diseño o los procesos desde un punto de vista estático.

Diagrama de secuencia (UML):
Diagrama de interacción que hace énfasis en la ordenación temporal de los mensajes.

EC:
Entorno Computacional. Plataforma que soporta desarrollo y ejecución de *aglets*.

Elemento (UML):
Un constituyente atómico de un modelo.

Estereotipo (UML):
Corresponde a una extensión del vocabulario UML, que permite la creación de nuevos tipos de bloques de construcción que se derivan de otros existentes pero que son específicos a un problema particular.

Evento:
Es una ocurrencia dentro de un objeto específico que puede ser de interés para uno o más objetos. Son transiciones que tiene un objeto para ir de un estado a otro.

FIPA:
Foundation for Intelligent Physical Agents. FIPA se ha convertido en el estándar con más repercusión y aceptación social. Trata todos los temas relacionados con los sistemas multiagente proporcionando una amplia gama de documentos.

FIPA ACL:
Toda la plataforma FIPA se basa en la comunicación entre agentes mediante ACL *(Agent Communication Language*). Un mensaje ACL consiste básicamente en una expresión de un determinado lenguaje y con un conjunto de términos específicos de la ontología usada.

FRAMEWORK:
Armazón basado en java para construir aplicaciones distribuidas usando el paradigma de agentes móviles.

FTP:
File Transfer Protocol. Protocolo para la transferencia de archivos.

Generalización (UML):
Una relación de especialización/generalización en la que objetos del elemento especializado (el subtipo) son sustituibles por objetos del elemento generalizado (el supertipo).

GIOP:
General Inter-ORB Protocol. Es el protocolo de comunicación estándar entre ORB's establecido por la especificación CORBA.

GUID:
Globally Unique Identifier. Es la identidad propia del agente. Es el nombre del agente.

HAP:
Home Agent Platform. Es la plataforma en la cual se creó un agente. La responsabilidad del HAP es garantizar la identidad del agente en sus relaciones con otros agentes y plataformas.

IDL:
Interface Definition Language. Lenguaje estandarizado por CORBA. OMG IDL es un lenguaje declarativo estándar. El léxico del lenguaje OMG IDL obedece las mismas reglas que el léxico del lenguaje C++.

IIOP:
Internet inter Orb Protocol. Protocolo para la comunicación entre ORB´s *(Object Request Broker)* en Internet.

Inter-EC:
Entorno computacional de agentes, el inter-EC es implementado en el servidor de *Aglets*.

Interfaz (UML):
Colección de operaciones que son utilizadas para especificar un servicio de una clase o de un componente.

Instanciar:
Es un término utilizado al concretar una clase. Es la definición de un objeto del tipo de una clase. Es la concreción de una clase.

Interacción (UML):
Comportamiento que consta de un conjunto de mensajes intercambiados por un conjunto de objetos dentro de un contexto particular para llevar a cabo un propósito específico.

IPMT:
Internal Platform Message Transport . Método de intercambio de mensajes dentro de la misma plataforma. Depende de la implementación.

Interfase:
Medio o elemento que permite establecer la comunicación entre dos entornos diferentes.

JATLITE:
Java Agent Template. Es un témplate de la universidad de *Standford.* Estándar de agentes desarrollado en java.

JVM:
Maquina virtual de java. El JVM se crea de acuerdo al sistema operativo donde se instale el java.

KIF:
Knowledge Interchange Format. Es un lenguaje de Contenido. Es la sintaxis en el lenguaje de comunicación común.

KQML:
Knowledge Querying and Manipulation Language. Es la pragmática en un lenguaje de comunicación común:

KSE:
Knowledge Sharing Effort. Es una organización de estandarización que busca compartir conocimiento entre agentes.

Localización Específica:
Unidad geográfica definida. Por ejemplo: Municipio, aldea, caserío, vereda etc.

MAC:
Message Authentication Code: sistema de seguridad para encriptación de llave privada mediante la obtención de un valor por técnica hash.

MACE:
Ambiente de ejecución interpretado para agentes móviles desarrollados en C/C++ que esta basado en un modelo de máquina abstracta. El compilador MACE genera *byte code* derivado de una representación interna de *RTL (Register Transfer Language)*. Código producido por el compilador GNU C de C/C++.

MAFAgentSystem:
El interfaz *MAFAgentSystem* define las operaciones de control de los agentes:

MAFFinder:

El *MAFFinder* es un objeto servidor de nombres. Antes de que un cliente pueda solicitar al objeto *MAFFinder* la búsqueda de un objeto. El cliente tiene que obtener el objeto referencia del *MAFFinder*.

