Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

Maestría en Ciencias de la Computación

Desarrollo del Agente Administrador del Mercado de Objetos de Aprendizaje

TESIS

Que para obtener el grado de

Maestría en Ciencias de la Computación

PRESENTA

Lluvia Erendira Ponce Martínez

ASESOR

Dra. Fabiola López y López

COASESOR

Dra. Darnes Vilariño Ayala

Noviembre 2007

Agradecimientos

A mis padres, sabiendo que jamás existirá una forma de agradecer en esta vida de lucha y superación constante, deseo expresarles que mis ideales, esfuerzos y logros han sido también suyos y constituye el legado más grande que pudiera recibir. Gracias, mil gracias, por ello los amo tanto, porque si ustedes no hubieran creído en mi, no habría llegado tan lejos, ustedes son mi fuente de fortaleza para seguir superándome día a día, sólo pienso en ustedes y con eso basta para seguir adelante, gracias papitos, esto es cien por ciento para ustedes.

A mi tía Ofelia, porque cuando me surgió la inquietud de buscar una escuela donde realizar mi maestría, no dudó en apoyarme y creer en mí para lograr este gran sueño; gracias tía, sin ti nunca hubiera dado este gran paso, nunca terminare de agradecértelo ¡te quiero mucho!.

A mi hermana, que aunque no ha estado junto a mi, se que cuento con su apoyo incondicional.

Gracias a mi asesora de tesis, la Dra. Fabiola López y López por su asesoramiento científico y estímulo para seguir creciendo intelectualmente.

Gracias a mi coasesora la Dra. Darnes Vilariño Ayala, por su disposición permanente e incondicional, por aclarar mis dudas, por sus sustanciales sugerencias durante la elaboración de esta tesis y sobre todo por su amistad.

A mis amigos, con quienes construimos conocimiento, compartimos mañanas, tardes y noches de estudio y trabajo.

Gracias al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo brindado.

Contenido

Índice de Tablas

Índice de Figuras

Capítulo 1

Introducción

A través del paso de los años, los materiales didácticos y la impartición de cursos de aprendizaje han sufrido considerables transformaciones. Heinz von Foerster definió al aprendizaje como aprender a aprender, concepto que marca la tendencia más importante quizás de la educación actual [1].

El paradigma constructivista del aprendizaje indica que el estudiante debe construir conocimiento por si mismo, y con la ayuda de otro (mediador) y que sólo podrá aprender elementos que estén conectados a conocimientos, experiencias o conceptualizaciones previamente adquiridos por él. El profesor pregunta, guía, conduce e interactúa, no enseña. Se adopta una estructura donde el estudiante aprende y el maestro facilita el aprendizaje, es decir, el profesor no es responsable del proceso de "asimilación instantánea", se parte de un salón de clases, en donde el elemento primordial es el alumno. El paradigma constructivista, considera a los alumnos como sistemas dinámicos que interactúan con otros sistemas dinámicos; lo cual es una característica central del proceso de enseñanza-aprendizaje [1].

Hoy en día se cuenta con el escenario de Educación a Distancia, lo que constituye una nueva forma de educación, donde surge la separación del estudiante y del profesor durante la mayor parte del proceso educativo. El tipo de material que se proporciona al alumno debe ser adaptado a las circunstancias de éste. Lo pedagógico es de suma importancia con respecto a lo tecnológico, es decir, la tecnología es un medio de comunicación que está sujeto al proceso de enseñanza-aprendizaje. Una vez comentado lo anterior, para poder

llevar a cabo la enseñanza a distancia, se necesita contar con contenidos que se ajusten a las necesidades y requerimientos de los alumnos.
La Combinación de educación y tecnología para llegar a las personas a través de grandes distancias es el sello del aprendizaje a distancia. Los cursos de aprendizaje con que se cuentan en la actualidad son elaborados como un todo, estos contienen grandes cantidades de información, que generalmente no es reutilizable, es decir, si un diseñador construye un curso de aprendizaje, este no podrá ser compartido ni reutilizado por otro diseñador que de igual manera, esté construyendo un curso (s) enfocado hacia el mismo tema.

Lo anterior nos conlleva a la producción de cursos repetitivos que se hubiesen podido compartir a través de la red y así eliminar esfuerzos en la creación de los mismos. Como solución a este problema surge el paradigma de Objetos de Aprendizaje (OA). Los OA son objetos digitales que pueden ser accedidos o entregados por Internet, estos OA poseen características de interoperabilidad, reusabilidad, durabilidad y accesibilidad.

Otra problemática que se presenta al plantear esta solución es como los OA van a ser promocionados, localizados, manipulados y controlados, al ser diseñados por proveedores y usados por consumidores deseosos de elaborar nuevos cursos. Esto puede ser resuelto adoptando la Arquitectura Orientada a Servicios (AOS), donde existen entidades propietarias de servicios (proveedores) y entidades deseosas de hacer uso eficiente de dichos servicios (consumidores)**,** y una entidad controladora de los servicios ofertados (administrador); de esta manera se puede manejar el concepto de Mercado.
Para el desarrollo de este mercado se propone usar el paradigma de agentes, en particular el administrador de este mercado será representado por un agente Administrador, que en dependencia del servicio que le sea solicitado, llevará a cabo la creación de un agente que estará especializado en proveer dicho servicio. Usando alguno(s) de estos agentes se permitirán publicitar OA elaborados por proveedores a través de su representante (Agente Proveedor) y se permitirá establecer el contacto con usuarios consumidores por medio de sus representantes (Agentes Consumidores) para adquirir dichos OA. Así pues, el mercado de objetos de aprendizaje se verá como un sistema multi-agente.

