BENEMERITA
UNIVERSIDAD AUTONOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

Ingeniería en Ciencias de la Computación

Búsqueda y recuperación de Objetos de Aprendizaje con Agentes Móviles

TESIS

Que para obtener el grado de

Licenciado en Ingeniería en Ciencias de la Computación

PRESENTA

Edgar Fabián Hernández Ventura

Asesor de Tesis
Dra. Darnes Vilariño Ayala
Coasesor
Dra. Fabiola López y López

INDICE

Índice de Figuras

Introducción

Nuestra Universidad, debe ir a la vanguardia tecnológica y educativa, vivimos en tiempos de cambio, donde la educación debe de ser dinámica y brindar los bloques de conocimiento necesarios para construir un conocimiento útil. No utilizar las nuevas tecnologías que se nos presentan sería olvidar el compromiso de estar a la vanguardia ofreciendo educación a todo aquel con necesidad de recibirla.

La tendencia la marca el más rápido, en esa premisa se basa la educación actual. Ya no es suficiente con un aprendizaje lineal en el cual se está sujeto a horarios, disposición y ritmo diferentes a los propios. Somos parte de una generación visual e interactiva, donde la construcción del conocimiento se da de una manera modular y aglomerada, con un constante bombardeo de información, además de aprender debemos de prescindir de toda aquella información que no nos es útil.

Las brechas económicas no disminuyen, pero si lo hacen las brechas culturales y educacionales, el acceso a la información es la clave del cambio y del desarrollo humano.

Aprovechando la posibilidad que nos ofrece la red de redes, el Internet, usaremos sus ventajas inheritas como la posibilidad de ser accesado desde cualquier parte y en cualquier momento para llevar la educación a quien lo requiera, no para sustituir la educación presencial que es un pilar fundamental, sino para ampliar el rango de recursos de aprendizaje de los educandos.

Un reto de la educación a distancia o e-learning, ha sido la poca reutilización de los cursos desarrollados. Esta desventaja se da por que una vez creado un curso para educación a distancia, este se muestra como una unidad indivisible, si en cambio se toma cada elemento de este como unidad, se tienen múltiples piezas de poca relevancia, como párrafos de texto sueltos y sin coherencia, o imágenes sin referencia.

La estrategia a utilizar para este proyecto será la de los Objetos de Aprendizaje (OA). Que es el punto medio de granularidad entre lo básico de un elemento multimedia (como lo sería una imagen o un archivo de sonido) y un curso completo. Un ejemplo podría ser una simple página con texto y dos imágenes describiendo pieza arqueológica de la cultura Olmeca, o un tutorial interactivo de como dar mantenimiento a una pieza de maquinaria.

Estos OA tienen que ser auto-contenidos y descritos de una manera precisa y estructurada para ser recuperados posteriormente. Esta descripción es dada en metadatos que son "datos acerca de los datos" similares a las fichas bibliográficas pero almacenadas en archivos llamados manifiestos.

Una vez que estos objetos han sido creados, propiamente empaquetados y descritos en sus manifiestos de metadatos, deben de ser almacenados de manera que se permita hacer consultas, negociaciones y recuperaciones de una manera simple, transparente y confiable. Por esto surge el esquema de un Mercado de Objetos de Aprendizaje (MOA), el cual sigue una Arquitectura Orientada a Servicios (AOS), la cual se compone de tres elementos principales: Consumidor, Proveedor y Administrador. Este modelo de mercado está manejado por agentes, en representación de los tres elementos mencionados con anterioridad. En particular el objetivo de esta tesis será crear el aparato de búsqueda y recuperación de OA previamente almacenados, con el objetivo de actualizar la base de datos que manejan los agentes proveedores de OA. Para esto se crearán agentes móviles que tengan un tópico o criterio de búsqueda especifico, se desplacen desde los diferentes proveedores y busquen OA que cumplan estos requisitos, los encapsulen en si mismos y regresen con estos al origen. Será por medio de estas búsquedas distribuidas que se facilita el trabajo del usuario al construir sus objetos de aprendizaje y cursos completos a partir de uno o más OA.

Los agentes móviles, son agentes de software con características de autonomía, sociabilidad, aprendizaje y la más importante, movilidad. Con agentes móviles nos referimos a procesos o aplicaciones que pueden transportar su estado de un ambiente a otro, con sus datos intactos, y seguir trabajando adecuadamente. Los agentes móviles deciden cuando y donde moverse la siguiente vez, lo cual es más avanzado que una simple llamada a procedimientos remotos. De la misma manera que un usuario no visita un sitio Web, sólo hace una copia de él, un agente móvil duplica sus datos. Cuando el agente móvil, decide moverse, guarda su estado de proceso, y transporta este estado al siguiente host para continuar desde el estado guardado. [1].

La estructura del MOA es descrita a continuación.

Agente proveedor (AP).

Representa al creador del objeto de aprendizaje. Estará encargado de almacenar los OAs y ponerlos a disposición de los consumidores. Las tareas que realizará serán.

- Registro en el MOA. Se encargará de darse de alta en el MOA y de esta manera poder ser localizado por los consumidores.
- Negociar. Si un consumidor solicita un OA se inicia un proceso de compraventa en el cual se negocia el precio y los términos de disponibilidad.

Agente Interfaz del Proveedor (AIP).

Esta es la interfaz gráfica de interacción con el usuario creador que proporciona controles para tareas comunes. Además provee de las siguientes.

- Mostrar contrato. Muestra el contrato con los términos, condiciones y restricciones de uso de los OAs propios.
- Mostrar políticas de ingreso y permanencia en el MOA. Muestra el contrato con los términos, condiciones y restricciones de acceso al MOA.

Agente Consumidor (AC).

Representa al diseñador de cursos el cual necesita los OA creados. Estos están almacenados por los proveedores, y será su función facilitar la búsqueda a través de diferentes proveedores. Algunas de las funciones son.

- Registro en el MOA. Se registrará en el MOA para que exista un control y acerca de los consumidores activos y almacenar estadísticas de uso.
- Localizar OA. Cuando se decida obtener un OA con ciertas características se invocará a esta función la cual será la encargada de crear un agente que se *mueva* donde los proveedores para encontrar el OA necesario.
- Negociar Contrato. Cuando es encontrado el OA, se inicia el proceso de negociación para llegar a un acuerdo con respecto a precio y condiciones de uso.
- Darse de baja del MOA. Función para darse de baja en el MOA y así cesar las actividades definitivamente.

Agente Interfaz del Consumidor (AIC).

Esta interfaz gráfica para el usuario brinda los controles para poder realizar las tareas comunes del diseñador de cursos.

- Mostrar contrato. Muestra el contrato con los términos, condiciones y restricciones de uso de los OA del proveedor.
- Mostrar políticas de ingreso y permanencia en el MOA. Muestra el contrato con los términos, condiciones y restricciones MOA.

Agente Administrador (AA).

Este agente representa al dueño del MOA. Su función es regir el uso del mercado y asegurar la visibilidad de los agentes proveedores y consumidores. Debido a la naturaleza distribuida del mercado, el Agente Administrador solamente tiene control sobre ciertas funciones no así de las transacciones entre proveedores y consumidores.

- Crear el MOA. Inicializa las bases de datos y la plataforma del MOA. Así como agentes secundarios de recolección y actualización para dar soporte al estado del MOA.
- Altas y bajas de Agentes Proveedores. Es el encargado de mantener registradas las direcciones de los agentes proveedores activos.
- Altas y bajas de los Agentes Consumidores. Parte responsable de mantener registradas las direcciones de los agentes proveedores activos.
- Manejo de Contratos. Esta función es realizada por un agente que controla el flujo de información referente a los contratos. Debido a la privacidad que se debe de mantener sobre los datos de los participantes en los convenios, así como para asegurar que el proceso sea seguro para ambas partes este debe de ser controlado por el MOA.

Agente de Búsqueda y Recuperación de Objetos de Aprendizaje (ABROA).

El usuario proveedor tendrá posibles OA almacenados en su servidor que deben de ser empaquetados propiamente para ser puestos a disposición de los consumidores. Para que un posible OA (una página web almacenada o una carpeta con su contenido descrito en un archivo manifiesto) en la máquina del usuario proveedor pueda ser publicado en el mercado

en forma de un OA tiene que estar autocontenido en un archivo ZIP, tiene que tener un archivo manifiesto describiendo su contenido, los recursos que la componen y los archivos de lenguaje de definición de esquema de XML (*XSDL*) necesarios para definir la estructura del archivo manifiesto. Además se requiere de un medio para publicar estos OA en la base de datos del proveedor y así el Agente Proveedor pueda publicarlos en la base de datos del mercado, por lo tanto se requiere de un método para entregar el OA de una manera segura al consumidor.

- Buscar posibles OA en el servidor del proveedor
- Empaquetar los posibles OA encontrados de acuerdo al estándar SCORM 2004
- Publicar OA. Una vez creado y almacenado un OA se publica la base de datos del proveedor para que se lleve un registro de los OA disponibles así como algunas de sus características.
- Dar de baja OA. Método para excluir un OA de la disponibilidad para los consumidores.
- Entregar OA. Una vez terminada satisfactoriamente la negociación se envía el OA al consumidor.

Objetivos

Objetivos Generales

1. Diseño e implementación de un Agente de búsqueda y recuperación de objetos de aprendizaje.
2. Diseño e implementación de un agente móvil para la entrega de objetos de aprendizaje solicitados (agente cartero)

Objetivos Específicos

1. Investigar los LMCS (*Learning Managment Content System*) existentes.
2. Investigar los repositorios de OA existentes.
3. Investigar los estándares para OA existentes.
4. Investigar los estándares para LOM existentes.
5. Investigar el funcionamiento de la plataforma JADE.
6. Diseñar e implementar un analizador de archivos manifiesto.
7. Diseñar e implementar un analizador de archivos HTML.

8. Desarrollar una interfaz para facilitar la interacción y mostrar el comportamiento del agente ABROA.

Estructura de la Tesis

Como fue descrito anteriormente esta tesis tiene como objetivo mejorar el funcionamiento del MOA, en cuanto a la entrega de los OA por parte del proveedor al consumidor. Uno de los más grandes retos son los analizadores de archivos manifiestos y HTML puesto que para poder analizarlos es necesario comprender su estructura y contenido. El paradigma de agente como metodología de desarrollo abre un gran número de oportunidades pero también de retos.

Por esto se hace énfasis en la descripción de los conceptos teóricos de agente. Después se describe brevemente que son los objetos de aprendizaje y su uso, en especial del estándar más reconocido y usado para el MOA, el estándar SCORM. Estos aspectos teóricos conforman el primer capitulo.

Posteriormente se procede con el diseño del sistema, siguiendo el proceso unificado de desarrollo de software. Se muestra el comportamiento del sistema a través de los diagramas de casos de uso, diagramas de clases general y detallado y los diagramas de secuencia, aspectos que se discuten en el capítulo 2.

En el tercer capítulo se discute la implementación desarrollada del sistema, explicando la plataforma JADE, y las primitivas de movilidad, se discuten además los analizadores desarrollados

Finalmente se muestran una serie de pruebas para demostrar el correcto funcionamiento de los agentes para finalizar con el cuarto y último capitulo de esta tesis

Capítulo I. Aspectos teóricos

I.1 Agentes

Desde el inicio de la historia escrita, hemos estado fascinados con la idea de entes no humanos. Las nociones populares acerca de *androides, humanoides, robots, cyborgs*, y criaturas de ciencia ficción están dispersas en nuestra cultura formando una idea errónea acerca de como son percibidos los agentes de software. La palabra "*robot*" se deriva de la palabra para "trabajo duro" en el idioma checo, y se hizo popular después de una obra de teatro de 1921 llamada *RUR: Rossum Universal Robots*. Los robots de la obra eran trabajadores de una fábrica, pero el público en general ha aceptado la idea de los robots como "mayordomos digitales" quienes algún día realizarán las tareas mundanas del hogar. A pesar de los inicios tan simples e inocentes de la palabra, el público puede ver a estas criaturas con inteligencia artificial con temor. ¿Podrá el fantástico poder de los *robots* volverse en contra de los humanos? Las experiencias cotidianas de los usuarios de computadoras personales contra los misterios del software común, los molestos fallos, características incomprensibles, y peligrosos virus refuerzan el temor de que el *software* que dará vida a las criaturas autónomas acarreará sin duda problemas. Entre más inteligente es un *robot*, es más capaz de perseguir sus propios intereses en vez de los de su amo. Entre mas se parezca un *robot* a un humano, es más probable que exhiba los defectos y excentricidades humanas. Estas preocupaciones no pueden pasarse por alto en el diseño de agentes de software, de hecho, muchas de ellas tiene algo de razón.

Aunque los autómatas de varios tipos han existido desde hace siglos, es sólo con el desarrollo de las computadoras, y la teoría de control desde la segunda guerra mundial cuando parecen los agentes autónomos. Quizá lo antecesores más relevantes de los agentes inteligentes de hoy en día son los servo-mecanismos y otros dispositivos de control, incluyendo el control de las fábricas y el piloto automático de los aviones. Sin embargo hay una diferencia significativa, ya que el enfoque ahora está en el software en vez del hardware, de las piezas que componen a un robot mecánico a los bits que forman a un agente digital **[3]**.

Nwana **[4]** divide la investigación de agentes en dos corrientes principales: la primera que inició alrededor de 1977, y la segunda que inició alrededor de 1990. La corriente 1, cuyas raíces eran principalmente la inteligencia artificial distribuida que "se ha concentrado principalmente en los agentes de toma de decisiones con modelos simbólicos internos. Cuyo trabajo ha contribuido a un entendimiento de los problemas macrometricos como la interacción y comunicación entre los agentes, la descomposición y la distribución de tareas, la coordinación y cooperación, la resolución de conflictos vía negociación, etc.". Y la corriente 2, en contraste, es un movimiento más reciente y de rápido crecimiento que estudia un rango más amplio de agentes, desde los simples hasta los moderadamente inteligentes. El énfasis ha cambiado sutilmente de la deliberación a la acción; del razonamiento a la acción remota. La gran diversidad de aplicaciones y enfoques de los agentes de software actuales es un indicador de que se están convirtiendo en una tendencia con mucha penetración en todas las áreas.

Cuando los primeros investigadores de la corriente 1 se rindieron, la tarea la retomaron los nuevos investigadores en inteligencia artificial distribuida, robótica, vida artificial, cómputo de objetos distribuidos, interacción de humano-computadora, interfaces inteligentes y adaptables, filtrado y búsqueda inteligente, recuperación de datos, adquisición de conocimiento, programación para usuarios, programación por demostración y una lista creciente de otras áreas de estudio.

Como se han creado "agentes" de muchos tipos, el término se ha utilizado indiscriminadamente sin consenso de lo que significa. Algunos programas son llamados agentes simplemente porque pueden calendarizar tareas a realizar en un host remoto, cuando esto se lleva haciendo mucho tiempo bajo el nombre de "tareas por lotes". Algunos otros porque realizan tareas de bajo nivel como revisión de sectores de disco duro, mientras son comandados con un lenguaje de programación de alto nivel. Algunos son llamados agentes porque abstraen y encapsulan información. Otros porque sirven de mediador entre las personas y los programas, actuando como "asistentes inteligentes". Otros porque pueden migrar por sí mismos de una computadora a otra. Algunos son llamados así porque pueden interpretar un ACL (lenguaje de comunicación de agentes) y otros porque pueden manifestar intención y otros aspectos de un "estado mental".

¿Que es un agente de software?

Esta sección resume las dos definiciones principales de lo que es un agente: agente como un miembro de un grupo y agente como una descripción.

"Agente" como un miembro de un grupo.