Máquina de estados (UML):
Comportamiento que específica las secuencias de estados por los que pasa un objeto durante su tiempo de vida en respuesta a eventos, junto con sus respuesta a dichos eventos.

Marco de trabajo (UML):
Patrón de la arquitectura que proporciona una plantilla extensible para aplicaciones dentro de un dominio específico.

MASIF:
Mobile Agent System Interoperability Facilities. Corresponde a la OMG. Es una organización de estandarización internacional para agentes. Considera a los agentes como objetos CORBA que tienen la posibilidad de moverse, ejecutarse autónoma y asincrónicamente en sistemas de ejecución seguros (sistemas de agentes).

Método:
Es la implementación de una operación.

MOA:
Mobile Objects and Agents. Es una plataforma que soporta agentes de software móviles creado por *Open Group*.

Modelo (UML):
Abstracción de un sistema cerrado semánticamente.

MIC:
Message Integrity Code: Sistema de integridad en la transferencia de agentes.

Nodo (UML):
Elemento físico que existe en tiempo de ejecución y que representa un recurso computacional, en general tiene alguna memoria y a menudo capacidad de procesamiento.

Nota (UML):
Comentario asociado a un elemento o a un conjunto de elementos.

Notificación:
Es un mensaje que un objeto envía a los grupos de interés par informarles que un evento específico ha ocurrido.

Objeto:
Es la instancia de una clase. Es la concreción de la clase, es un objeto determinado que posee valores.

OMG:
Object Management Group. Organización Internacional que desarrollo la especificación CORBA.

Ontología (Ontology):
Una ontología da significado a los símbolos de un determinado dominio de conversación. Para que dos agentes puedan entenderse es necesario que ambos estén dando el mismo significado a los símbolos de los mensajes intercambiado.

ORB:
Object Request Broker. Permite que el agente use otros servicios fuera del mundo del objeto.

OSM:
Objeto de software móvil. Objeto que se mueve entre dos o más aplicaciones que pueden estar ubicadas en diferentes sitios.

Paquete (UML):
Mecanismo de propósito general para organizar elementos en un grupo.

POT:
Plan de Ordenamiento territorial donde se especifica a largo plazo el desarrollo de un municipio y/o ciudad.

Protocolo (Protocol):
Un patrón de mensajes intercambiados entre agentes para realizar una determinada tarea. Son normalmente usados para simplificar la lógica que es necesaria implementar para posibilitar el dialogo entre agentes. Es un conjunto de reglas preestablecidas entre dos componentes que garantiza el poder intercambiar información.

Proxy:
Objeto que proporciona una interfaz local a un objeto remoto. Redirige las peticiones al objeto remoto.

RMI*:*
Remote Method Invocation. Método de invocación remota soportado por JAVA.

Polimorfismo:
Permite que un método tenga múltiples implementaciones que se seleccionan en base a que tipo de objeto se le pasa en la llamada al método.

Relación (UML):
Conexión semántica entre elementos.

RPC:
Llamada a procedimientos remotos. Programación que no exige transferencia de código en el entorno de red, únicamente transporte de resultados. Este tipo de programación no se debe confundir con la programación de sistemas distribuidos basados en objetos móviles.

Sistema (UML):
Colección de subsistemas organizados para llevar a cabo un propósito específico y descrito por un conjunto de modelo, posiblemente desde diferentes puntos de vista.

Sistema distribuido:
Conjunto de computadores autónomos unidos por una red de comunicaciones y equipados con software de sistemas distribuidos.

Subsistema (UML):
Agrupación de elementos de los que algunos constituyen una especificación del comportamiento ofrecido por los otros elementos contenidos.

SMTP:
Simple mail transfer protocol: Protocolo sencillo de transferencia de correo electrónico, protocolo que proporciona una función básica de correo electrónico entre computadoras remotas.

UML:
Lenguaje de modelado Unificado. Lenguaje estándar grafico para visualizar, especificar, construir y documentar el modelado de un sistemas de software.

XDR:
Representación externa estandarizada para transmitir y recibir valores. *eXternal Data Representation* de SUN.

7. BIBLIOGRAFIA

REFERENCIAS

1. N. R. Jennings, K. Sycara, M. Wooldridge. *"A Roadmap of Agent Research and Development". Autonomous Agents and Multi-Agent Systems, I, 1.998.*

2. "Diccionario de la Lengua Española". Real Academia Española. XXI Edición, 1.992.

3. *"MASIF-RTF Results". Object Management Group*, 1.998.

4. Russel, S., Norvig, P., *Artificial intelligence: A Modern Approach. Prentice Hall.* 1995.

5. Hayes-Roth, B. *An architecture for adaptive intelligent systems. Artificial Intelligence, Pages* 72, 329 – 365. *Age* 1995.