Actualmente la Benemérita Universidad Autónoma de Puebla está colaborando con la Universidad Autónoma del Estado de Hidalgo y el Centro Nacional de Investigación y Desarrollo Tecnológico para el desarrollo de un sistema de compartición y uso de objetos de aprendizaje, que puedan ser usados por los profesores que deseen construir sus cursos en línea. Dichas instituciones educativas actuarán como proveedores de objetos de aprendizaje, para ponerlos a disposición de los diseñadores de cursos.

El presente trabajo se estructura de la siguiente manera: En el capítulo 1 se brinda la introducción y los objetivos generales y específicos. En el capítulo 2 se describe el paradigma de objetos de aprendizaje, el paradigma de agentes, los sistemas multi-agentes, la arquitectura orienta a servicios y por último se da la descripción de una base de datos así como el modelo entidad-relación. En el capítulo 3 se presenta el modelo de análisis del mercado de objetos de aprendizaje, se realiza una introducción al planteamiento del problema y las soluciones que se le dan a éste; se muestra la arquitectura general del mercado de objetos, así como los diversos servicios plasmados en sus respectivos casos de uso y los diagramas de clases general usando la notación UML. En el capítulo 4 se presenta el modelo de diseño, donde se muestran a detalle los diagramas de clases y de secuencia, que nos permiten modelar el sistema de objetos de aprendizaje. En el capítulo 5 se presenta la implementación del mercado y por último se tienen las conclusiones y recomendaciones.

1.1 Objetivo General y Específico

1.1.1 Objetivo General

Desarrollar el Agente Administrador del Mercado de Objetos de Aprendizaje (MOA), y a su subconjunto de agentes, en quienes delegará tareas de servicios específicos solicitados por agentes externos al MOA (consumidores y proveedores). El agente administrador aplicará políticas de ingreso y permanencia en el MOA, y será el responsable de proveer el servicio de recepción y entrega de OA a los agentes involucrados en una transacción de compra-venta.

1.1.2 Objetivos Específicos

Para lograr lo descrito en la sección 1.1.1 se seguirá la metodología de Proceso Unificado de Desarrollo de Software (PUDS), teniendo como objetivos específicos:

- Investigar los beneficios que ofrece la plataforma JADE para el desarrollo de sistemas multi-agentes.
- Analizar la Arquitectura Orientada a Servicios (AOS).
- Modelar, diseñar e implementar un sistema de software que permita la publicación, localización y entrega de diversos objetos de aprendizaje a un par de entidades a quienes denominaremos, para nuestros efectos agentes consumidores y agentes proveedores de OA.
- Realizar el modelado, diseño e implementación utilizando UML (Lenguaje de Modelado Unificado).

Capítulo 2

Estado del Arte

2.1 Paradigma de Objetos de Aprendizaje

Los OA han permitido plantear una nueva forma de pensar la estructura del aprendizaje, es decir, la forma de pensar en el diseño que permitirá dar flexibilidad en el desarrollo de contenidos, disminución de costos y flexibilidad para contar con un material actualizado.

Los OA los podemos definir como una colección de materiales digitales llamados recursos (fotografías, documentos, animaciones, simulaciones, etc.) integrada con un objetivo formativo, de resultado medible y creados para dar soporte a un proceso de aprendizaje.

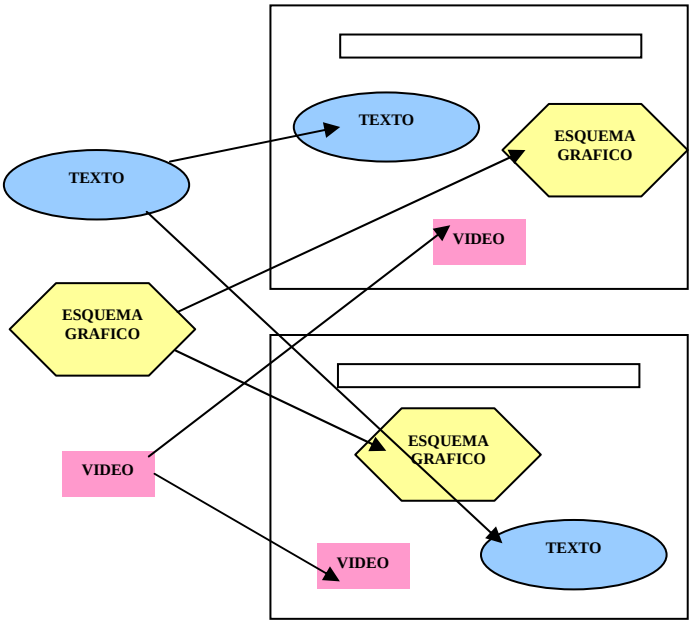

Figura 2.1 Esquema de Objetos de Aprendizaje

Los OA pueden verse como unidades de construcción, que pueden agruparse de diversas maneras para producir diferentes cursos que satisfagan diferentes objetivos de aprendizaje, como se ilustra en la figura 2.1.

Características de los Objetos de Aprendizaje:

Reutilizables: Pueden ensamblarse de diversas formas para formar diferentes cursos.
Interoperables: Interoperan entre ellos independientemente de su creador.
Duraderos: Soportan la evolución de las tecnologías de presentación y difusión.
Accesibles: Están disponibles en cualquier momento, lugar y red.