Como se ha comentado anteriormente, una de las características principales de la investigación y desarrollo reciente de agentes de software es la poca similitud entre los diferentes enfoques. Aunque hay cosas comunes entre ellos, dar una definición definitiva de agente es muy difícil. El agente inteligente para unos, es un objeto inteligente para otros, y el objeto inteligente de hoy será el "programa tonto" del mañana. La distinción principal está en las expectativas y nuestro punto de vista personal. El diccionario define agente como "el que actúa o tiene el poder o autoridad para actuar por alguien o representarlo" o "los medios por los cuales algo es hecho o encausado; un instrumento". El término se deriva del presente participio del verbo en latín *agere*: conducir, liderar, actuar o hacer. Como en el sentido cotidiano, esperamos que el agente de software actúe en representación de alguien para realizar una tarea en particular que le ha sido delegada. Y como es poco práctico dictar cada detalle de la acción a realizar, se desea que nuestros agentes infieran todos los pasos necesarios desde una orden simple. Los agentes solamente podrán hacerlo si "saben" algo del contexto en el cual se realiza la solicitud. Los mejores agentes, entonces deberán no sólo poder realizar las tareas correctamente, sino también tomar en cuenta las peculiaridades del usuario y la situación. "Un experto en el área y un experto en nosotros" como lo expresa Negroponte **[3]**.

"De hecho el concepto de agente en humanos que asisten a humanos se caracteriza en que la experiencia se mezcla con el conocimiento de la persona a quien se está ayudando. Un buen agente de viajes mezcla conocimiento de hoteles y restaurantes con conocimiento de lo que le pudiera agradar al cliente. Un agente de bienes raíces crea un modelo de los gustos del cliente basado en mostrarle una sucesión de casas que complacen de cierta manera al cliente, variando los grados de acierto o error. Ahora podemos imaginar un agente que conteste el teléfono, un agente de noticias, un agente de manejo de correo electrónico. Lo que todos tienen en común entre ellos es la habilidad de modelar al usuario".

Mientras que la descripción anterior descarta a muchos supuestos agentes. Aun hay controversia. "Recientemente un gran número de desarrolladores han rebautizado componentes existentes de su software como agentes, porque realizan autónomamente tareas sin que haya una interacción con el usuario o "contrato social". Esto en vez de crear confianza, hace dudar al usuario si es confiable delegar inclusive las tareas más triviales" **[5]**.

Shoham **[6]** provee un ejemplo práctico ilustrando que aunque casi cualquier cosa puede ser definido como un agente, no siempre conviene hacerlo: "es perfectamente coherente tratar un interruptor de un foco como un agente con la capacidad de transmitir corriente a voluntad, que invariablemente transmitirá la corriente cuando nosotros queramos que sea transmitida y no de otra manera; el mover el interruptor es simplemente nuestra manera de comunicarle nuestros deseos. Sin embargo, aunque esto es coherente, no nos explica nada, porque entendemos el mecanismo para tener una descripción más simple y mecánica de su comportamiento".

Agente como una descripción

Una definición más especifica de "agente de software" que muchos investigadores pueden encontrar aceptable es que es: "una entidad de software que funciona continua y autónomamente en un ambiente en particular, en algunos casos sin que existan otros agentes o procesos" **[6]**.

El requerimiento de continuidad y autonomía se deriva de nuestro deseo de que un agente sea capaz de realizar actividades de una manera inteligente y flexible, que sea sensible a cambios en el ambiente sin requerir guía o intervención humana. Además esperamos que el agente pueda desenvolverse en un ambiente donde habiten otros agentes o procesos con los que sea capaz de comunicarse y quizás moverse de un lugar a otro mientras lo hace.

La mayoría de los agentes de software de hoy en día son como bestias frágiles y de propósito especifico, ninguno de los cuales puede hacer la mayoría de las cosas que se describieron anteriormente de una manera genérica. Por lo cual el término "agente de software" puede ser mejor visto como un "término sombrilla" que cubre un rango de otros tipos de agentes más específicos y limitados **[4]**.

Aunque como individuos las capacidades de un agente puedan ser restringidas, cuando se unen varios agentes pueden extender las funcionalidades de un sistema. Consistentemente con los requerimientos de un problema particular, cada agente puede poseer en mayor o menor medida, atributos como los enumerados por Etzioni y Weld **[7]** y Franklin y Graesser **[8]**:

Reactividad: la habilidad para sentir y actuar selectivamente.

Autonomía: que sea dirigido a metas, proactivo y auto iniciado.

Colaboración: que pueda trabajar en conjunto con otros agentes para lograr un objetivo común.

Habilidad de comunicación a un nivel de conocimiento: la habilidad de comunicarse con personas u otros agentes con un lenguaje parecido al humano.

Capacidad de inferenciación: que pueda actuar en tareas abstractas usando conocimiento anterior de metas generales y preferencia de métodos para alcanzar mayor flexibilidad.

Continuidad temporal: la persistencia de la identidad y su estado a través de periodos de tiempo.

Personalidad: la capacidad de manifestar atributos de carácter como emociones.

Adaptabilidad: ser capaz de aprender e improvisar si tiene suficiente experiencia.

Movilidad: que sea capaz de migrar por sí mismo de un host a otro.

Para proveer una manera más simple de caracterizar el espacio de los tipos de agentes que resultaran si uno intenta cada combinación de posibles atributos, se ha propuesto diversos esquemas de clasificación y taxonomias en la comunidad de investigadores.

Por ejemplo, los investigadores de inteligencia artificial distinguen entre nociones de agencia suave y dura, la segunda son los agentes que son diseñados para poseer cualidades explicitas mentales o emocionales.

La comunidad de Inteligencia artificial distribuida ha caracterizado a los agentes por su capacidad de resolver problemas:

“Un agente reactivo cambia con su entorno o con los mensajes de otros agentes, un agente intencional puede razonar con sus intenciones y conocimientos para crear planes de acción, y ejecutar esos planes, además de las anteriores, un agente social posee modelos explícitos de otros agentes” **[9]**.

Gilbert **[10]** describe en un artículo para IBM a los agentes inteligentes en términos del espacio definido por sus tres dimensiones: agencia, inteligencia y movilidad. Figura **[1.1]**

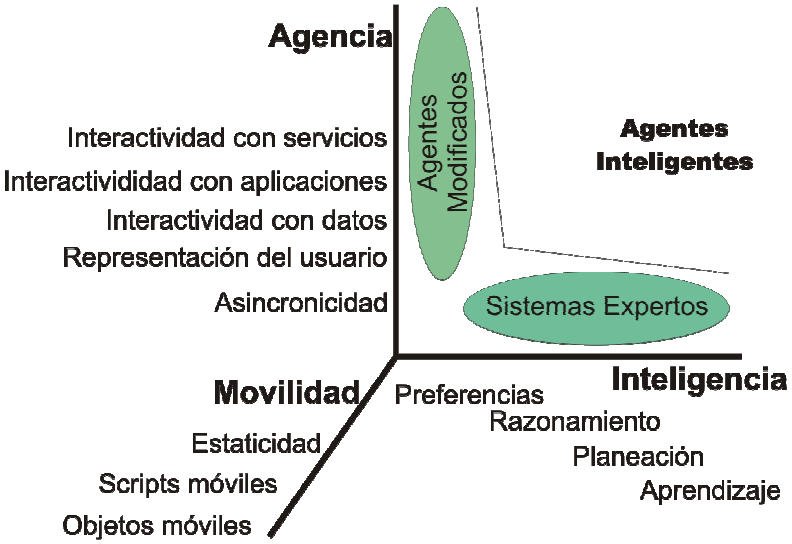

Figura 1.1 Alcance de un agente

"Agencia es el grado de autonomía y autoridad en el agente, y puede ser medida al menos cualitativamente por la naturaleza de la interacción entre el agente y otras entidades en el sistema. Como mínimo un agente debe de correr asincrónicamente. El grado de agencia es aumentado si un agente representa a un usuario de alguna manera. Un agente mas avanzado puede interactuar con datos, aplicaciones, servicios u otros agentes.

Inteligencia es el grado de razonamiento y comportamiento aprendido; la habilidad del agente de aceptar la declaración de metas del usuario y realizar las tareas relegadas a él. Como mínimo, debe de haber declaración de preferencias. Niveles más altos de inteligencia incluyen un modelado del usuario y razonamiento. Un poco mas arriba en la escala de inteligencia están los sistemas que aprenden y se adaptan a su ambiente, ambos en términos de los objetivos del usuario, y en términos de los recursos disponibles para el agente.

Movilidad es el grado en el cual los agentes por si mismos viajan a través de la red. Desde scripts móviles que pueden ser escritos en una máquina y enviados a otra para su ejecución, hasta los objetos móviles que son transportados de máquina en máquina en medio de su ejecución acarreando un estado de datos acumulado con ellos"

Nwana **[4]** propone una tipología de agentes que identifica otras dimensiones de clasificación. De acuerdo a ésta los agentes deben de ser clasificados de acuerdo a:

Movilidad: estáticos o móviles.

Presencia de un modelo simbólico de razonamiento: como deliberativos o reactivos.

Atributos primarios ideales: como autonomía, cooperación, aprendizaje. De estas características, Nwana deriva cuatro tipos de agentes: colaborativos, de aprendizaje colaborativo, de interfaz y los inteligentes.

Roles: como de información, Internet, etc.

Filosofías híbridas: que pueden combinar uno o más enfoques en un sólo agente.

Atributos secundarios: como versatilidad, benevolencia, veracidad, confiabilidad, continuidad temporal, habilidad para fallar controladamente y cualidades mentales y emocionales.

Después, Nwana describe la investigación actual en siete categorías: agentes colaborativos, agentes de interfaz, agentes móviles, agentes de información/Internet, agentes reactivos, agentes híbridos y agentes inteligentes. Figura **[1.2]**

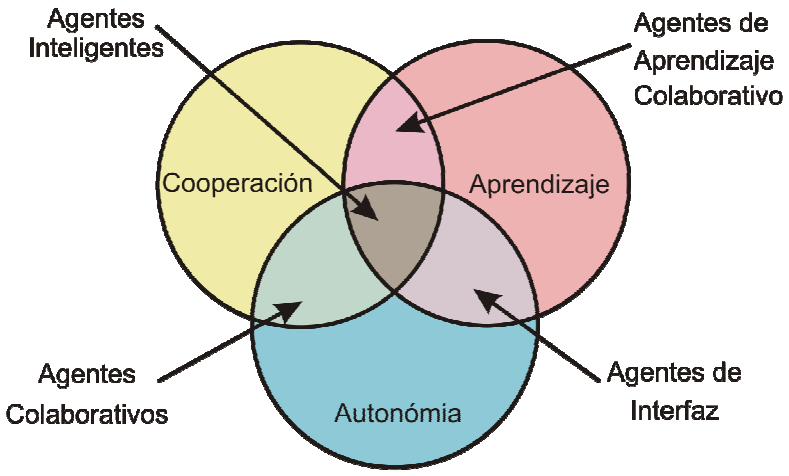

Figura 1.2 Tipología de agentes

Franklin y Graesser **[8]** dan su definición: "Un agente autónomo es un sistema situado dentro de un ambiente que es sensible a ese ambiente y actúa en él, a través del tiempo, en búsqueda de sus metas hasta afectar lo que cense en el futuro". Observando que por esta definición aun un termostato puede calificar como un agente, ellos discuten varias

propiedades de agentes y ofrecen la taxonomía de la figura **1.3** para cubrir la mayoría de los ejemplos encontrados en la literatura. Bajo esta clasificación inicial ellos sugieren que los agentes pueden ser catalogados en estructuras de control, entornos (por ejemplo, base de datos, sistema de archivos, red, Internet), lenguaje en el cual están escritos y aplicaciones.

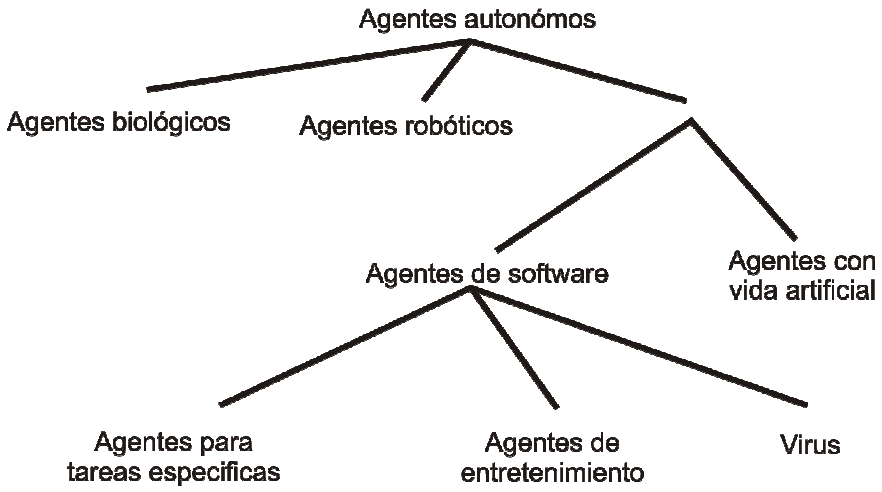

Figura 1.3 Taxonomía de agentes

Finalmente, Petrie **[11]** propone varios intentos de la comunidad de investigadores para distinguir agentes de otros tipos de software. Primero señala la dificultad para definir inteligencia y autonomía. Entonces muestra que la mayoría de los "agentes" basados en Web de búsqueda y filtrado, aunque son útiles, son "esencialmente mecanismos de respuesta para una consulta" y son propiamente descritos por un término más simple: "servidor". Similarmente, los "procesos móviles" describen mejor el término "agente móvil" para aquellos applets en Java los cuales su única función es "permitir a los procesos correr de manera segura en hosts foráneos".

Con el tiempo y la experiencia se definirá el término "agente". Como el uso de muchos otros términos computacionales como "escritorio" o "puntero de mouse", se inicia como una metáfora pero termina definiendo algún software en concreto. Mientras crece la exposición del público en general a implementaciones útiles y técnicamente viables de agentes de software, el término se vuelve más entendible porque se observan muchos ejemplos de él, o cae en desuso porque describe un concepto que ya no será apropiado.

Agentes Móviles

Los agentes móviles **[16]** son un paradigma que se deriva de dos disciplinas diferentes **[15]**. El primero es la inteligencia artificial, que creó el concepto de agente **[17]** y el segundo son los sistemas distribuidos, el cual define el concepto de la movilidad de código **[12]**

De acuerdo con las definiciones estándar, los agentes móviles son similares a los agentes no-móviles (autónomos, reactivos, proactivos, y sociables), pero además se pueden desplazar o migrar entre plataformas para cumplir con las tareas asignadas.

Desde el punto de vista de los sistemas distribuidos, un agente móvil, es un programa con una identidad única que puede mover su código, datos y estado entre máquinas conectadas a través de una red. Para lograr esto, los agentes móviles son capaces de suspender su ejecución en cualquier momento y continuar una vez que se encuentren en otro lugar.

A continuación referiremos a los agentes móviles en relación con otros paradigmas clásicos **[12]**

- Cliente-Servidor. El paradigma más usado, donde los servicios son ofrecidos por un servidor y consumidos por uno o más clientes, usualmente de manera remota.
- Ejecución Remota: Un componente manda código a otro componente para ejecución remota ya sea por decisión propia, por solicitud del componente remoto, o quizás como parte un contrato pre-existente. Una vez ejecutado el código, el componente normalmente regresará algún resultado al componente origen.
- Agentes Móviles: Un componente se manda a sí mismo(o a otro si se le es permitido) a un host remoto para ser ejecutado. El componente se mueve con su código, sus datos, y quizá su estado intacto. Los motivos pueden ser similares a los anteriores casos, pero lo más común es que el componente (agente móvil) decida por si mismo que desea moverse a un lugar distinto al actual.