6. *The Foundation for Intelligent Physical Agents.* FIPA *Specification* http://www.fipa.org.

7. *Object Management Group* http://www.omg.org

8. *Specification of KQML Agent-Communication Language plus example agent policies and architectures, The DARPA Knowledge Sharing Initiative, External Interfaces Working Group,* 1993. http://www.cs.umbc.edu

9. *FIPA Abstract Architecture Specification and Agent Management Specification,* http://www.fipa.org.

10. *Agent Society Home Page.* http://www.coldgin.demon.co.uk/people/pagojo/ColloqPaper/ColloqPaperWeb.html. Información sobre movilidad de agentes.

11. *Ravhid Guerraoui, Mohamed E. Fayad. Oriented Object Distributed Programming Is Not Distributed Oriented Object Programming* 1999.

12. *B. Venners. "Solve Real Problems with Aglets, a Type of Mobile Agents". JavaWorld. mayo 1.997.* http://www.javaworld.com/javaworld/jw-05-1997/jw-05-hood.html

13. Caramazana Cárcamo, Alberto. Agentes Móviles. Consultor Senior de *Ideal Objects*.

14. Fernández Moya, Francisco. INTRODUCCIÓN A LA ARQUITECTURA CORBA PARA SISTEMAS DISTRIBUIDOS. Grupo de Arquitectura y Redes de Computadores 2001.

15. Diosa, Henry Alberto, "Propuesta para la conformación de grupo de trabajo en procesamiento basado en objetos distribuidos con CORBA y Objetos de Software Móviles Usando Java". Universidad Distrital Francisco José de Caldas. Bogota. Agosto de 2000.

16. Cruz Martínez, Miguel. Programación Orientada a Agentes móviles en Java. Sistemas Multi-agente.

17. Posada Yagüe Juan Luis, Arquitectura de Procesos en sistemas Reactivos Distribuidos. Departamento de informática de sistemas y computadores septiembre del 2000.

18. García Daniel Alonso. Introducción al estándar FIPA. Departamento de Sistemas Informáticos y Programación, UCM Informe Técnico UCM-DSIP 98-00 Versión 1.0.

19. IBM. Página Web del sistema de agentes móviles *aglets* de IBM. http://www.trl.ibm.co.jp/aglets.

20. Pavón Mestras, Juan. Agentes Móviles. Departamento de Sistemas Informáticos y Programación. Universidad Complutense Madrid. Ponencia "*Mobile Agents - An overview",* presentado en la conferencia ACTS *IS&N, Cernobio (Como), Italy, May* 27-29, 1997 por el Dr. *Thomas Magedanz (GMD Focus e IKV++).*

21. Zunino Alejandro, Iturregui Ramiro y Campo Marcelo. ARQUITECTURAS DE AGENTES MÓVILES. ISISTAN *Research Institute.* Facultad de Ciencias Exactas, Dpto. Computación y Sistemas. Universidad Nacional del Centro Campus Univ. Paraje Arroyo Seco - (7000) Tandil - Bs. As., Argentina.

22. GÓMEZ LABRADOR, RAMÓN M. Agentes Móviles Y CORBA, Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla, Febrero de 1999.

23. Morales Cuesta Pedro. Sistemas Multiagente. Área de Lenguajes y Sistemas Informáticos. Escuela Superior de Ingeniería Informática. Universidad de Vigo. Versión 2.0 Marzo 7 de 2001.

24. Wooldridge, M. J., & Jennings, N. R. (Ed.). *Intelligent Agents: ECAI-94 Workshop on Agent Theories, Architectures, and Languages. Berlin: Springer-Verlag.* 1995.

25. Franklin, S., & Graesser, *A. Is it an agent, or just a program?: A taxonomy for autonomous agents.* 1996.

26. Ley 388 de 1997. Congreso de La Republica de Colombia. 1997.

27. Ley 99 del 22 de diciembre de 1993. Congreso de La Republica de Colombia. 1993.

28. *Reticular System Inc. AgentBuider, User´s Guide version* 1.3 Rev. 0 *April* 30, 2000 http://www.agentbuilder.com/AgentTools/index.html

29. Nils J., Nilsson, *Inteligencia Artificial. Stanford University. McGraw* Hill. España, 2001.

30. Newell, A. Putting *It All Together. In D. Klahr & K. Kotovsky (Eds.), Complex Information Processing: The Impact of Herbert Simon. Hillsdale, NJ: Lawrence Erlbaum.* 1988.