Podemos decir entonces, que el diseño e implementación de cursos en línea es una de las fases más complejas en la creación de un programa académico en línea. Como una solución a este problema surge el Paradigma de Objetos de Aprendizaje.

2.2 Paradigma de Agentes

El paradigma basado en agentes ha invadido la tecnología de cómputo durante los últimos 10 años, ha pasado de ser un área de interés dentro de la Inteligencia Artificial, a ser una tecnología de cómputo genérica en donde diferentes ciencias han aportado sus teorías para dar origen a la tecnología de agentes.

La tecnología de agentes está siendo usada para abordar la solución de múltiples problemas que se están presentando en la actualidad dentro de las Ciencias Computacionales, ya que un Agente se puede definir como un sistema informático situado en un entorno, en el cual es capaz de realizar acciones flexibles y autónomas para poder alcanzar sus objetivos [2].

Los agentes inteligentes poseen características tales como:

Autonomía. Un agente es capaz de realizar una acción autónoma flexible en algún ambiente.
Reactivo: Reacciona a los cambios en el medio ambiente. Mantiene una interacción en curso con su medio ambiente, y responde a los cambios que ocurren en él.
Pro-Activo. Genera y trata de llegar a sus metas, no es conducido solamente por los acontecimientos, toma la iniciativa y también reconoce oportunidades.
Habilidades Sociales. Capacidad de comunicarse con otros agentes por medio de algún lenguaje de comunicación.

Las características de las *Habilidades Sociales* se pueden clasificar como sigue:

1. Cooperación
2. Competencia
3. Negociación

Basándonos en lo anterior, se pretende diseñar e implementar agentes que sean capaces de autoevaluarse, autoaprender y cambiar su acción de acuerdo a su medio ambiente.

2.3 Sistemas Multi-Agentes

Ahora bien, una vez descrita en la sección 2.2 de manera formal la definición de un agente y las características que este posee, también se puede aunar a esto, que los agentes generalmente operan en medios ambientes en los cuales interactúan, y posiblemente cooperan, con otros agentes. A estos ambientes se les conoce como sistemas multi-agentes (SMA).

Un sistema multi-agente se puede definir como varios agentes interactuando para conseguir sus metas individuales o una sola meta común global, acordada por todos ellos de común acuerdo.
Hay sistemas multi-agentes que pueden ser vistos constituyendo un sólo agente, pero otros son clasificados mejor como sociedades de agentes. Por ejemplo; cuando una colección de

agentes planificadores se reúne para programar una reunión entre sus usuarios, ellos persiguen una meta común y un comportamiento de grupo inteligente emerge. Cuando la planificación se completa, los agentes se dispersan, quizás para nunca reunirse otra vez en el mismo grupo [3].

2.4 Arquitectura Orientada a Servicios

De acuerdo a David De Roure et al. [4]: en una AOS existen entidades que proporcionan servicios bajo varias formas de contrato. El propósito de tal modelo es identificar los componentes claves y especificar como se relacionan el uno con el otro. Un servicio puede ser visto como una encapsulación y caracterización abstracta de algún contenido, así también todos los servicios tienen un propietario o un conjunto de propietarios; el propietario en este caso es el cuerpo (individuo o institución), el cual es el responsable de ofrecer servicios para el consumo de otros. El propietario establece los términos y condiciones bajo las cuales el servicio puede ser accedido.

Al ofrecer un servicio para el consumo de otros, el propietario espera atraer a los consumidores para ofrecerles dicho servicio, los consumidores serán las entidades que decidirán hacer un intento por invocar el servicio que requieren o necesitan.

Por otra parte se plantea una relación entre el consumidor y el propietario del servicio, la cual está dada a través de un contrato; en éste se especifican los términos y condiciones bajo las cuales el propietario del servicio provee el servicio al consumidor.
Así también, en esta arquitectura se describe un ambiente particular al cual denominan Mercado, y la entidad que regula el mercado es nombrada como propietario del mercado.

Los componentes con que cuenta una AOS son los que siguen:
- Los propietarios del servicio, “propietario de un tipo de servicio”.
- Los consumidores de servicios, “solicitan un tipo de servicio”.
- Propietario del mercado, “establece las reglas, lleva a cabo el registro del servicio y garantiza la transacción en el mercado”, ver figura 2.2.

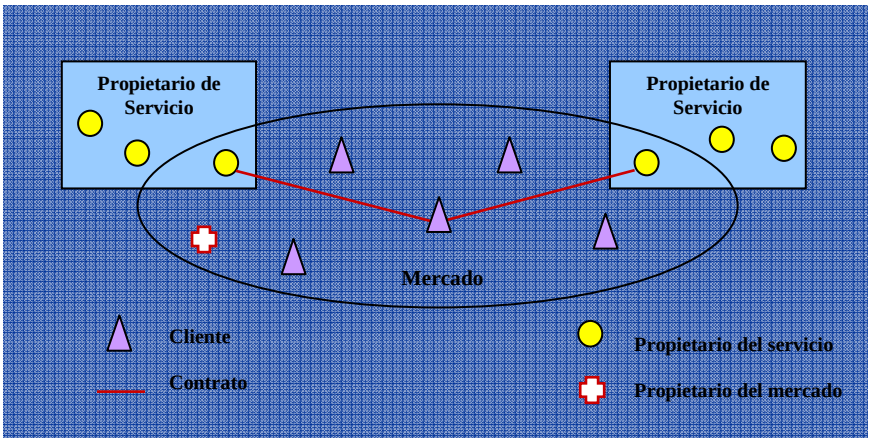

Figura 2.2.Modelo del Mercado

Para finalizar, de acuerdo a David De Roure et al. [4] en este modelo que se describe, los servicios tienen un ciclo de vida que constan de las siguientes etapas:

1. Creación de un servicio. El propietario del servicio define el servicio que quiere poner a la disposición de los demás. Todo el tiempo se están creando y destruyendo servicios, lo cual hace que el sistema sea dinámico. El proceso de creación de un servicio involucra tres actividades, algunas de las cuales podrían automatizarse:

- La especificación de cómo se realizará el servicio en un lenguaje apropiado. Esta parte en general no está disponible a la vista externa. En términos de agentes ésta seria la parte correspondiente a la creación de un agente.
- La especificación de la meta-información asociada con el servicio (e.g. la especificación de quien puede usar el servicio y las posibles formas en que puede satisfacerse).
- Hacer disponible el servicio en el mercado apropiado. Esto requiere de propaganda para el servicio y facilidades de registro en el mercado.

2. Procuración del servicio. Este involucra un propietario y un consumidor del servicio, estableciendo un contrato para legalizar que el servicio acordado se proporcione en los términos y condiciones convenidos. Aquí se consideran un número de puntos acerca de este proceso:

- El proveedor puede fallar.
- En muchos casos, dado que los agentes propietarios del servicio y consumidores del servicio pueden representar a usuarios con intereses no totalmente compatibles, el contrato se firma después de un proceso de negociación.
- La negociación puede ser hecha fuera de línea por los usuarios propietarios de los agentes, o bien en línea por los propios agentes.

3. Cumplimiento del Contrato. Aquí el proveedor del servicio lleva a cabo las acciones necesarias para cumplir con sus obligaciones. Al finalizar el proveedor debe informar al consumidor su cumplimiento con el contrato. Cuando el proveedor se vea imposibilitado de cumplir con el contrato, éste se podrá renegociar, modificando los términos y condiciones.

2.5 Bases de Datos

Una Base de datos se define como una colección de información organizada y relacionada entre si, cuya estructura es similar a una matriz (tabla) con filas (registros o entradas) y columnas (campos) [5]. La intersección de una fila con una columna se le denomina celda, el tipo de información de un campo se conoce como atributo. En la figura 2.3 se muestra un ejemplo gráfico de lo descrito anteriormente.

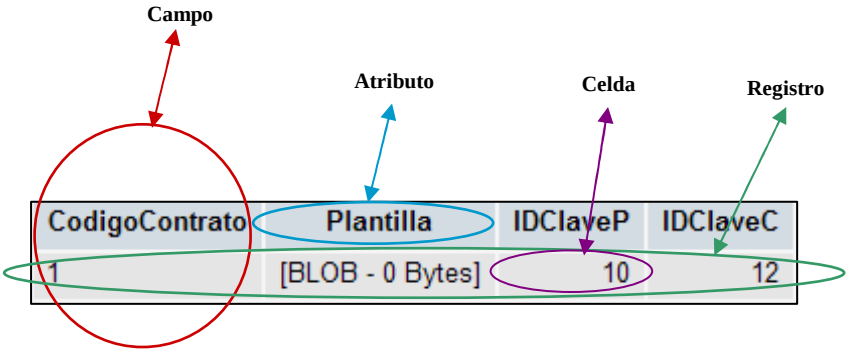

Figura 2.3 Tabla de una Base de Datos

Los valores que toman o que son asignados a las celdas pertenecen a un dominio [5]. Una relación dentro de una base de datos, se forma por la condiciones de la información dentro de la base de datos, esto nos permite tener información referenciada a otras tablas y conectar los atributos de una tabla a otra(s).

Para ello se debe definir un campo que será el responsable de dicha tarea dentro de la tabla que quiera ser relacionada con otra, este campo se denomina clave, la clave en una tabla puede ser primaria (primary key) o secundaria (foreign key), la primera sirve para identificar de forma única a cada tupla o registro en una relación [5], la segunda será la clave que conectará la clave primaria de la tabla que se quiera relacionar con la tabla que contenga esta clave secundaria, asemejando el proceso de herencia.

Se recomienda que las tablas de una base de datos contengan siempre un campo que represente la clave primaria, ya que ello permite identificar de manera única los registros en cada tabla y de esta forma tener un mejor desempeño en la base de datos.

Para diseñar una base de datos se utiliza el modelo Entidad-Relación (ER). Este modelo fue introducido por Codd en 1970 [6], se basa en una estructura de datos simples y uniforme – la relación – y tiene fundamentos teóricos sólidos. Es una forma particular de estructurar y procesar una base de datos.
El modelo relacional se ha establecido como el primer modelo de datos para las aplicaciones de procesamiento de datos.

2.5.1 El Modelo Relacional

El modelo relacional se basa en dos ramas de las matemáticas: la teoría de conjuntos y la lógica de predicados de primer orden. El hecho de que el modelo relacional esté basado en la teoría de las matemáticas es lo que lo hace tan predecible, seguro y robusto. Al mismo tiempo, estas ramas de las matemáticas proporcionan los elementos básicos necesarios para crear una base de datos relacional con una buena estructura, y proporcionan las líneas que se utilizan para formular buenas metodologías de diseño [6].