Un agente móvil como se muestra en la figura **[1.4],** consiste en 3 partes: Código, estado y datos, el código es lo que se ejecuta cuando migra a una plataforma. En el caso más sencillo solamente se migra el código. El estado de ejecución de los datos en el entorno del agente, incluyendo el contador de programa y la pila de ejecución y migrará también en agentes que usan una migración fuerte. Los datos consisten en las variables usadas por los agentes,

como la información recopilada, también llamada conocimiento, identificadores de archivo, etc. En la migración débil esta parte es estrictamente necesaria porque el código del agente es construido como una máquina de estados con las variables requeridas para mantener el estado de la información.

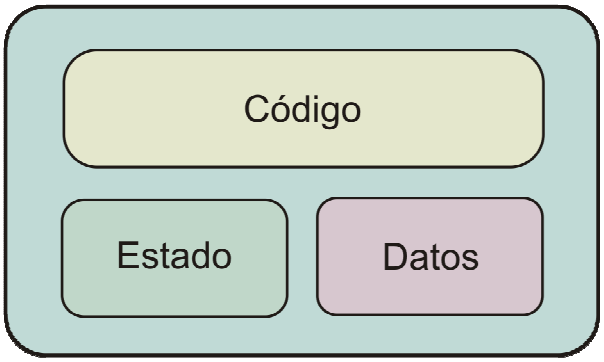

Figura 1.4 Estructura de un agente móvil

Algunas ventajas y desventajas de los agentes móviles.

Ha habido muchos debates acerca de las varias ventajas y desventajas de los agentes móviles, usualmente en comparación con los agentes no-móviles. A continuación se muestra un extracto de estas:

Algunas de las ventajas típicas son:

- *Procesamiento asíncrono e independiente:* una vez que ha migrado a una nueva plataforma, los agentes no tienen que contactar al agente creador para poder hacer una tarea. Ellos pueden mandar solamente los resultados como respuesta. Esto es especialmente útil cuando consideramos los dispositivos móviles con recursos limitados; Un agente puede ser migrado a otra máquina para realizar tareas complejas y periódicamente regresar algún resultado.
- *Tolerancia a fallos***:** Pueden lidiar con condiciones erróneas moviéndose a una plataforma alternativa cuando se detecten problemas. De la misma manera si los destinos de migración no se encuentran disponibles, una plataforma intermediaria puede actuar como host temporal. Esto los hace bastante prácticos en entornos hostiles y cambiantes.
- *Aplicaciones para grandes cantidades de datos***:** Los agentes móviles son ideales para aplicaciones que necesitan procesar grandes cantidades de datos remotos, los

agentes móviles pueden moverse a los datos, en vez de que los datos se muevan al host lo cual lo hace una opción mucho más eficiente.

Pero los agentes móviles también tienen algunas desventajas, Como es descrito por Mir en **[13]**, los más relevantes son:

- *Escalabilidad y desempeño:* aunque los agentes móviles reducen la carga de trabajo de la red, también tienden a incrementar la carga de procesamiento, esto es porque usualmente están programados en lenguajes interpretados y muchas veces tienen que obedecer estándares rigurosos de interoperabilidad que pueden generar sobrecarga de procesamiento de datos.
- *Portabilidad y estandarización***:** los agentes no pueden operar entre ellos si no siguen estándares de comunicación comunes. La adopción de estos estándares como el OMG MASIF (*Mobile Agent System Interoperability Facility*) o el FIPA son necesarios especialmente para la movilidad entre plataformas.
- *Seguridad*: el uso de los agentes móviles puede traer problemas de seguridad. Cualquier código móvil muestra riesgos potenciales y deben de ser cuidadosamente autenticados antes de la invocación.

Migración fuerte y migración débil.

En los sistemas de agentes móviles **[14] [15]** los dos tipos elementales de migración son: la migración débil y la migración fuerte.

La migración fuerte es más compleja. Es cuando la ejecución de un agente es paralizada, entonces el agente migra y después continúa la ejecución a partir de la siguiente instrucción. Esta técnica requiere del almacenamiento y la protección del estado del agente durante el proceso de migración. La implementación de esta técnica puede ser compleja porque requiere acceso a los parámetros internos de la ejecución del agente, que generalmente solo están disponibles al sistema operativo y que son muy dependientes de la arquitectura.

La migración débil es más simple, ya que no migra el estado del agente. La ejecución del agente siempre se reinicia desde el inicio del código, este tipo de migración requiere que el

agente sea implementado como una maquina de estados finitos para que su estado sea conservado.

Itinerarios de migración.

Un itinerario de migración define los lugares que el agente móvil visitará para completar un conjunto de tareas. Existen dos tipos básicos de itinerarios:

- *Itinerarios estáticos***:** son determinados en el momento de la creación del agente sin la posibilidad de modificación durante la ejecución del agente.
- *Itinerarios dinámicos***:** son determinados durante la ejecución del agente, de acuerdo a las necesidades y deseos. Pueden crearse métodos híbridos a partir de estos dos tipos **[12]**

I.2 Objetos de Aprendizaje

Una tendencia de la enseñanza contemporánea es la universalidad y alcance de la educación. Terminar con los esquemas rígidos de la educación estándar enclaustrada en los muros de un edificio para alcanzar a todas las personas que la requieran. Aprovechar la plétora de medios de información que se tienen disponibles hoy en día no solamente es una ventaja sino una obligación. Es así como surgen términos e ideologías como e-learning o e-learning2.0 **[18]**.

Las teorías de aprendizaje conocidas hasta ahora fueron basadas en el aprendizaje en aulas, y resultan insuficientes para abordar la nueva acepción de aprendizaje actual, con toda su complejidad, su carácter informal y contextual que caracteriza al conocimiento en nuestro entorno.

En definitiva, aprender será ahora instanciar patrones de conectividad en la mente, en la red neural. No elaboramos significado ni construimos significado sino que lo hacemos crecer, incrementamos el valor del significado de forma casi infinita mediante la propia conectividad. Los roles de tutor y alumno se diluyen en este tipo de aprendizaje. En palabras de Downes **[18]**, “cambian, de tratar sobre la realidad, de verificar la realidad, a crearla”.

El rol del tutor cambia, de director de grupos hasta coordinador o moderador de comunidades, En este sentido, un factor importante será la motivación, el fomento de la curiosidad, la oferta de los contenidos *on demand* (a demanda). En la línea también del "movimiento de competencia", otra importante tarea del tutor será la de potenciar los logros, la autoestima para la participación.

Por un lado estas son herramientas formativas desde un punto de vista pedagógico y tecnológico. Pedagógicamente se trata de hacer una pedagogía de transmisión del saber más allá de la propia inserción de contenidos en la red (la actual sociedad de la información debe convertirse en sociedad del conocimiento). Tecnológicamente, las tecnologías disponibles como Internet y LMS (*Learning Managment Systems*) darán soporte de este nuevo concepto de educación. Por otro lado es uso formativo "una fuente de servicios para alcanzar su cometido formativo" **[19]**. Además, etimológicamente, e-learning es aprendizaje electrónico: todo proceso formativo que uso cualquier tipo de tecnología electrónica no tradicional. Desde este punto de vista, el profesorado lleva haciendo uso del e-learning desde la inclusión de los aparatos de audio, visuales y audiovisuales.

En *e-learning* tradicionalmente los cursos están construidos como una sola entidad la cual no puede ser dividida. Esto impide compartirlos, es decir, las partes que integran un curso en línea no pueden ser compartidas para formar otros cursos; por lo tanto la producción del material didáctico es redundante y sus costos se incrementan.

El que los cursos sean producidos como una sola entidad también afecta la calidad de desarrollo, así como accesibilidad, interoperabilidad, durabilidad y reutilización de los materiales y los hace dependientes de las plataformas en los que han sido desarrollados.

Estos efectos negativos, lejos de beneficiar o de pasar desapercibidos afectan a la comunidad del aprendizaje electrónico. Es debido a eso, que es importante contar con medidas que disminuyan esos problemas y garanticen la calidad en la producción de contenido de aprendizaje. Si los cursos en línea fueran creados con pequeñas piezas de información digital a modo de que puedan ser unidas para generar un módulo, unidad, tema o un curso completo, y que estas piezas fueran reutilizables y modulares de forma que pudieran ser intercambiables entre varios diseñadores de contenido para no volver a crear contenido con esas características se reducirán los efectos negativos que se tienen ahora.

La popularización de la divulgación de contenidos educativos a través de Internet comienza al inicio de la década de los años 90. Poco después, en 1994, Wayne Hodgins marca una pauta importante al introducir el término “Objeto de Aprendizaje”. La aparición de este concepto ha suscitado desde entonces una creciente expectación, aunque no libre de polémica debido a la falta de acuerdo en las definiciones básicas adoptadas por diferentes autores. Uno de los temas más importantes en la actualidad dentro del ámbito del aprendizaje electrónico tiene que ver con la idea de la reutilización para los contenidos didácticos. Es, de hecho, uno de los principios que fundamentan el concepto de objeto de aprendizaje **[20]** y su importancia es crucial. Es importante resaltar que la reutilización atañe principalmente al diseño de instrucción y no a los formatos digitales o a la estructura de los contenidos. Las características que determinan que un objeto de aprendizaje sea apto en una variedad de contextos educativos son esencialmente pedagógicas.

A continuación daré algunas definiciones que nos ayudarán a comprender un poco más el concepto de Objeto de Aprendizaje.

“Un objeto de aprendizaje es una unidad de aprendizaje independiente y auto-contenida que esta predispuesta a rehusarse en múltiples contextos de instrucción” **[20]**.

- Una entidad informativa digital desarrollada para la generación de conocimiento, habilidades y actitudes que tienen sentido en función de las necesidades del sujeto y que se corresponde con la realidad.

"Un recurso digital que puede ser reusado para ayudar en el aprendizaje." **[21]**.

"Una entidad, digital o no digital, que puede ser usada para aprendizaje, educación o entrenamiento" **[22].**

"Fragmentos interactivos de e-learning, orientados a la Web y diseñados para explicar un objetivo de aprendizaje independiente" **[23]**.

"Una entidad digital, autocontenible y reutilizable, con un claro propósito educativo, constituido por al menos tres componentes internos editables: contenidos, actividades de aprendizaje y elementos de contextualización. A manera de complemento, los objetos de aprendizaje han de tener una estructura (externa) de información que facilite su identificación, almacenamiento y recuperación: los metadatos" **[24]**.

Así concebimos a los OA como piezas de conocimiento pequeñas y almacenadas independientemente en una base de datos, estas piezas son auto-contenidas, intercambiables, interoperables, durables y accesibles que pueden ser una imagen, una página web, una presentación de diapositivas, documento de texto, archivos de audio en formato MP3 o WAV, una aplicación, etc. Con ellos se reducirán los efectos mencionados, debido a que los OA brindan reutilidad e interoperabilidad, brindando consecuentemente un ahorro importante en los costos de producción y no solamente en ello, si no también, el tiempo empleado para producir contenido educativo, sumando a la lista de beneficios que ofrece la interoperabilidad y la reutilidad, se puede mencionar la calidad de contenido, accesibilidad a la pieza de información y durabilidad de esta.

Algunos ejemplos de recursos que pueden componer un objeto de aprendizaje son descritos en la figura 1.5.

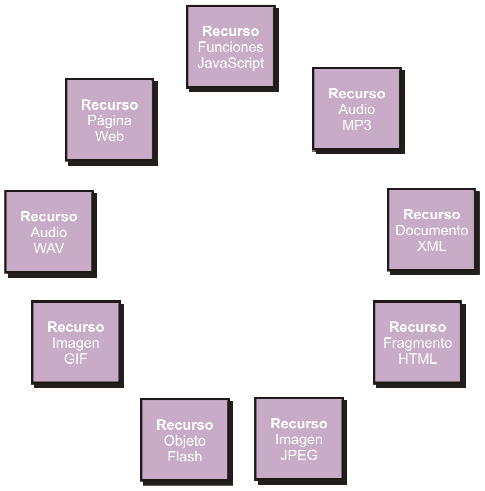

Figura 1.5 Recursos de un objeto de aprendizaje

Se comprende inmediatamente el ahorro que este planteamiento supone para las instituciones educativas y productoras de contenidos. Si los OA se construyen correctamente y además se almacenan y se catalogan de una manera practica y funcional podrán ser aprovechados con más facilidad, más allá el alcance y del contexto que habrían logrado si hubieran seguido otros planteamientos tradicionales menos flexibles. Detrás de este pensamiento se esconde la filosofía que da lugar a los denominados repositorios de OA. Estas unidades mínimas se combinan y deben funcionar dentro de un LCMS (*Learning Content Management System ó* Sistema de Gestión de Contenido de Aprendizaje).

Los OA proporcionan estos beneficios por que están construidos de tal manera que los hace independientes a las plataformas de cómputo existentes, así como de los sistemas operativos o LMS (*Learning Management Systems*). Pero aún existen inconvenientes al crear los OA, pues deben de ser construidos en base a estándares para poder proporcionar los beneficios mencionados y reducir los problemas que existen, de lo contrario será una opción inútil o una propuesta que no funcione. En el medio de los OA, existen algunas organizaciones que se encargan de crear especificaciones que normalizan su construcción para su correcta implementación y uso adecuado; algunas de estas organizaciones son *Aviation Industry CBT Committee* (AICC), El *Institute of Electrical and Electronics Engineers* (IEEE), el *IMS Global Learning Consorsium*, *Advanced Distributed Learning* (ADL) y *Alliance for Remote Instructional and Authoring and Distribution Networks for Europe* (ARIADNE) entre otros. AICC es una asociación internacional basada en la tecnología que se dedica a elaborar pautas para el desarrollo, la entrega y evaluación de tecnologías de formación profesional. El AICC cuenta con una API (Aplication Program Interface) la cual le permite tener interacción al contenido con el sistema de administración de aprendizaje**.**

El IEEE por su parte, es un grupo multinacional que se encarga de desarrollar estándares internacionales para sistemas eléctricos, electrónicos, computacionales y de comunicación. Está organizado en distintos comités, y uno de ellos es el comité para los Estándares de la Tecnología de Aprendizaje (LTSC por sus siglas en inglés *Learning Technology Standars Committee*). Una de sus especificaciones más conocida hace referencia a los metadatos de los OA (LOM por sus siglas en inglés *Learning Object Metadata*); es decir, lo que intenta LTSC es facilitar el desarrollo y la reutilización de contenidos.

Las ventajas que se obtienen en la estandarización son las siguientes **[25]**:

- Incrementar la cantidad y la calidad de los contenidos
- Posibilitar el intercambio de cursos (compra-venta)
- Personalizar los contenidos y reutilizarlos.
- Asegurar la compatibilidad con diferentes plataformas.
- Garantizar el intercambio de contenidos entre diferentes entornos virtuales de formación.
- Permitir la búsqueda de contenidos por toda la red.
- Fomentar la profesionalización en la elaboración de contenidos.
- Iniciar sistemas de compra-venta de contenidos.
- Aumentar la eficiencia de los contenidos en línea desarrollados, además de facilitar su gestión.
- Garantizar la viabilidad futura de la inversión en el aprendizaje electrónico de una empresa, lo que impide que la empresa no dependa de un único LMS.
- Aumentar la oferta de cursos disponibles en el mercado
- Los productos no quedarán obsoletos a corto plazo

Algunos estándares que se encuentran en el mercado actualmente son SCORM, versiones 1.2 y 2004, *IMS Global Learning Consortium*, LOM, *Dublin Core*, *Edusource*, entre otros. Entre otras especificaciones para los OA por parte de los estándares internacionales especifican los esquemas para crear los metadatos en el formato XML.

El estándar SCORM indica que los metadatos acerca del OA deberán de estar en un paquete junto con el OA. Este paquete será en formato ZIP; en donde el contenido de él serán: el OA, el archivo XML que contenga los metadatos (que debera llamarse imsmanifest.xml y es denominado Archivo Manifiesto) y una organización como se muestra en la figura **[1.6]**.