31. *Mobile Agents: Are they a good idea?. Mobile Object Systems. Second International Workshop, MOS'96. Linz, Austria. Julio* 1996

32. Hyacinth Nwana, Lyndon Lee, Nick Jennings. *Coordination in Software Agent Systems. BT Technology Journal, April* 14, 1996.

33. Iglesias Fernández, Carlos Ángel. Definición de una Metodología para el desarrollo de sistemas Multiagente. Universidad Politécnica de Madrid. Departamento de Ingeniería de Sistemas Telemáticos. 1998.

34. Perez Diaz, Jesus Arturo. Arquitectura de Seguridad Integral para sistemas de Agentes Móviles. Departamento de Informática, Universidad de Oviedo. España 2000.

35. Álvarez García, Fernando. Objetos Distribuidos y Agentes Móviles (Parte I). Departamento de Informática. Universidad de Oviedo. Junio de 2001

36. *White, J. E. Telescript technology: The foundation for the electronic marketplace. White paper, General Magic,* Inc., *2465 Latham Street, Mountain View,* CA 94040. 1994.

37. *White, J. E.. Telescript technology: Mobile agents. In Bradshaw, J., editor, Software Agents. Press/MIT Press.* 1996.

38. Avancini, Henri Hector. Un *Framework* para Sistemas Multi-agente basado en Composición. Universidad Nacional Del Centro De La Provincia De Buenos Aires. Facultad De Ciencias Exactas. Junio de 2000.

39. http://ict2.udlap.mx/people/adolfo/doctesis.html Página sobre *JatLite Beans* y Referencia a tecnología de agentes móviles.

40. Naughton, Patrick. Manual de Java. *McGrawHill*. España. 1996.

41. García Molina, Jesús. El proceso del Software Basado en UML para Sistemas de Información. Departamento de Informática y Sistemas. Universidad de Murcia. 2002. http://dis.um.es/~jmolina/publi.html.

42. Martínez Barrera, Crisman. Tecnología CORBA. Departamento de Investigaciones Universidad Central. Nómadas No. 17. Colombia 2002.

43. http://www.informatica.us.es/~ramon/tesis/CORBA/Seminario-MASIF/ Página que referencia la arquitectura grasshopper.

44. Herrera, Luz Helena. Ingeniera Ambiental y Sanitaria. Lineamientos generales para el desarrollo de Laboratorios el Arcano del Universo. Tabio(Cund.). 2001.

45. B. Bamberg (DTB), D.Baiotta(ITL) and others. *MARINE Agent Platforms. Editorial F Chatzipapadopoulos(NTU). M, M Perdikeas(NTU)* http://www.fokus.fhg.de/research/cc/ecco/climate/ap-documents/marine-agplatf.doc. 1998.

46. Arquitectura MOA. Mobile Objects and Agents. http://www.usenix.org/publications/library/proceedings/coots98/full_papers/milojicic/milojicic_html/milojicic.html

47. Jacobson Ivar, Booch Grady, Rumbaugh James. El proceso Unificado de Desarrollo de Software. *Addison Wesley*. México. 1999.

48. Jacobson Ivar, Booch Grady, Rumbaugh James. El proceso Unificado de Desarrollo de Software. *Addison Wesley*. México. 1999.

49. Cartilla IGAC. Instituto Agustín Codazzi. Página del instituto: http://www.igac.gov.co/pot.htm

50. ASDK disponible en http://www.javasoft.com/products/jdk/1.1/index.html. Empresa propietaria: *IBM Corporation* 1996, 1997 *All Rightss Reserved.* Versión utilizada: *IBM Aglets Class Library* 1.1 beta 3 Revisión 13. *Aglets* API 1.2. La versión utilizada es libre.

51. JDK disponible en http://java.sun.com/ Empresa propietaria: *Sun Microsystems*, Inc. Copyright © 1996-1999, 901 San Antonio Road, Palo Alto, CA 94303 USA. La versión utilizada es libre.

52. Access 97 disponible para actualización en www.microsoft.com. Empresa propietaria © Microsoft Corporation Copyright 1992 -2001. La versión utilizada es preinstalada.

53. JBuilder disponible en: www.borland.com Empresa propietaria: Borland Software Corporation. 100 Enterprise Way, Scotts Valley, CA 95066-3249. COPYRIGHT © 1997-2003 Reservados todos los derechos. La versión utilizada es de prueba por 3 años.

54. ODBC disponible para actualización en www.microsoft.com. Empresa propietaria © Microsoft Corporation Copyright 1992 -2001. La versión utilizada es preinstalada.

BIBLIOGRAFÍA

Larman, C. UML y patrones. Prentice Hall. 1999.