Una relación es una tabla con columnas y filas. Un Sistema Gestor de Bases de Datos (SGBD) sólo necesita que el usuario pueda percibir la base de datos como un conjunto de tablas. Esta percepción sólo se aplica a la estructura lógica de la base de datos y no se

aplica a la estructura física de la base de datos, que se puede implementar con distintas estructuras de almacenamiento [6].

Un atributo es el nombre de una columna de una relación. En el modelo relacional, las relaciones se utilizan para almacenar información sobre los objetos que se representan en la base de datos. Una relación se representa gráficamente como una tabla bidimensional en la que las filas corresponden a registros individuales y las columnas corresponden a los campos o atributos de esos registros. Los atributos pueden aparecer en la relación en cualquier orden [6].

Un dominio es el conjunto de valores legales de uno o varios atributos. Los dominios constituyen una poderosa característica del modelo relacional. Cada atributo de una base de datos relacional se define sobre un dominio, pudiendo haber varios atributos definidos sobre el mismo dominio.

Capítulo 3

Modelo de Análisis

Para lograr una localización y distribución exitosa de los OA es necesario resolver una serie de problemas asociados a los OA mediante las siguientes preguntas:

¿Cómo crearlos?
¿Cómo almacenarlos?
¿Cómo encontrarlos?
¿Cómo usarlos?
¿Cómo actualizarlos?

Para dar respuesta a cada pregunta se requiere de una investigación exhaustiva, inclinada fundamentalmente a dos campos importantes que proveen las soluciones a estas dificultades: la educación y la tecnología; educación porque todos los OA contienen material educativo elaborado por personas expertas en determinadas áreas de la educación, y tecnológica porque para la elaboración de los OA se requiere el uso si no de todas, de algunas tecnologías existentes, que apoyan en la creación del OA, su almacenamiento, su localización, su uso y su actualización.

Nuestro proyecto de tesis se centra en dar solución a dos de los planteamientos antes mencionados: el almacenamiento, y la localización de OA.
De acuerdo con lo anterior el objetivo general de nuestro trabajo es desarrollar un sistema de software que permita la promoción, localización y acceso controlado de diversos OA.

Por consiguiente, la idea esencial del proyecto es contar con un repositorio de OA que permita a diversos usuarios poder utilizarlo para el diseño de sus cursos; la construcción de este repositorio puede realizarse de las siguientes maneras:

- Modelo de repositorio centralizado
- Modelo de repositorio distribuido y abierto
- Modelo de repositorio distribuido y controlado

Modelo de repositorio centralizado

Consiste en una base de datos local ubicada en un servidor centralizado, esta base de datos contendría los OA que van a ser publicados y localizados por los diferentes usuarios involucrados en la publicación y búsqueda de OA, esta base de datos de OA se encontraría administrada por un usuario denominado administrador, ver figura 3.1.

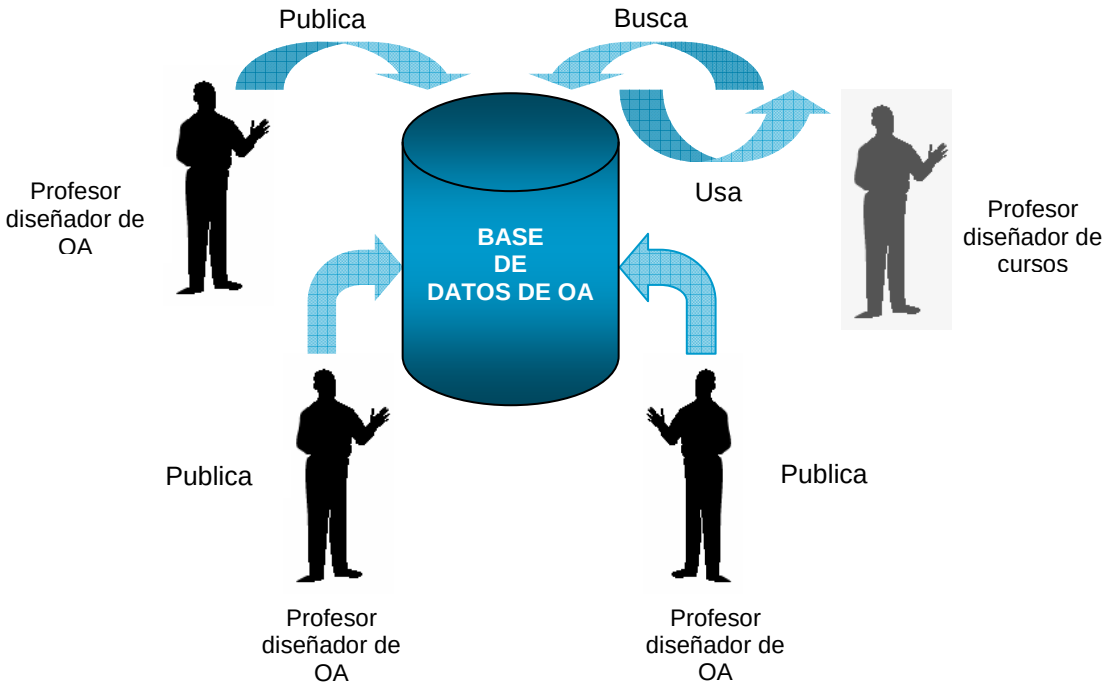

Figura 3.1 Modelo Centralizado de Almacenamiento y Localización de OA

Ventajas del modelo:

- Permitiría tener un control sobre los OA almacenados, así como también de los usuarios que los publican y acceden.
- Facilitaría la compartición de OA.