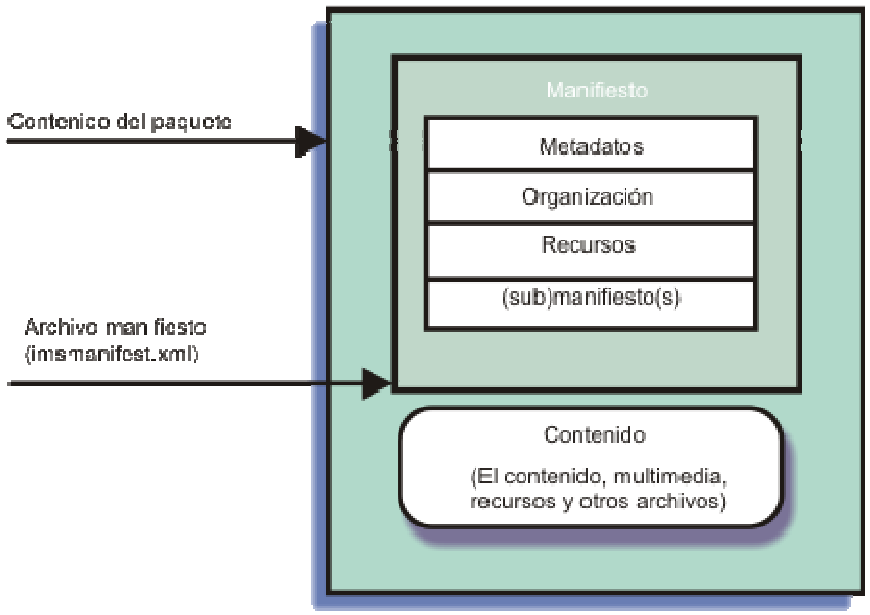

Figura 1.6 estructura de un OA SCORM

En SCORM, el LMS determina que material será entregado y cuando. También determina como medir el progreso y desempeño del alumno a través del contenido de aprendizaje.
SCORM apoya la noción de contenido de aprendizaje formado por objetos de pequeño contenido y reusable para formar unidades de instrucción tales como cursos, módulos, capítulos, etc.

I.3 Metadatos

Literalmente los metadatos para los OA son datos acerca de los OA, que informan sobre las características de los OA existentes, de modo que los usuarios sean capaces de entender lo que representan y como lo representan. Para ello la información incluida en los metadatos describe: la fecha de los datos, el contenido, la extensión que cubren, el sistema de referencia espacial, el modelo de representación espacial de los datos, su distribución, restricciones de seguridad y legales, frecuencia de actualización, calidad, etc. Los metadatos de un OA son una parte esencial en la administración de un conjunto de ellos, tanto en un repositorio donde sólo se almacenan o en un mercado donde se realizan transacciones. De igual manera facilitan la búsqueda y localización de los objetos de

aprendizaje. La selección de los metadatos que se incorporan a los OA es una decisión importante porque va a determinar su eficaz localización y en consecuencia su capacidad de reutilización por otras personas. Para la construcción de los metadatos se utiliza el lenguaje XML (Xtensible Markup Language) y para su correcta definición existen diversas organizaciones que se encargan de estandarizar la creación de estos elementos.

Capitulo II. Diseño del Sistema

En este capítulo se define la estructura y comportamiento de los agentes que conforman el sistema. Para el diseño del sistema se ha utilizado el proceso unificado de desarrollo de software, en particular se utiliza el lenguaje de modelado (UML) ya que, aunque existen metodologías para el diseño de sistemas multiagentes, entre las que podemos citar a: AUML, AML, MESSAGE, GAIA, no son necesarias ya que se enfocan fundamentalmente a la comunicación entre sistemas multi-agentes y la especificación de mensajes y ontologías.

II.1 Tareas del sistema de Búsqueda y Recuperación de Objetos de Aprendizaje.

El sistema de Búsqueda y Recuperación de Objetos de aprendizaje actúa como interfaz del usuario proveedor y el mercado de Objetos de aprendizaje (MOA).

Recordemos que el MOA es el lugar donde se controla la compra y venta de Objetos de aprendizaje, donde los proveedores anuncian sus OA y los consumidores intentan obtenerlos. Es mediante un proceso de negociación que ambas partes acceden a la entrega del OA.

El Agente de Búsqueda y Recuperación de Objetos de Aprendizaje (ABROA) se encuentra asistiendo al usuario proveedor en la administración de sus OA, encontrando, empaquetando y publicando OA existentes en la máquina del proveedor. Y cuando reciba una solicitud del agente administrador del MOA creará a un agente móvil (agente cartero) que transportará dentro de sí mismo el objeto de aprendizaje solicitado a la máquina del consumidor y lo depositará allí. Considerando esto se pueden definir las siguientes tareas:

Tareas del ABROA:

- Buscar posibles OA en la máquina del proveedor.
- Comprimir estos posibles OA de acuerdo al estándar SCORM 2004.
- Publicarlos en la base de datos del proveedor.
- Verificar OA existentes.

- Mantener actualizada la base de datos del proveedor.
- Recibir peticiones de entrega de OA

Tareas del agente cartero:

- Leer y encapsular OA
- Moverse a la máquina cliente
- Desencapsular y escribir el OA.

II.2 Diagrama de casos de uso del sistema

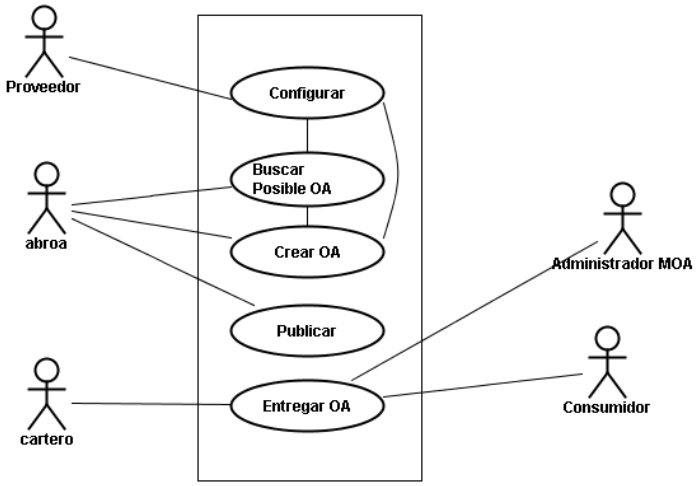

Figura 2.1 Diagrama de Casos de Uso del Sistema de Búsqueda y Recuperación de OA

Caso de uso Configurar
Actor: proveedor

Descripción: El usuario proveedor editará un archivo de configuración (conf.xml) para definir sus preferencias: nombres de archivo de inicio, carpetas de búsqueda y características de creación de OA.

Caso de uso Buscar Posible OA
Actor: Agente ABROA

Descripción: El agente ABROA busca en el host posibles OA y todos los recursos de los que depende.

Caso de uso Crear OA
Actor: Agente ABROA

Descripción: El agente ABROA creará OA con el estándar SCORM 2004

Caso de uso Publicar
Actor: Agente ABROA

Descripción: el agente ABROA publicará todos los OA disponibles en la base de datos del proveedor.

Caso de uso Entregar OA
Actores: cartero, Administrador MOA, Consumidor

Descripción: El agente Administrador del MOA enviará un mensaje de envío de un OA. El agente abroa creará un agente cartero que será enviado a la máquina del consumidor conteniendo el OA solicitado.

II.3 Diagramas de clases general

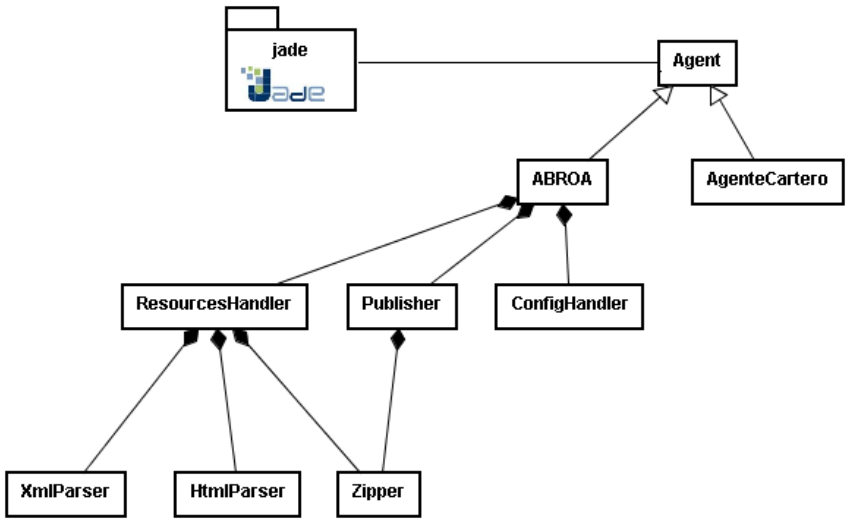

Figura 2.2 Diagrama de clases general del Sistema

Muestra las relaciones principales de la clase ABROA. Esta clase se deriva de la clase *Agent*, perteneciente al paquete de clases JADE. A su vez se asocian las clases *ResourcesHandler, Publisher y ConfigHandler*, que se encargan de encontrar el OA, publicar OA en la base de datos del proveedor y la configuración del sistema respectivamente. El *ResourcesHandler* y el Publisher deben trabajar con archivos

comprimidos por lo cual utilizan una instancia de la clase *Zipper*. Además el *ResourcesHandler* analizará archivos xml y html por lo cual necesita las clases *XmlParser* y *HtmlParser* para cada tipo de archivo. Otra Clase que se deriva de la clase Agent es la Clase AgenteCartero que será la encargada de entregar un OA al consumidor.

II.4 Diagrama de clases detallado.

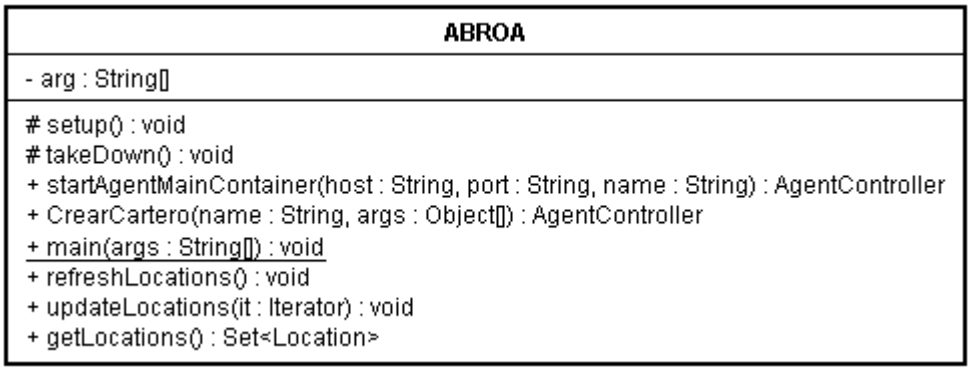

Figura 2.3 Clase ABROA

En la figura **[2.3]**, se muestran los métodos y atributos del Agente ABROA, esta clase es la que se encarga de asistir al usuario proveedor en la búsqueda y recuperación de objetos de aprendizaje en su máquina. La inicialización de la plataforma jade y de la creación del agente en la plataforma lo hace el método *main()* invocado por el usuario. El método *setup()* inicializa el agente y si el usuario indicó inicializar una interfaz gráfica de usuario (GUI) será inicializada, de lo contrario se iniciarán los comportamientos automáticos de búsqueda, publicación y envío de OA.

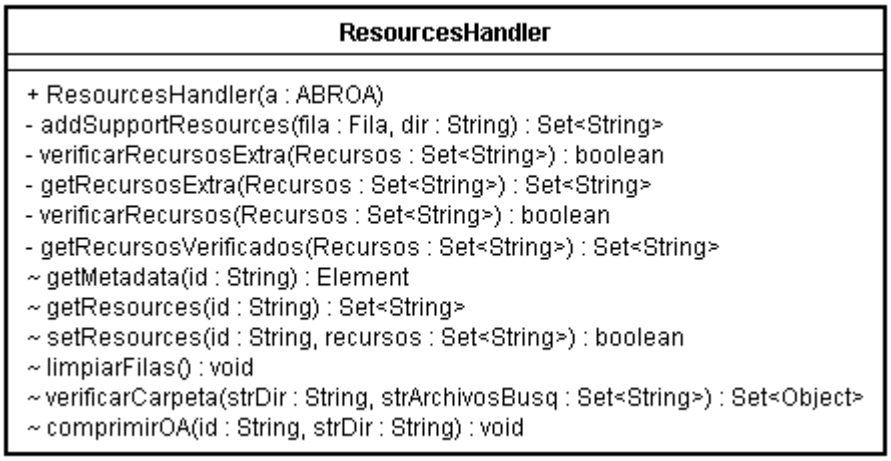

Figura 2.4 Clase ResourcesHandler

La figura **[2.4]** que muestra la clase *ResourcesHandler* es la encargada de acceder y administrar a todos los elementos que compondrán al OA (recursos), manejar la lista de OA

encontrados. Tiene funciones para verificar y añadir recursos, y crear OA bajo el estándar SCORM 2004.

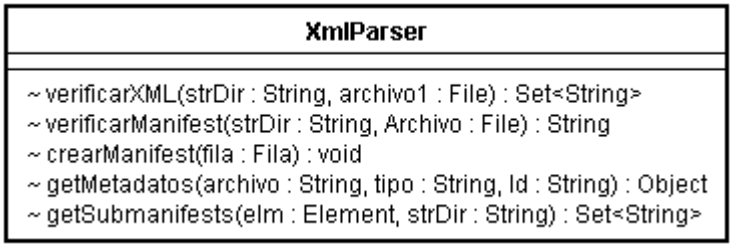

Figura 2.5 Clase XmlParser

La figura **[2.5]** muestra la clase *XMLParser* la cual obtiene los recursos referenciados en el archivo manifiesto de un OA. Crea manifiestos sobre la base de los recursos encontrados.

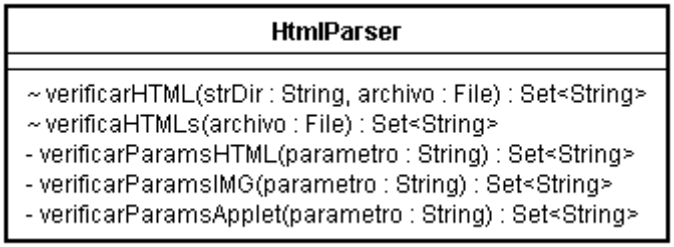

Figura 2.6 Clase HtmlParser

Como se muestra en la figura **[2.6]** la clase *HtmlParser* es utilizada para obtener recursos que están referenciados dentro de etiquetas HTML. Dentro del primer archivo puede haber referencia a otros archivos HTML, así que la función *verficaHTMLs()* se llama recursivamente hasta encontrar todos los recursos necesarios para componer el OA. Además cuenta con funciones especiales para extraer recursos referenciados dentro de etiquetas especiales como applets, javascript y object flash.