Martínez Barrera, Crisman. Agentes de software Móviles. Departamento de Investigaciones Universidad Central. Nómadas No. 15. Colombia 2001.

Olmedo Aguirre, José Oscar. Invocación a Métodos Remotos. 2001.

Acuerdo No. 16 de 1998. Corporación Autónoma Regional de cundinamarca. CAR. 1998.

Gobernación de Cundinamarca. Cundinamarca en cifras. Departamento Administrativo de planeación. Unidad de análisis estadístico Septiembre 1995.

Manual de Radiación Solar. Estaciones climáticas en Colombia.

Reticular System Inc. AgentBuider, Reference Manual versión 1.3 Rev. 0 April 11, 2000 www.agentbuilder.com

TANENBAUM, ANDREW S. Redes de Computadoras. Prentice Hall. Tercera Edicion. Mexico. 1997.

Stalling William. Comunicaciones y Redes de Computadores. Prentice Hall. Sexta edición. Madrid 2000.

Tomasi Wayne. Sistemas de Comunicacion es Electrónicas. Prentice Hall. Segunda edición. Mexico 1996.

Coutch W. Leon. Sistemas de Comunicación Digitales y analógicos. Prentice Hall. Quinta edición. Mexico 1998.

Agullo Soliveres, Pedro. Acceso a Bases de Datos con Java: JDBC(I). Publicado en la Revista Profesional para Programadores (RPP).

Silberschatz Abraham, Korth Henry. Fundamentos de bases de datos. Tercera edición. McGrawHill. Mexico. 1998.

Becerra Santamaria, Cesar. Los 600 Métodos de Java. Editarial. Computador Ltda. Bogota Colombia. 1998.

Joyanes Aguilar, Luis. Programación Orientada a Objetos. Segunda edición. McGraw Hill. España. 1998.

Marck Allen, Weiss. Estructura de Datos en Java. Addison Wesley. Madrid. 2000.

Tennembaum, Adnrew S. Sistemas Operativos Distribuidos. Prentice Hall. Primera Edición. Mexico 1996.

Danny B. Lange y Mitsuru Oshima. Programing and Deploying Java Mobile Agents with Aglets. 1998.

Comunications Magazine, IEEE, July 1998. Mobile Software Agents for Telecomunications.

Borland Software Corporation. Introducción a JBuilder. Versión 8. www.borland.com

Borland Software Corporation. Creación de Aplicaciones con JBuilder. Versión 8. www.borland.com

Borland Software Corporation. Diseño de Aplicaciones con JBuilder. Versión 8. www.borland.com

Borland Software Corporation. Guia del Desarrollador con JBuilder. Versión 8. www.borland.com

Borland Software Corporation. Guia del Desarrollador de Enterprise JavaBeans con JBuilder. Versión 8. www.borland.com

Borland Software Corporation. Procedimientos iniciales con Java. Versión 8. www.borland.com

Borland Software Corporation. Guia del Desarrollo de Aplicaciones de Base de Datos. Versión 8. www.borland.com

SJeng. Shyn-Kang. *Mobile Agents. National Taiwan University. Departament of Electrical Engineering. Institute of Communication Engineering.* 2000.

htpp://www.Java.sun.com Página de la compañía *sun Microsystem.* Documentación y software de Java.

htpp://www.comsoc.org Página que referencia la sociedad de comunicaciones de la IEEE

htpp://www.cs.uit,no/forskning/DOS. Página de distribución de sistemas operativos y plataformas de agentes como Tacoma.

http://fccn.pt/crc1999/FINAIS/artigo02/ARTIGO02.htm. Pagina de agentes utilizando SNMP.

htpp://www.tinac.com. Telecommunicactions Information Networking Arquitecture Consortium.

htpp://www.SoftwareDirectory.org
Directorio de sitios de software

htpp://www.cs.dartmouth.edu. Página del Departamento de ciencias de la computación. Dartmouth College.

htpp://www.omg.org/library\wpjava.html. Página de la organización de gestión de objetos. Documento sobre Java, RMI, y CORBA.

htpp://www.igac.gov.co
Instituto Geográfico Agustín Codazzi de Colombia

http://www.red.gov.co/LaInstitucion/Normatividad/Ley388-1997/ley388-1997.html. Pagina que contiene la ley 388 de 1997. Decreto reglamentario del plan de ordenamiento territorial.

http:www.rediris.es/list/info/agentes.es.html. Grupo de investigación de agentes y sistemas multiagentes.

http://mx.gropups.yahoo.com/group/agentes-inteligentes/. Grupo de Agentes móviles.

htttp://www.google.com. Buscador de Internet.

8. ANEXOS