Desventajas del modelo:

- No es tan confiable, ya que cualquier falla en el servidor afectaría el acceso al repositorio.
- No cubriría por completo las necesidades de los usuarios proveedores y consumidores.
- Los tiempos de respuesta no serian los adecuados, ya que la base de batos de OA estaría compartida por todos los usuarios.

Modelo de repositorio distribuido y abierto

El modelo de repositorio distribuido y abierto permitiría a los usuarios proveedores diseñadores de OA publicar dichos objetos a través de la red, por medio de una página web y de la misma manera permitiría a los usuarios consumidores realizar un acceso libremente a través de un buscador de Internet para localizar el OA deseado, ver figura 3.2.

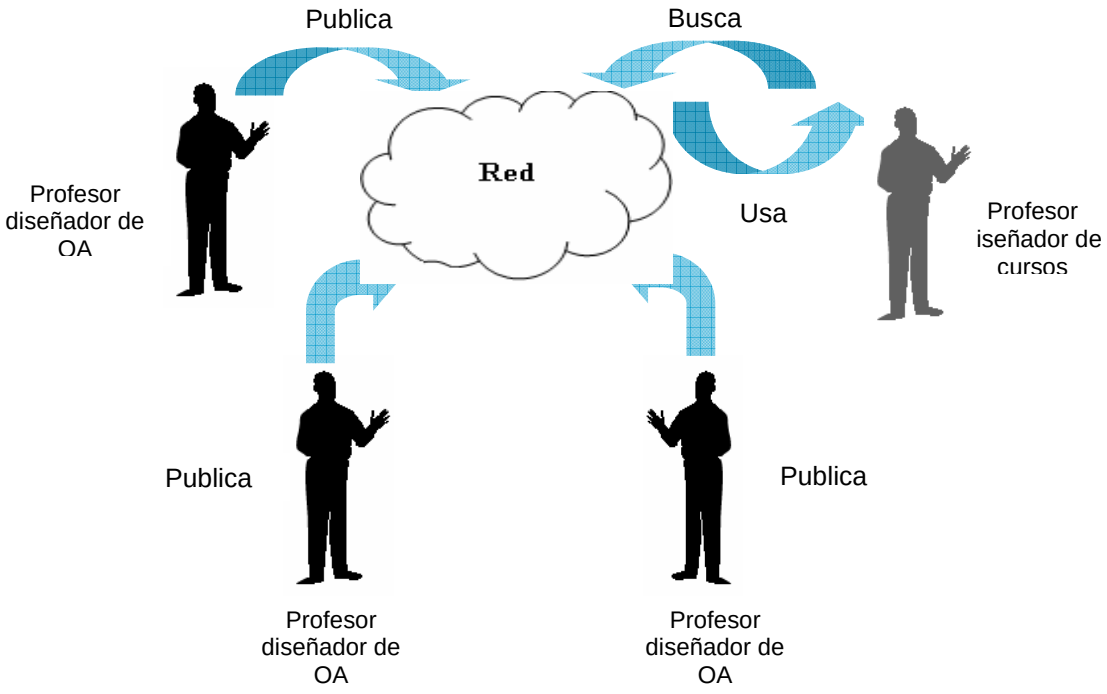

Figura 3.2 Modelo Distribuido y Abierto de Almacenamiento y Localización de OA

Ventajas del modelo:

- Permitiría que los recursos (OA) se entreguen al usuario de manera transparente.
- Admitiría la compartición de OA.
- Proveería un alto rendimiento en cuanto al procesamiento de los datos.
- Tiene características de escalabilidad, fiabilidad, es tolerante a fallos y es de carácter abierto.

Desventajas del modelo:

- No existiría ningún control acerca de quien accede y hace uso de los OA, por lo tanto cualquier actualización no podría distribuirse uniformemente.
- No existiría la certeza de que los OA posean la calidad requerida, de acuerdo a una estandarización y por consiguiente se tendría la incertidumbre de que los OA tengan la característica de interoperabilidad, debido a que cualquier persona ajena puede almacenar cualquier OA en las bases de datos distribuidas, sin cumplir con los estándares requeridos.

Modelo de repositorio distribuido y controlado

Los repositorios distribuidos y controlados al igual que el modelo de repositorio distribuido y abierto, son repositorios que pueden ser accedidos a través de Internet pero con la gran diferencia de que estos se van a regir bajo ciertas reglas. Esto nos permitiría por un lado tener el control sobre quien alimenta los repositorios, y por otro, quien esta autorizado a accederlos. Este control no pretende ser una limitación para el compartimiento de los OA, sino para asegurar su calidad, su estandarización y su interoperabilidad.

3.1 Hacia un Modelo de Mercado de Objetos de Aprendizaje

Una vez descrita en la sección 2.4 la Arquitectura Orienta a Servicios (AOS), se realiza la siguiente propuesta, describiendo previamente el objetivo y las necesidades del proyecto.

El objetivo general de este proyecto es poder desarrollar un sistema para la promoción, localización y acceso controlado de OA, atendiendo las siguientes necesidades:

- Distribución de OA en una red.
- Se requiere que los OA cumplan con estándares internacionales.
- Servicio de acceso controlado a los OA.
- Establecimiento de diferentes permisos de uso.

Por ello se propone edificar repositorios de OA usando la Arquitectura Orientada a Servicios, donde las instituciones educativas fungirán como nuestros consumidores y nuestros proveedores de OA, dichos OA deberán estar distribuidos en los diferentes servidores de las instituciones educativas proveedoras.