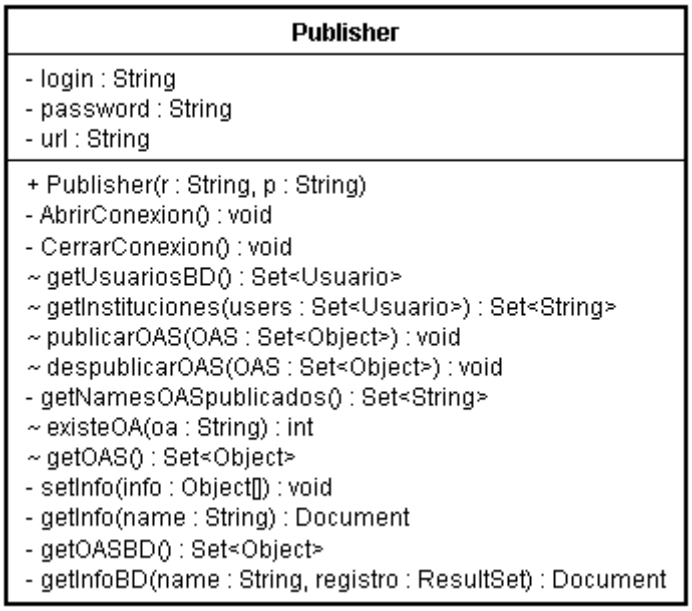

Figura 2.7 Clase Publisher

La clase *Publisher* mostrada en la figura **[2.7]** se utiliza para mantener actualizada la base de datos con los OA existentes. Esto lo logra con la ayuda de archivos xml de información de cada uno de los OA disponibles. La razón por la cual se utilizan archivos extra en vez de usar los manifiestos dentro del OA es que esta información solamente es útil al usuario proveedor para mantener su base de datos actualizada, y no es necesario integrarla al OA que se entregará al consumidor.

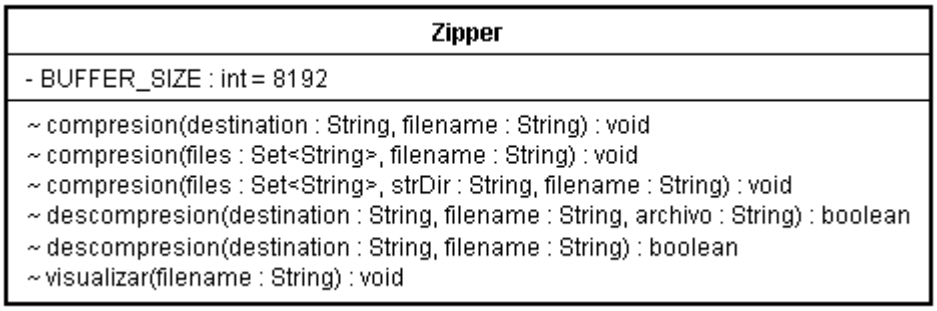

Figura 2.8 Clase Zipper

La clase *Zipper* de la figura **[2.8]** contiene funciones para manejar archivos en formato ZIP. Tiene funciones para compresión de un archivo específico dentro de un ZIP predeterminado como la descompresión de un archivo ZIP completo en una carpeta especificada.

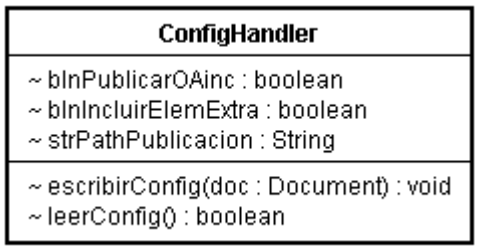

Figura 2.9 Clase ConfigHandler

La clase ConfigHandler (figura **[2.9]**) es una pieza importante del funcionamiento del agente y todas sus clases dependientes puesto que lee la configuración definida por el usuario y rige el comportamiento, las carpetas de búsqueda y creación de OA.

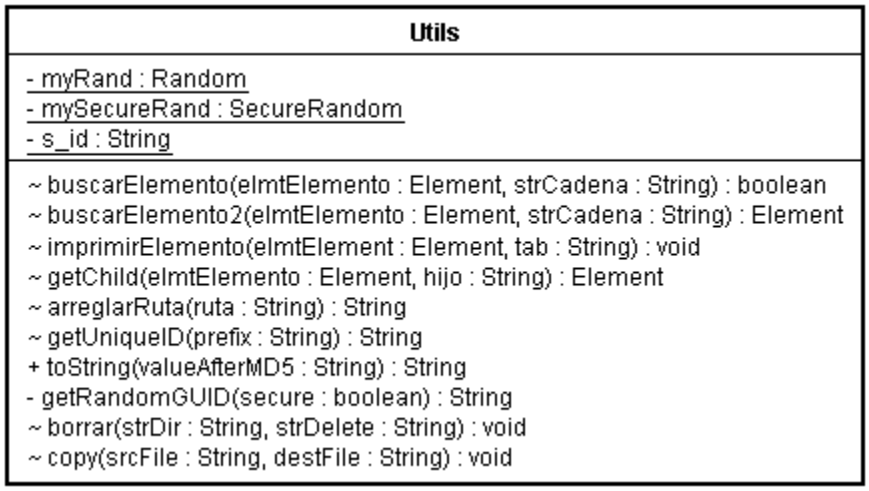

Figura 2.10 Clase Utils

La clase Utils mostrada en la figura **[2.10]** contiene diversas funciones que facilitan el uso de estructuras XML y búsquedas de elementos. Además de funciones para la creación de identificadores únicos, borrado de carpetas y copia de archivos.

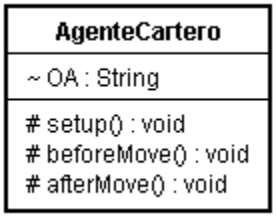

Figura 2.11 Clase AgenteCartero

La clase para AgenteCartero mostrada en la figura **[2.11]** muestra los métodos del agente Cartero. El método setup Configura el agente, el agente lee un OA antes de moverse y lo almacena como un arreglo de bytes. Al llegar a su destino lo escribe.

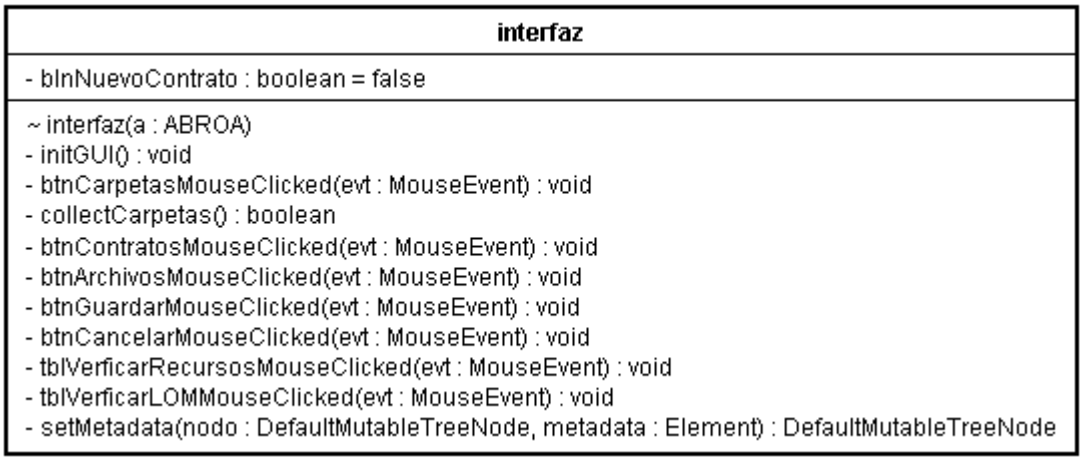

Figura 2.11 Clase interfaz

La Figura **[2.11]** muestra los métodos de la clase interfaz la cual proporciona una interfaz gráfica de usuario GUI para administrar los OA encontrados, comprimirlos, y publicarlos, además de un apartado con vínculos a páginas de repositorios de objetos de aprendizaje y un editor gráfico del archivo de configuración.

II.5 Diagramas de secuencia

Publicación de OA

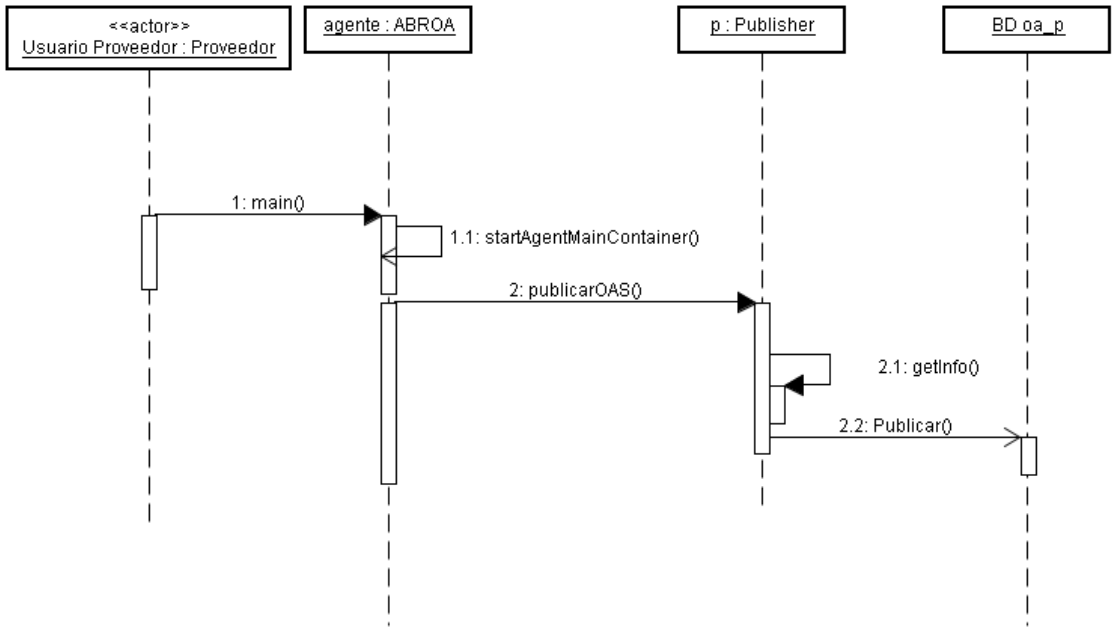

Figura 2.13 Diagrama de secuencia Publicación de OA

Descripción

1. El usuario proveedor inicia el método *main()*
   1.1. Se inicia la plataforma principal con el método *startAgentMainContainer()* y se crea un objeto *Agent* de tipo ABROA
2. Se llama al método *publicarOAS()* de la clase Publisher.
   2.1. Se obtiene la información de un OA, en caso de no existir, el método *getInfo()* la crea.
3. Se publica el OA en la Base de Datos del proveedor con la información obtenida.

Entrega de OA

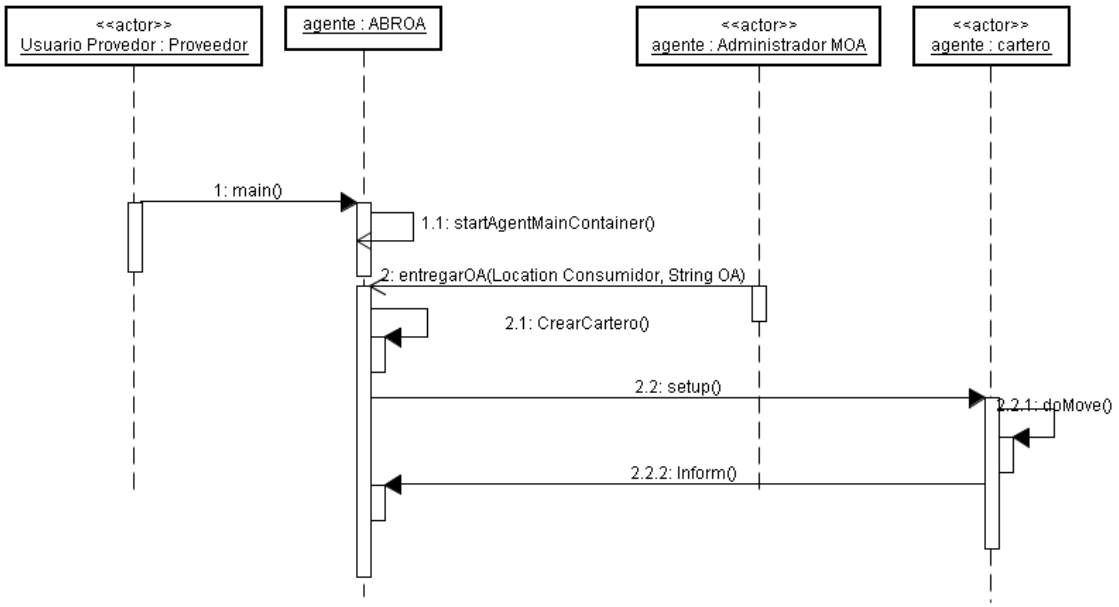

Figura 2.14 Diagrama de secuencia Entrega de OA

Descripción

1. El usuario inicia el método *main()*.
   1.1. Se inicia la plataforma principal con el método *startAgentMainContainer()* y se crea un objeto *Agent* de tipo ABROA
2. El agente administrador envía un mensaje que contiene la ubicación del consumidor y el OA solicitado.
   2.1. Se crea un objeto *Agent* de tipo cartero.
   2.2. Se configura el agente cartero con el nombre del OA solicitado.
      2.2.1. El agente cartero lee el OA antes de moverse. Se mueve a la ubicación del cliente y después escribe el OA en el destino.
      2.2.2. El agente cartero manda un mensaje para informar que la escritura del OA ha sido exitosa.

Búsqueda de Posibles OA

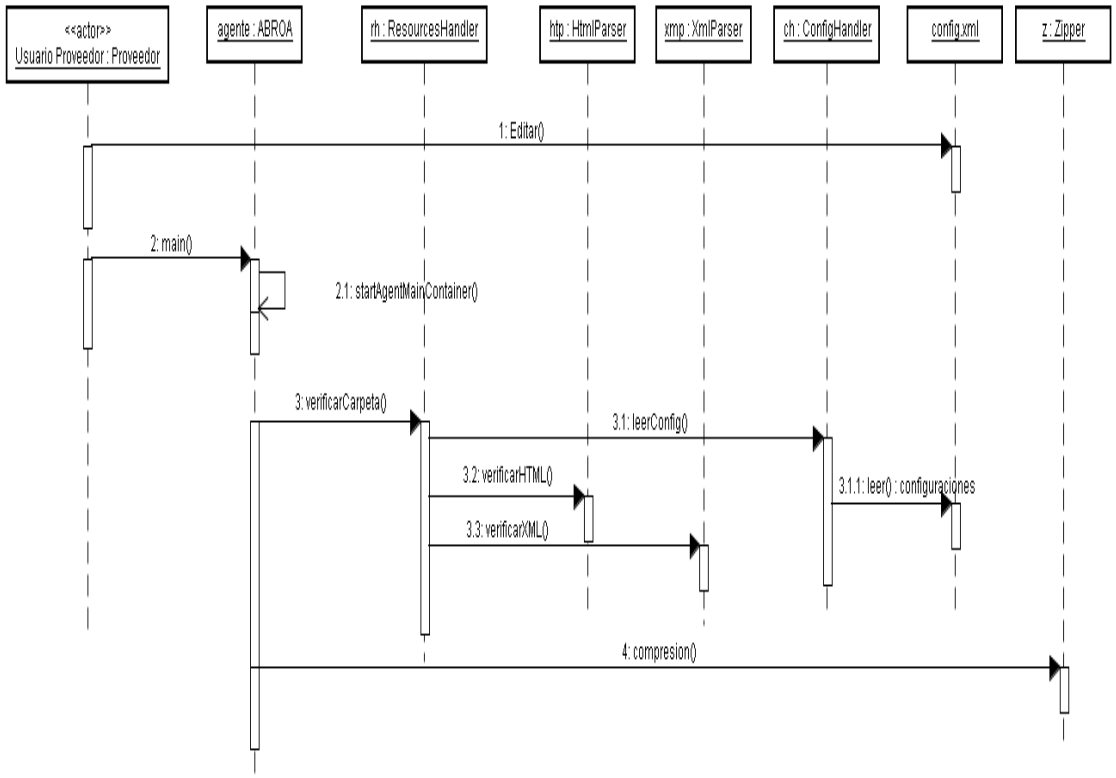

Figura 2.15 Diagrama de Secuencia Búsqueda de Posibles OA

Descripción

1. El usuario proveedor edita el archivo de configuración para determinar los parámetros y preferencias de búsqueda y creación de OA.
2. El usuario inicia el método *main()*.
   2.1. Se inicia la plataforma principal con el método *startAgentMainContainer()* y se crea un objeto *Agent* de tipo ABROA
3. Se llama al método *verificarCarpeta()* de la clase *ResourcesHandler()*.
   3.1. Se llama al método *leerConfig()* de la clase *ConfigHandler()* la cual obtendrá la información del archivo *config.xml*
   3.2. Se llama al método *verificarHTML()* para obtener los recursos pertenecientes a los OA que tengan como archivo inicial un archivo HTML.

3.3. Se llama al método verificarXML() para obtener los recursos pertenecientes a los OA que tengan como archivo inicial un archivo XML.

4. Se comprime el OA encontrado con la lista de recursos obtenida llamando al método *compresion()* de la clase *Zipper*.

Capítulo III Implementación

Plataforma JADE

Para la implementación se ha utilizado JADE (Java Agent DEvelopment Framework) que es una plataforma para desarrollo de Sistemas MultiAgentes (SMA). Emplea JAVA como lenguaje de programación y el tipo de comunicación que provee está basada en el estándar FIPA y emplea ACL como lenguaje de comunicación. Tiene capacidades de migración débil y de empleo de ontologías. Incluye:

- Un entorno de ejecución donde los agentes JADE se ejecutan y el cual debe estar activo en un equipo determinado antes de que uno o más agentes puedan ser ejecutados en dicho equipo.
- Una biblioteca de clases para poder programar agentes, comportamientos, ontologías y diversas herramientas para mejorar y medir el desempeño de los agentes.
- Un conjunto de herramientas gráficas que permite la administración y monitorización de la actividad de agentes en ejecución.

Ya que es usado por todo el MOA, por sus características y documentación existente se empleó en el desarrollo del agente ABROA y del agente móvil Cartero. A continuación daremos una breve introducción a las funcionalidades y procedimientos de la plataforma JADE antes de proseguir con la implementación en específico de nuestro sistema.

Creación de un agente en JADE

La clase *Agent* incluida en el paquete jade.core, define las características básicas para que el agente interactúe con la plataforma y los métodos para implementar el comportamiento del agente tales como: envío y recepción de mensajes, uso de protocolos de interacción estándar y modelo computacional multitarea.

Desde el punto de vista del programador, un agente JADE es simplemente una instancia de una clase de usuario que hereda de la clase base *Agent*.

A continuación se muestra el código Jade para la creación de un agente.

```
import jade.core.Agent;
public class Agent1 extends Agent {
protected void setup() {
```

```
System.out.println("Hola! El agente " +getAID().getName()+" está
activo.");
}
}
```

En el código anterior, se declara la clase jade.core.Agent para poder utilizarla como base para la clase Agent. En java, una relación de herencia entre clases se declara con la palabra reservada **extends**.

El método que se implementa en el ejemplo es el método setup(), el cual se utiliza para inicializar al agente. Las tareas que un agente tiene que realizar, son usualmente llevadas a cabo dentro de **"comportamientos".**

Para explicar los comportamientos, es necesario conocer el ciclo de vida de los agentes JADE, el cual se describe a continuación.

Ciclo de vida de los agentes JADE

FIPA especifica un ciclo de vida para agentes, el cual es seguido por los agentes JADE, este ciclo es mostrado en la siguiente figura.

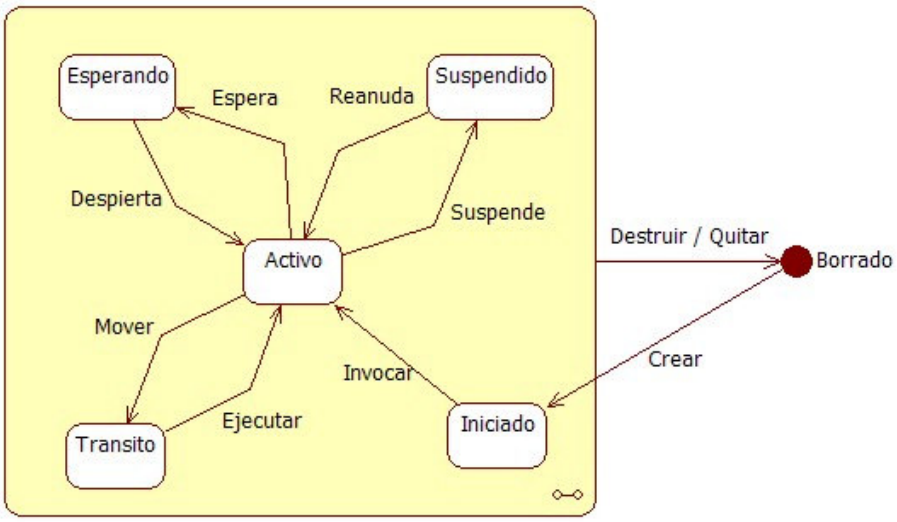

Figura 3.1 Ciclo de vida de un agente definido por FIPA

Como se muestra en la figura **[3.1]**, un agente JADE puede estar en uno de varios estados, los cuales se describen a continuación.

- INICIADO: El objeto Agente es construido, pero aún no tiene nombre o dirección y no puede comunicarse con otros agentes.

- ACTIVO: El objeto Agente está registrado en la plataforma, tiene un nombre y dirección y tiene acceso a todas las características de JADE.
- SUSPENDIDO: El objeto Agente está detenido. Su hilo de ejecución está pausado y ninguno de sus comportamientos está en ejecución.
- ESPERANDO: El objeto Agente esta bloqueado, esperando por algo. Su hilo interno esta durmiendo en un monitor de Java y despertará cuando se cumpla determinada condición (usualmente cuando llega un mensaje).
- BORRADO: El Agente está muerto definitivamente. El hilo interno ha concluido su ejecución y el agente ya no está registrado en la plataforma.
- TRÁNSITO: Un agente móvil entra en este estado mientras está migrando a la nueva ubicación. El sistema continúa almacenando mensajes que serán enviados a la nueva ubicación.

La clase *Agent* brinda métodos para realizar las transiciones entre los diversos estados. Por ejemplo, el método *doWait()* pone al agente en el estado ESPERANDO desde un estado ACTIVO, *doSuspend()* pone al agente en el estado SUSPENDIDO desde los estados ACTIVO o ESPERANDO. Otros métodos son doActivate(), doDelete(), doMove(), doWake().

Ahora bien, las tareas del agente se realizan mediante comportamientos, y un agente solamente puede ejecutarlos cuando está en estado ACTIVO.

Comportamientos de agentes JADE

Un agente debe ser capaz de llevar a cabo varias tareas concurrentes en respuesta a diferentes eventos externos. Todas las tareas de un agente son modeladas como objetos de la clase *Behaviour*.

El programador que desee implementar una tarea específica debe definir una o más subclases *Behaviour*, instanciarlas y agregarlas a la lista de tareas del agente. La clase *Agente* posee dos métodos: *addBehaviour(Behaviour)* y *removeBehaviour(Behaviour).*

Los comportamientos de un agente son programados por un planificador, el que lleva a cabo una planificación de tipo round-robin entre todos los comportamientos del agente. Este planificador ejecuta el método *action()* de cada comportamiento, y cuando este método

regresa, el método *done()* es llamado para verificar si el comportamiento ha terminado su tarea. De ser así, el comportamiento es eliminado de la cola de espera.

JADE ofrece una serie de comportamientos predefinidos, los cuales pueden contener subcomportamientos y ejecutarlos de acuerdo a cierta política. Por ejemplo, existe una clase *SequentialBehaviour*, la cual ejecuta sus subcomportamientos uno tras otro en cada invocación del método *action()*.

Algunos de estos comportamientos son mostrados en el siguiente diagrama de UML, junto con su explicación correspondiente.

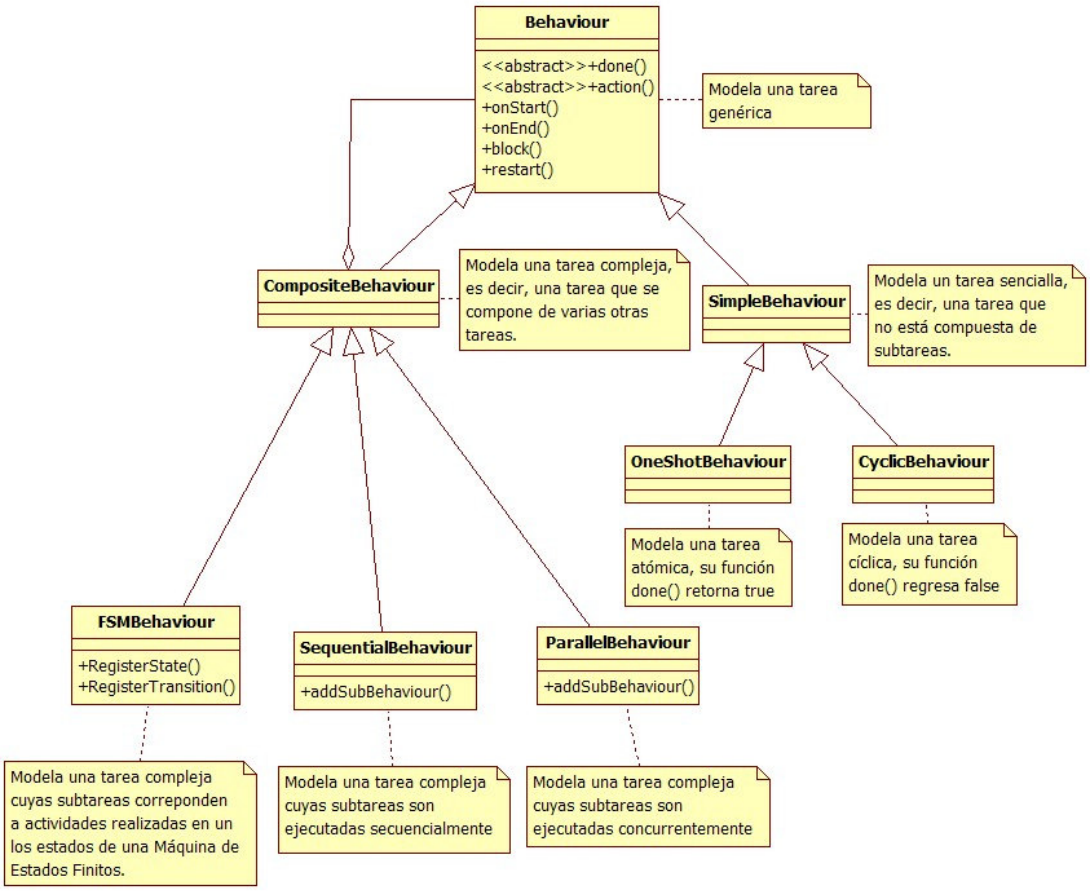

Figura 3.2 Modelo UML de la jerarquía de clases Behaviour

Servicio de movilidad intra-plataforma

JADE provee de un servicio en la plataforma llamado *Agent Mobility Service* que implementa la movilidad intra-plataforma. Esto provee a los agentes de software de la habilidad para moverse dentro de diferentes contenedores dentro de la misma plataforma. Sin embargo este mecanismo no permite mover a los agentes a contenedores de otras plataformas.

Métodos de movilidad.

En JADE la movilidad es controlada simplemente por el método *doMove()* en la clase *Agent*:

```
void doMove(Location destino)
```

El parámetro destino debe de ser un objeto de una clase implementando la interfaz *Location.* En la plataforma JADE hay dos clases que implementan esta interfaz, ambas dentro del paquete *jade.core*. La primera es *ContainerID*, la cual es usada para especificar que el destino de el agente serpa un contenedlo de la plataforma donde se encuentra ejecutándose actualmente. La Segunda es *PlataformID*, la cual es usada para indicar que el destino del agente es el contenedor principal de una plataforma diferente. Cuando una plataforma remota es indicada como destino del agente móvil, se deben de utilizar mecanismos de movilidad inter-plataforma.

Una vez invocado este método se inicia el proceso de migración del agente al contenedor destino especificado. La mayoría del código utilizado para hacer esto esta localizado en el paquete *jade.core.mobility*. La invocación de de *doMove()* es reenviada a el *Agent Mobility Service* a través del método *move()*. Lo primero que hace es cambiar el estado del agente de ACTIVE a TRANSIT causando que el agente interrumpa sus actividades actuales y sea suspendido mientras la plataforma lo reubica. Los usuarios pueden especificar las operaciones a ser realizadas cuando el proceso de movilidad sea iniciado, por ejemplo, para guardar el estado del agente. Estas operaciones son especificadas sobrecargado el método del la clase *Agent*:

```
void beforeMove()
```

Una contraparte del método llamada *afterMove()* también puede ser especificada para activar las operaciones que deben ser ejecutadas después de que el agente se mueve, antes de que recupere el estado ACTIVATE en el lugar destino.

Serialización de Agentes.

Después de la invocación *beforeMove()*, se inicia un comando vertical solicitando que el agente sea movido al destino contenido dentro del comando. Si el comando contiene un objeto destino con una instancia de la clase *PlataformID*, se iniciará el Inter-Platform Mobility Service que iniciará la migración del agente a la plataforma remota especificada. Cuando el objeto destino es una instancia de la clase *ContainerID*, el Agent Mobility Service dentro de la misma plataforma. El *ContainerID* o cualquier contenedor pueden ser solicitados al agente AMS de la plataforma en vez de construirlo a partir de la clase *ContainerID*.

La migración de un agente debe de incluir la transmisión de al menos su código y sus datos, y posiblemente también su estado. En JADE, los agentes modelan su estado de ejecución como datos internos del agente, entonces tiene sentido que solamente se necesite transferir el código y los datos. Los datos de un agente son contenidos en un objeto de java representando al agente, así que transmitir este objeto junto con su código será suficiente para restaurar el agente en su destino.

La serialización de java es usada para transmitir una instancia de agentes a través de una red por medio de la codificación recursiva de los valores de los miembros internos de un objeto agente en un flujo de bytes. El usuario debe especificar cuales miembros de datos serán trasmitidos juntos con el agente usando el modificador *transient*. Por ejemplo. Si un agente tiene un miembro *FileInputStream* (el cual no es serializable) y el cual no es declarado como transient, el proceso de movilidad fallará y enviará una excepción *NotSerializableException*.

Programación de un agente móvil.

Una meta esencial de los servicios de movilidad de JADE es que sea tan fácil de usar como sea posible. Por esto es necesario sobrecargar algunos métodos y llamar a los métodos *doMove()* o *doClone()* de la clase *Agent* para mover o clonar al agente (nótese que *doClone()* solo esta disponible para el *Intra-Platform Mobility Service*).

Para indicar el destino de un agente móvil o un agente clonado, JADE define la interfaz *jade.core.Location*. Hay dos implementaciones de esta interfaz: *jade.core.ContainerID* y *jade.core.PlatformID* para migrar en la misma plataforma y entre plataformas

respectivamente. El *ContainerID* debe de ser inicializado con el nombre del contenedor destino y su dirección de transporte. El *PlataformID* debe de ser inicializado con el AID del AMS de la plataforma remota incluyendo su dirección de transporte.