3.2 Modelo de Mercado de Objetos de Aprendizaje

Para darle solución a la propuesta citada en la sección 3.1, se propone un modelo de Mercado de Objetos de Aprendizaje, basado en la arquitectura Orientada a Servicios.

Tomando como referencia la AOS tendremos a:

- Los proveedores: Tendrán la tarea de poner a disposición de los diseñadores de cursos diversos OA, sobre los cuales mantendrán ciertos derechos y políticas de acceso.
- Los consumidores: Tendrán las facilidades para localizar, acceder y negociar condiciones de uso sobre los OA.

- Propietario de mercado: Tendrá la tarea de controlar toda transacción efectuada en el mercado de objetos de aprendizaje.

Tendremos entonces, a tres entidades fundamentales, la primera entidad proporcionará servicios a otras entidades consumidoras, que usarán los servicios proveídos, para nuestros efectos son los OA.

Utilizando la tecnología de agentes representamos a estas tres entidades como agentes de software, ahora los proveedores de objetos serán agentes de software (agentes proveedores) que representarán a un usuario proveedor o a una institución proveedora de OA. Los consumidores de objetos de aprendizaje serán agentes de software (agentes consumidores) quienes solicitarán el servicio de búsqueda de OA, proveídos por algún agente proveedor representando a un usuario o a una institución proveedora.

En base al modelo de De Roure et al. [4] podemos proveer los siguientes servicios:

- Ofrecer OA
- Encontrar los OA y;
- Acceder a los OA.

De esta manera tanto consumidores como proveedores podrán converger de forma controlada al mercado para satisfacer sus necesidades, no siendo necesario que tanto los consumidores como los proveedores se conozcan previamente para poder realizar alguna transacción, ya que de acuerdo con el modelo de De Roure et al, estos mercados actúan no sólo como introductores, sino como reguladores de las relaciones que puedan establecerse entre ellos.

Por último para que el cliente consumidor tenga acceso a un OA determinado se debe establecer un proceso de negociación, donde se establezcan los términos y las condiciones para lograr el acceso. Una vez que ambas entidades llegan a un acuerdo, estos acuerdos deben ser legalizados en un contrato, donde no sólo se especifiquen los términos y las

condiciones de uso, sino también las obligaciones, los beneficios y las penalizaciones por incumplimiento de algún (s) término (s).

De acuerdo a De Roure et al.[4] los agentes de software son un paradigma ideal para representar proveedores, consumidores y propietarios de servicios y los sistemas multi-agentes podrán usarse para modelar los mercados.

De acuerdo con esto, y teniendo en claro todos los beneficios que nos ofrece la AOS, podemos permitirnos crear el modelado de un mercado que provea el servicio de promoción, localización y acceso controlado de objetos de aprendizaje, al cual denominaremos Mercado de Objetos de Aprendizaje (MOA).

3.2.1 Arquitectura General del MOA visto como un Sistema Multi-Agente

Hemos desarrollado un modelo general del mercado de objetos de aprendizaje, que muestra los componentes de la AOS implementados, utilizando el Paradigma Orientado a Agentes. Para proveer el servicio de promoción, localización y acceso controlado al MOA se propone un sistema multi-agente compuesto por:

- Agentes Proveedores (AP)
- Agentes Consumidores (AC)
- Agente Administrador (AA)

Nuestro trabajo se centra primordialmente en poder atender las solicitudes de varios AP y AC simultáneamente, esto se logra por medio del AA, quien controla todo el funcionamiento del mercado de objetos de aprendizaje, así mismo, el AA delega subtareas a agentes independientes (agente de registro, agente localizador de OA, agente manejador de contratos, agente publicador, agente eliminador de agentes, y agente eliminador de OAs), que convenientemente crea según la solicitud de servicio que haya recibido. En esté sentido puede verse al MOA como un sistema multi-agente, ya que se tienen a varios agentes interactuando entre sí, para conseguir sus metas individuales, con una meta común

global que consiste en el buen funcionamiento del mercado. La arquitectura del Mercado de Objetos de Aprendizaje se puede ver en la figura 3.3.