Para mover un agente a otro contenedor o plataforma, se debe de incluir una llamada al método *doMove()* dentro del comportamiento de un agente. Este método cambia el estado del agente a TRANSIT indicando que esta a punto de migrar. El siguiente código ilustra la migración intra-plataforma.

```
// Crear algunas variables
String containerName = "Container-1";
ContainerID destination = new ContainerID();
// Inicializar el objeto destino
destination.setName(containerName);
// Cambiar el estado del agente a mover
myAgent.doMove(destination);
```

En caso de la migración entre plataformas, un ejemplo similar seria el siguiente:

```
// Construir el AID del AMS de la plataforma remota
AID remoteAMS = new AID("ams@remotePlatform:1099/JADE", AID.ISGUID);
// Especificar el MTP colocando la dirección del AMS remoto
AMS.addAddresses("http://remotePlatformaddr:7778/acc");
// Se crea el objeto Location
PlatformID destination = new PlatformID(remoteAMS);
// Se cambia el estado del agente a mover
myAgent.doMove(destination);
```

Como estos servicios de movilidad implementan migración débil, la estructura del código debe estar basada en una maquina de estados finitos. Esto es porque el contador del programa del agente en ejecución no se trasmite, haciendo imposible continuar la ejecución en la línea siguiente después del llamado a *doMove()* o *doClone()*. En vez de eso, la ejecución de un agente solo puede continuar desde el inicio del código del agente. Usando una representación de una maquina de estados finitos, puede ser creada una estructura que segmente el código del agente en secciones con variables asignadas indicando el estado de ejecución del código.

Si se usa una sentencia switch, por ejemplo, un agente con los roles de vendedor y comprador puede tener código con dos estados, uno para vender y otro para comprar. El agente debe de asignar una variable de estado antes de dejar el contenedor o la plataforma para indicar cual rol será iniciado en la próxima ubicación. Para crear una representación de la máquina finita de estados, en una plataforma con dos contenedores en un arreglo de

ubicaciones, el código del comportamiento a ser ejecutado es separado como se muestra a continuación:

```
addBehaviour(new CyclicBehaviour(this){
public void action() {
switch(_state)
{
case 0:
// El agente empieza a migrar
      _state++;
      myAgent.doMove(_dests[0]);
      break;
      case 1:
      // el agente migra al segundo contenedor
      _state++;
      myAgent.doMove(_dests[1]);
      break;
      case 2:
      // el agente muere
      myAgent.doDelete();
      break;
      default:
      myAgent.doDelete();
}
}
private ContainerID[] _dests = ...;
private int _state = 0;
});
```

Este ejemplo muestra como debe de ser estructurado el código del agente en JADE para preservar su estado usando una variable.

Durante los procesos la serialización y de-serialización del código del agente migrando, algunos recursos usados por el agente también serán transferidos, mientras otros, serán desconectados antes de la migración del agente y reconectados en el destino. JADE provee dos métodos de en la clase *Agent* para el manejo de recursos y solamente necesitan ser sobrecargados por el programador.

- **beforeMove()** Es llamado en la ubicación origen, cuando la operación de migración se ha realizado exitosamente, antes de que el agente sea activado en el contenedor destino y la instancia original del agente esta a punto de ser eliminada. Este método es el lugar ideal para liberar cualquier recurso local usado por la instancia original del agente (por ejemplo cerrar archivos abiertos, y GUIs). Si los recursos fueron liberados antes de saber si la migración fue exitosa, tendrían que ser abiertos de nueva cuenta. De la misma manera, cualquier información que sea transportada por

el agente a la nueva ubicación debe de ser establecida antes de que se llame al método *doMove()*. Cualquier operación realizada en *beforeMove()* no tendrá impacto alguno en la instancia migrada.

- **afterMove()** Es llamado en la ubicación destino tan pronto como el agente llega y es asignada su identidad, pero antes de que los comportamientos del agente sean reiniciados.

Para el clonado de agentes, JADE provee de un par de métodos correspondientes, *beforeClone() y afteClone()*, estos métodos son similares a los anteriores. Estos cuatro métodos son miembros *protected* de la clase *Agent*, definidos como clases vacías. Los agentes definidos por el usuario pueden sobrecargar estos métodos según se necesite.

Agentes JADE en el Sistema de Búsqueda y Recuperación de Objetos de Aprendizaje

Se crearon 2 Agentes para este sistema. El primero es un agente de búsqueda de posibles OA en el servidor del proveedor. Para detectar posibles OA debe de analizar archivos en la estructura de directorios local. Estos archivos deben coincidir con un nombre especificado por el usuario en el archivo de configuración. El agente intentará crear OA a partir de estos archivos analizando sus recursos y empaquetándolos. También publicará los OA que tenga disponible en la base de datos del proveedor. Se ha llamado a este agente ABROA por ser un Agente de Búsqueda y Recuperación de OA. Tiene tres comportamientos cíclicos: Uno que localizará todos los posibles OA y creará archivos manifiestos de ser necesario, después los empaquetará. Otro que publicará todos los OA disponibles en la base de datos. Y otro que estará esperando mensajes del administrador con peticiones de OA, acto seguido creará un agente cartero con esa información. El Agente ABROA también cuenta con una GUI en caso de que el usuario desee tener más interacción con las acciones del agente.

Por otra parte el Agente cartero será creado por el agente ABROA cuando le sea solicitado un OA para ser entregado a un consumidor. El agente Cartero es inicializado entonces con una dirección destino y la dirección del OA solicitado. Leerá y empaquetará el OA, se moverá al consumidor y entonces lo desempaquetará y escribirá.

Implementación de Analizadores

A continuación se describirán los analizadores de archivo que fueron necesarios implementar para la realización de este sistema. Se utilizaron las bibliotecas JDOM y htmlparser para facilitar el análisis de los archivos XML y HTML respectivamente, puesto que Java solamente ofrece funciones de lectura y escritura secuencial. A continuación se describirán ambos.

Analizador XML

El analizador XML fue implementado en la clase *XMLParser*. Su objetivo es reconocer los distintos elementos que componen al archivo manifiesto para poder obtener información de él. Esta clase utiliza la biblioteca JDOM para el análisis de XML.

El archivo manifiesto de un OA SCORM contiene: un encabezado, una sección <metadata> que es el LOM del OA y es opcional, una sección <resources> que muestra los recursos que componen al OA, y una sección <organizations> que muestra como será mostrado el OA al usuario y que también es opcional. Nuestro principal objetivo es verificar que los recursos existan antes de intentar empaquetarlos en un archivo ZIP. En caso de no existir un manifiesto, por ejemplo si se desea crear un OA a partir de un archivo HTML, esta clase también será la encargada de crearlo.

Para conformar el documento XML como una estructura arbórea en memoria, primero se importa la biblioteca.

```
import org.jdom.*;
```

Después se declara un objeto de tipo *Document* donde se almacena el documento y "se construye " el árbol a partir del archivo.

```
try
{
File archivo = new File(strRutaArchivo);
SAXBuilder builder = new SAXBuilder(false);
      Document doc = builder.build(archivo);
      Element raiz = doc.getRootElement();
} catch (JDOMException e)
{
      e.printStackTrace();
}
catch (IOException e)
```

```
{
      e.printStackTrace();
}
```

De esta manera se tiene el nodo raíz almacenado en el objeto tipo *Element* Raiz. Para realizar todas las operaciones necesarias.

La clase XmlParser cuenta con las siguientes funciones:

- crearManifest(Fila). Crea un archivo imsmanifest.xml de acuerdo al estándar SCORM 2004 a partir de los atributos pasados como argumentos. Además copia los archivos adicionales de esquema (XSD).
- getMetadatos(String, String, String). Obtiene el LOM para un posible OA, el primer argumento será la ruta del archivo de la cual se obtendrá esta información.
- getSubmanifests(Element, String). De acuerdo al estándar SCORM 2004 un archivo manifiesto puede usar submanifiestos para extender la información de sus recursos. Si bien no utilizaremos la información de estos submanifiestos, si deseamos incluirlos en la lista de recursos que se empaquetarán en el ZIP puesto que están declarados. Esta función devuelve una lista con estos submanifiestos en caso de existir.
- verificarManifest(String, File). Esta función es utilizada por la GUI y verifica la existencia del LOM y obtiene el esquema y la versión del esquema utilizado en el manifiesto.
- verificarXML(String, File). Esta función obtiene todos los recursos declarados en la sección resources del archivo manifiesto y si contiene referencias a archivos HTML también obtiene los recursos que estén declarados dentro.

Analizador HTML

El analizador HTML fue implementado en la clase HtmlParser. Su objetivo es reconocer recursos necesarios para la creación de un OA. Esta clase utiliza la biblioteca htmlparser para facilitar el análisis de archivos HTML, reconociendo etiquetas específicas y proporcionando un acceso estructurado a sus atributos.

A continuación se muestra como obtener todas las etiquetas **<A>** de un archivo HTML. Estas etiquetas se usan para crear un hipervínculo a otra ubicación.

```
import org.htmlparser.*;
import org.htmlparser.Parser;
```

```
…
      // inicializa el parser con la ruta
      Parser analizador = new Parser(strRuta);
      // crea la lista de nodos y el filtro de etiquetas a utilizar
      NodeList listaNodosA= new NodeList();
      NodeFilter filtroA = new TagNameFilter ("A");
      // recorre el archive html añadiendo las etiquetas encontradas a
      //  a la lista  de nodos
      for (NodeIterator e = analizador.elements(); e.hasMoreNodes();)
      {
            org.htmlparser.Node Nodo =e.nextNode ();
            Nodo.collectInto (listaNodosA, filtroA);
      }
      …
      //Búsqueda en tags <A>
      if(listaNodosA.size()!=0)
      {
      for(org.htmlparser.Node Nodo:listaNodosA.toNodeArray())
      {
            TagNode nodo= (TagNode)Nodo;
      for(Iterator ite=nodo.getAttributesEx().iterator();
      ite.hasNext();)
            {
                  org.htmlparser.Attribute;
                  attr=(org.htmlparser.Attribute)ite.next();
                  String value =attr.getValue();
                  if(attr!=null && value!=null&& attr.getName()!=null)
                  {
                        //Agrega los recursos que encuentre.
                        strHTMLs.addAll(verificarParamsHTML(value));
                        strIMGs.addAll(verificarParamsIMG(value));
                  }
            }
      }}
```

De esta manera se obtienen todos los posibles recursos en la etiqueta A, pero esto se tiene que realizar también para otras etiquetas como <IMG> para imágenes, <OBJECT> para archivos flash embebidos, <APPLET> para applets de java, <SCRIPT> para archivos con código ejecutable (javascript, visualbasicscript, etc), <LINK> para vínculos a otros archivos como puede ser una hoja de estilos en cascada (css).

La clase HtmlParser cuenta con las siguientes funciones:

- verificaHTMLs(File). Función recursiva que encuentra todos los recursos dentro de los archivos HTML referidos dentro del archivo HTML de inicio.
- verificarHTML(String, File). Función que devuelve toda la lista de posibles recursos referidos desde el archivo HTML de inicio.

- verificarParamsApplet(String). Hace un tratamiento a la cadena enviada como parámetro para obtener los recursos referidos en un atributo dentro de una etiqueta de tipo <APPLET>.
- verificarParamsHTML(String). Obtiene todos los recursos dentro de la cadena enviada como parámetro. Esta función se ocupa de los archivos HTML que son referenciados dentro de funciones script.
- verificarParamsIMG(String). Obtiene todos los recursos dentro de la cadena enviada como parámetro. Esta función se ocupa de los archivos de imágenes (JPG, GIF, BMP, PNG) que pueden estar referenciados dentro de funciones script.

Interfaz de usuario.

Se programó una Interfaz Gráfica de Usuario (GUI) para asistir al usuario y mostrar el funcionamiento de la creación del OA y la publicación. Contiene además un apartado para la edición del archivo de configuración *conf.xml*, una colección de ligas a algunos repositorios de OA y una más para mostrar la creación de un agente cartero para la entrega de un OA .

A continuación se mostrarán estas pantallas y una breve descripción de su funcionamiento.

Verificación de OA.

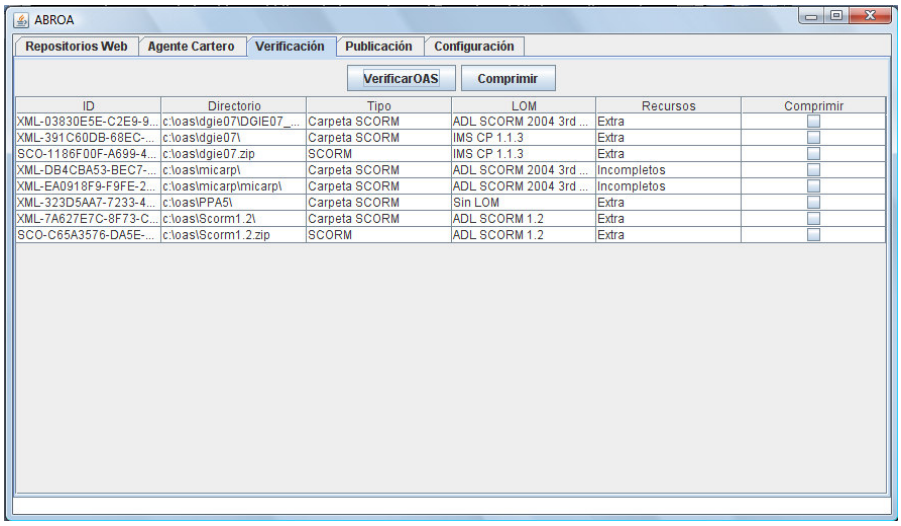

Figura 3.3 GUI de agente ABROA. Pestaña de verificación de OA.

En esta pestaña (figura **[3.3]**) se muestra la verificación de los OA existentes en la máquina del proveedor. Al hacer clic en el botón **verificar OA** se localizan los posibles OA existentes en las carpetas definidas en la configuración. Se crean los archivos manifiestos si es necesario. Al hacer doble-clic sobre la celda **LOM** en alguna fila se mostrarán los metadatos correspondientes a ese OA. Se muestran los recursos encontrados al hacer doble-clic en la celda de la columna **Recursos** en la fila del OA correspondiente. La columna **Recursos** muestra en la celda uno de tres letreros: completos por si están declarados exactamente todos los recursos contenidos dentro de la carpeta, incompletos por si hay

recursos declarados que no existan y extra por si existen recursos en la carpeta que no estén declarados dentro del archivo manifiesto.

Se deben de seleccionar las carpetas que se desean comprimir seleccionando la casilla de verificación correspondiente y haciendo clic en el botón **Comprimir**. Se comprimirán todas las carpetas seleccionadas y los archivos ZIP resultantes serán copiados a la carpeta de ejecución del agente.

Publicación de OA.

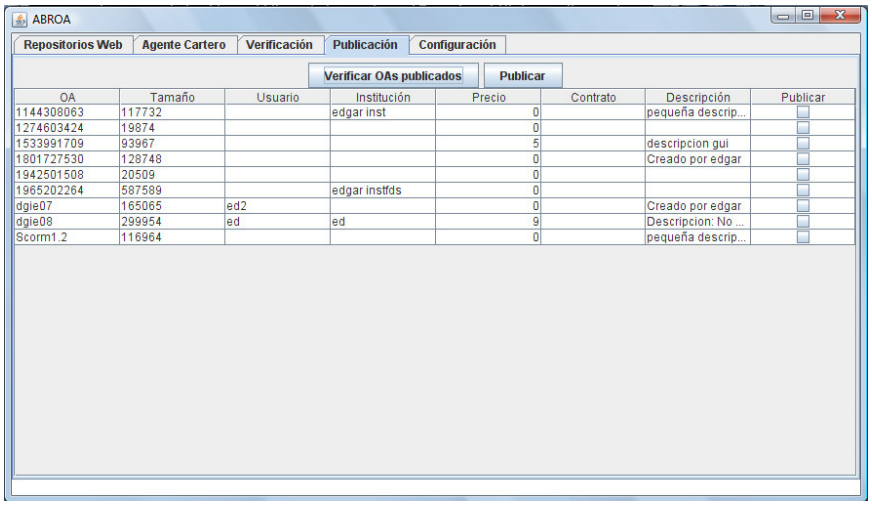

Figura 3.4 GUI de agente ABROA. Pestaña de publicación de OA.

En esta pestaña de publicación de OA mostrada en la figura **[3.4]** se demuestra como funciona la publicación de un OA en la base de datos del proveedor. Para mostrar todos los OA disponibles hacer clic en el botón **Verificar OA publicados**. La columna **Publicar** mostrará su estado, publicado o no publicado.

Se pueden editar las columnas: **Usuario**, el nombre del usuario que publica previamente registrado en la base de datos. **Institución**, la institución a la que pertenece el usuario. **Precio,** el precio del OA. **Descripción,** una descripción del OA inicialmente obtenida de los metadatos del OA.

Para publicar o remover de la base de datos de datos del proveedor un OA se debe seleccionar o deseleccionar de la casilla de verificación respectivamente en la columna **Publicar** y dar clic en el botón **Publicar.**

Configuración del Agente

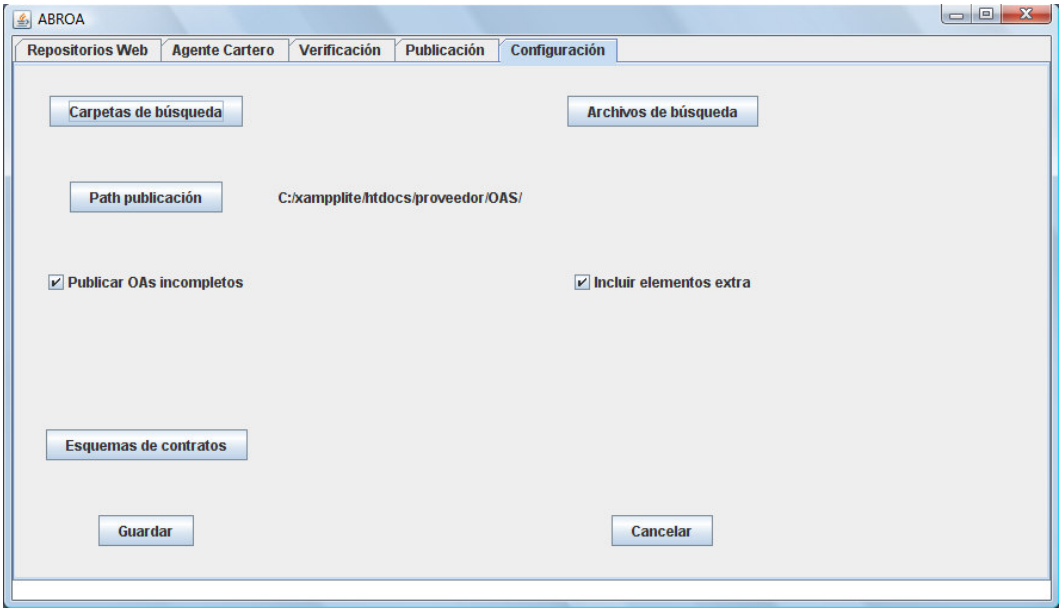

Figura 3.5 GUI de agente ABROA. Pestaña de configuración

En la pestaña de configuración mostrada en la figura **[3.5]** se permite al usuario cambiar el archivo de configuración *conf.xml*. Las opciones de las que dispone el usuario son: **Carpetas de búsqueda**, carpetas donde al agente está permitido buscar posibles OA incluyendo sus subcarpetas. **Archivos de búsqueda,** establece los nombres de los archivos iniciales de búsqueda. Se recomienda utilizar *imsmanifest.xml* e *index.html*. **Path de publicación** es la ruta donde se ubicarán los OA al ser publicados, esta Ruta depende de la configuración del servidor MySQL y apache.

La opción **publicar OA Incompletos** especificará si se empaquetarán OA que contengan referencias a archivos incompletos y la opción **Incluir elementos extra,** incluirá todos los archivos en la carpeta y subcarpetas aunque no estén declarados en el archivo manifiesto.

En la opción de **esquemas de contratos** se especificarán los esquemas o estructuras bases que serán utilizados para crear contratos de compra venta de OA.

Al dar clic en el botón **Guardar** se escribirán los cambios en el archivo *conf.xml*.

Repositorios Web

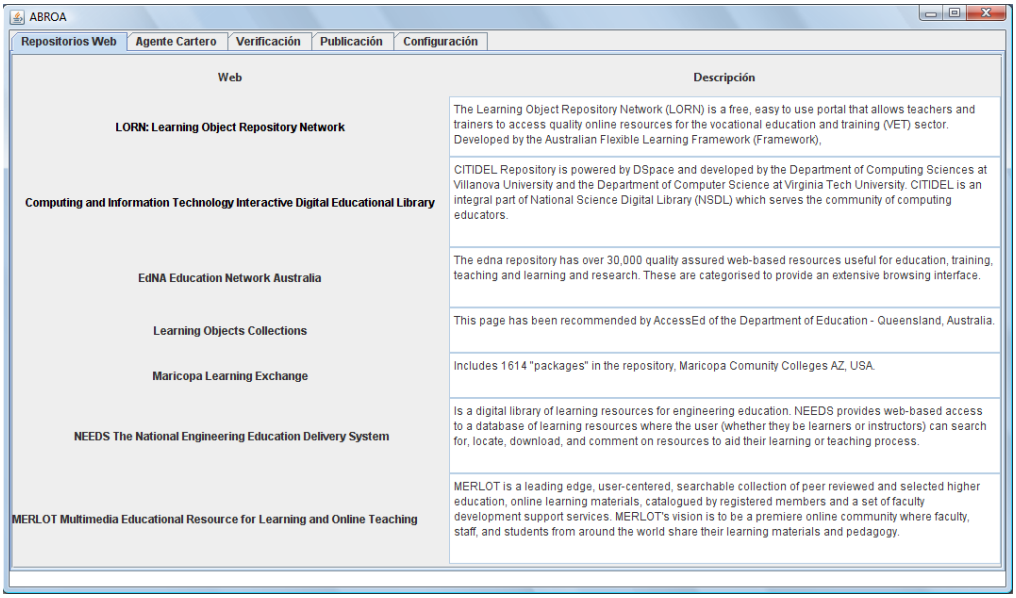

Figura 3.6 GUI de agente ABROA. Pestaña de repositorios Web.

Esta pestaña muestra una colección de ligas a los principales repositorios Web así como una breve descripción de su contenido. Estos repositorios almacenan miles de OA de diferentes tipos y son actualizados constantemente. El usuario entonces podrá accesar a alguno de estos repositorios, adquirir un OA y almacenarlo en su máquina, y de esta manera incluirlo en los OA a su disposición para ser publicados.

Agente Cartero

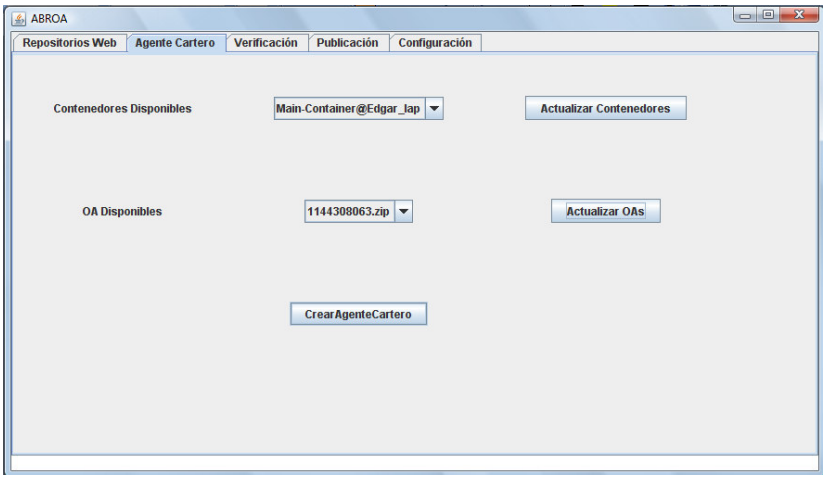

Figura 3.7 GUI de agente ABROA. Pestaña de agente Cartero.

En la pestaña de agente Cartero (figura **[3.7]**) será mostrado el mecanismo para la creación de un agente cartero. Al dar clic en el botón **Actualizar contenedores**, se mostrarán los contenedores disponibles el la caja de selección **Contenedores disponibles.** Se debe seleccionar un contenedor diferente al de donde reside el agente ABROA para que se pueda realizar la migración. Al dar clic en el botón **Actualizar OA** se mostrarán los OA disponibles para ser entregados en la caja de selección. Al dar clic en el botón **Crear Agente Cartero** se creará el agente cartero que leerá el OA indicado, lo empaquetará dentro de sí mismo, se moverá al contenedor indicado y desempaquetará y escribirá el OA.

CONCLUSIONES Y RECOMENDACIONES

Este sistema fue realizado como una actualización a las capacidades del Mercado de Objetos de Aprendizaje el cual es un proyecto llevado a cabo entre miembros del CENIDET, la Universidad Autónoma del Estado de Hidalgo y la Benemérita Universidad Autónoma de Puebla. El proyecto está financiado por el fideicomiso SEP-UNAM 159/2006. El trabajo realizado para la implementación del ABROA adquiere su propósito en la mejora continua del MOA. Se completaron satisfactoriamente los objetivos de esta tesis y se expandió el conocimiento en tres áreas principales: Agentes y agente móviles, Objetos de Aprendizaje y Analizadores de Texto en distintos formatos. Aunque concluye con esta tesis un proceso de investigación y desarrollo, queda mucho campo para mejorar las capacidades y características del ABROA y del MOA en general, de acuerdo a las necesidades de los usuarios finales.

Respecto al trabajo con Agentes considero que es un paradigma bastante interesante y amplio. Por su amplitud y riqueza, debemos delimitar muy bien nuestros objetivos en términos prácticos y enfocados al usuario, de lo contrario corremos el riesgo de desarrollar características innecesarias a favor de hacer más inteligentes o más autónomos a nuestros agentes. Una modificación al ABROA bastante útil sería la creación automática de OA basándose en documentos HTML almacenados en la Web. Aunque esto trae implicaciones de derechos de autor y de permisos de uso, las modificaciones a las herramientas para la creación de OA serán mínimas en el ABROA.

Si hablamos de los Objetos de Aprendizaje de igual manera nos referimos a un término bastante amplio. Inclusive hablando específicamente del estándar SCORM para OA, cada dependencia que lo usa crea su implementación del estándar y del LOM. Considerando esto el MOA tiene capacidades de expansión para manejar fuentes diversas de OA y solamente podremos intentar el intercambio de OA con instituciones de enseñanza de todo el mundo siempre y cuando concurramos en la especificación del estándar.

Los OA creados por el agente a partir de documentos HTML cumplen con el estándar SCORM 2004, pero deben de ser modificados por el diseñador de cursos para dar la organización de como serán presentados los recursos a los alumnos. Esta modificación es

realizada con editores externos al MOA y es un proceso manual y cansado. Es posible crear una propuesta de organización inicial integrando métodos de análisis del contenido del documento HTML a los métodos actuales del ABROA, pero esto sale del alcance de esta tesis.

El reforzar el MOA ampliando sus características y optimizando sus mecanismos constantemente proporcionará una herramienta eficiente y vigente para el intercambio de Objetos de Aprendizaje y de esta manera se logrará el impulso planeado al e-learning en nuestras instituciones.

Bibliografía

[1] Computer Security Division (2002), accesado el 15 de abril de 2008, Nacional Institute of Standars and Tecnology, http://csrc.nist.gov/mobileagents/projects.html.

[2] Newcomer, Eric; Lomow, Greg (2005). *Understanding SOA with Web Services:* Addison Wesley. ISBN 0-321-18086-0.

[3] Negroponte, N. (1997). *Agents: From Direct Manipulation to Delegation. In Software Agents.* J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

[4] Nwana, H. S. (1996). *Software Agents: An Overview*. *Knowledge Engineering Review,*11(3): 205-244.

[5] Foner, L. (1993). *What's an Agent, Anyway? A Sociological Case Study*, Agents Memo, 93-01, Media Lab, Massachusetts Institute of Technology.

[6] Shoham, Y. (1997). *An Overview of Agent-oriented Programming*. In *Software Agents,* ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

[7] Etzioni, O., y Weld, D. S. (1995). *Intelligent agents on the Internet: Fact, fiction, and forecast.* IEEE Expert, 10(4), 44-49.

[8] Franklin, S., y Graesser, A. (1996). *Is It an Agent or Just a Program? A Taxonomy for Autonomous Agents*. Proceedings of the Third International Workshop on Agent Theories,Architectures, and Languages. New York: Springer-Verlag.

[9] Moulin, B., y Chaib-draa, B. (1996). *An Overview of Distributed Artificial Intelligence*. Foundations of Distributed Artificial Intelligence*,* editores. G. M. P. O'Hare and N. R. Jennings, 3–55. New York: Wiley.

[10] Gilbert, D.; Aparicio, M.; Atkinson, B.; Brady, S.; Ciccarino, J.; Grosof, B.; O'Connor, et al. (1995). *IBM Intelligent Agent Strategy*, IBM Corporation.

[11] Petrie, C. J. (1996). *Agent-Based Engineering, the Web, and Intelligence*. IEEE Expert, 11(6): 24-29.

[12] Picco, G.P.(2000). *Understanding Code Mobility (Tutorial Session)*. In *ICSE '00:* Proceedings of the 22nd International Conference on Software Engineering, pp. 834, ACM Press, New York.

[13] Mir, J. (2004). *Protocolos criptográficos para canales de comunicación anónimos y protección de itinerarios en agentes móviles*. PhD thesis, Escola de postgrau.
[14] Tanenbaum, A.S. y Van Steen(2001). M. *Distributed Systems: Principles and Paradigms*. Prentice Hall PTR, Upper Saddle River, NJ.
[15] Brown, P. y Rossak, W. (2005). *Mobile Agents*. Morgan Kaufmann Publishers and dpunkt.verlag.
[16] White, J.E. (1996).*Telescript Technology: Mobile Agents*. Bradshaw Jeffrey: AAAI Press/MIT Press,
[17] Russell, S.J. and Norvig, P. (2003). *Artificial Intelligence: a Modern Approach*, 2da edición :Prentice Hall.
[18] Stephen Downes (30 de enero de 2008) *La realidad del Aprendizaje virtual* DNDLearn Conference, Cornwall Ontario.
[19] García Peñalvo, F. J. y García Carrasco, J. (2001). *Los espacios virtuales educativos en el ámbito de Internet: Un refuerzo a la formación tradicional*, Teoría de la Educación. Educación y Cultura en la Sociedad de la Información, accesado 15 de abril de 2008, http://www3.usal.es/~teoriaeducacion/rev_numero_03/n3_art_garcia-garcia.htm.
[20] R. Polsany (2003). Use *and abuse of reusable learning objects*, Journal of Digital Information.
[21] Wiley D (2000). *Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy*. The Instructional Use of Learning Objects: Online Version. Accesado en marzo de 2007, http://reusability.org/read/chapters/wiley.doc.
[22] Learning Technology Standards Committee (2002). *IEEE Standard for Learning Object Metadata*. IEEE Standard 1484.12.1, Institute of Electrical and Electronics Engineers, New York.
[23] FAQs (2005), *CETL Reusable Learning Objects*, accesado el 15 de abril de 2008, http://www.rlo-cetl.ac.uk/faqs.htm.
[24] Blog de Andrés Chiappe (2007) - Objetos de Aprendizaje, accesado el 15 de abril de 2008, http://andreschiappe.blogspot.com/2007/09/que-es-un-objeto-de-aprendizaje-what-is.html#links.

[25] Metodología para el Uso de Estándares Internacionales en la Creación de Objetos de Aprendizaje" Priego Meléndez Juan Leobardo, Tesis de maestria, BUAP, otoño 2006.