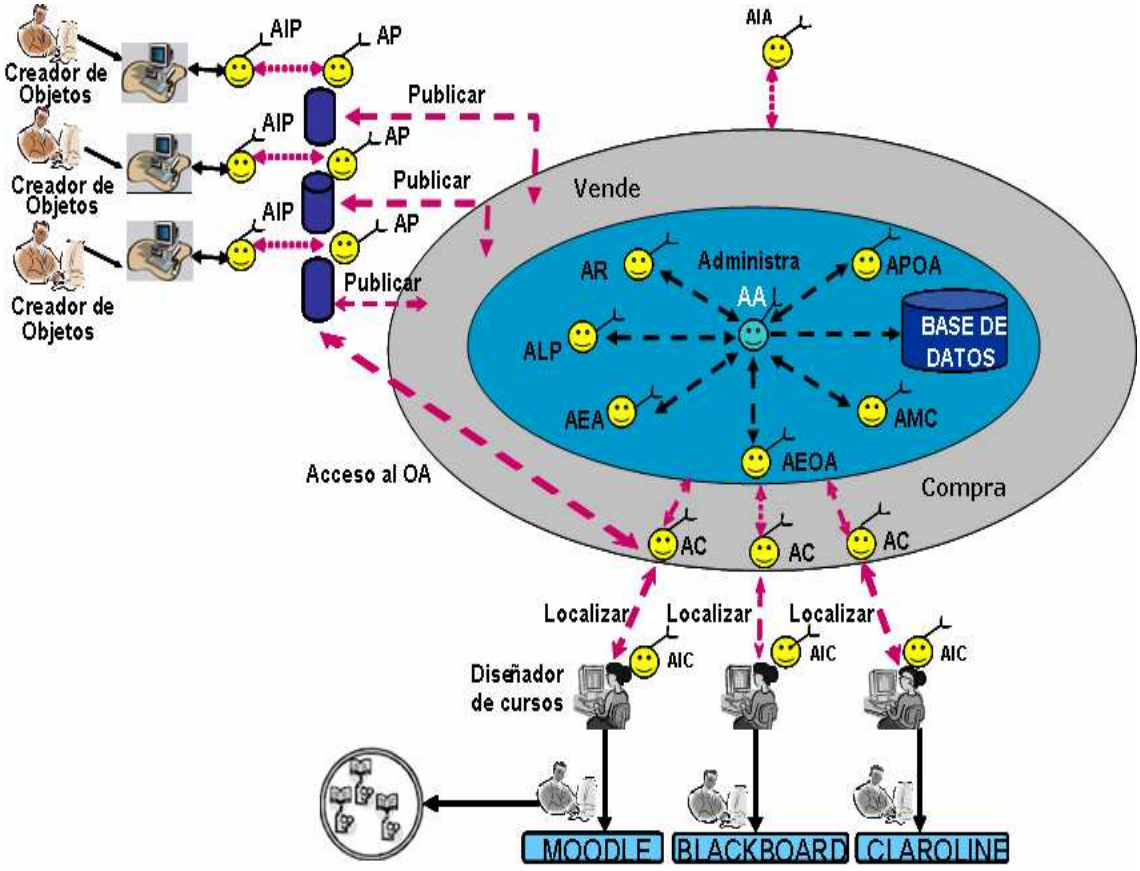

Figura 3.3 Modelo General del Mercado de Objetos de Aprendizaje

Es importante destacar que en base al modelo propuesto se tienen agentes residentes en el mercado y agentes externos al mercado, por tanto, el sistema multi-agente del mercado de objetos de aprendizaje estará compuesto por:

- Agente Administrador (AA)
  - Agente Interfaz del Agente Administrador (AIA)
- Agente de Registro (AR)
- Agente Localizador de Proveedores de OA (ALP)
- Agente Manejador de Contratos (AMC)
- Agente Publicador de OA (APOA)

- Agente Eliminador de Agentes (AEA)
- Agente Eliminador de OA (AEOA) y;
- Agente Consumidor (AC)

Estos agentes nombrados con anterioridad residen en el mercado, sin embargo se tienen otros agentes que residen en las máquinas de los proveedores y consumidores respectivamente:

- Agente Interfaz del Consumidor (AIC).
- Mini Agente Reportador (MAR) (no visible al usuario).
- Agente Proveedor (AP).
    - Agente de Interfaz del Proveedor (AIP).

3.2.2 Metas y Tareas del Sistema Multi-Agente

Agente Consumidor (AC)

Meta 1. Registrarse en el MOA

Tarea1. Buscar MOA.
Tarea 2. Enviar lista de MOA´s a la interfaz.
Tarea 3. Realizar petición de registro al AR.
Tarea 4. Enviar plantilla de registro para llenar a la interfaz, y plantilla de políticas.
Tarea 5. Enviar plantilla llena con datos al AR.
Tarea 6. Enviar clave de acceso a la interfaz.

Meta 2. Localizar OA

Tarea 1. Realizar solicitud de búsqueda de proveedores al MOA.
Tarea 2. Enviar datos de la búsqueda al ALP.
Tarea 3. Enviar lista de proveedores a la interfaz.

Meta 3. Negociar Contrato

Tarea 1. Contactar con el AP.

Tarea 2. Enviar propuestas al AP.

Tarea 3. Negociar.

Tarea 4. Enviar contrato al Agente Interfaz.

Tarea 5. Registrar contrato en el mercado.

Tarea 6. Recibir dirección del AA donde recogerá el OA, una vez que haya realizado el pago pactado en el contrato.

Meta 4. Darse de Baja del MOA

Tarea 1. Enviar solicitud de baja al MOA.

Agente Interfaz del Consumidor (AIC)

Meta 1. Esperar Acciones del Usuario

Tarea 1. Esperar y responder a solicitudes realizadas por el usuario.

Tarea 2. Guardar en un archivo clave de acceso, nombre del mercado y datos del usuario consumidor/proveedor.

Meta 2. Mostrar Contrato

Meta 3. Mostrar Políticas de Ingreso y Permanencia en el MOA

Metas de Usuario Asociadas a la Interfaz del Agente Consumidor

Meta 1. Registrar en el MOA

Tarea 1. Enviar solicitud de registro al AC.

Tarea 2. Leer plantilla de registro y llenarla.

Tarea 3. Aceptar las políticas del MOA.

Tarea 4. Enviar plantilla de registro al AC.

Meta 2. Petición de un OA

Tarea 1. Petición de búsqueda de OA al AC.

Tarea 2. Elegir OA de algún proveedor (proponer precio).

Tarea 3. Firmar contrato o denegar el acuerdo.

Tarea 4. Notificar al AC donde y como almacenará el OA, una vez que lo haya recuperado.

Meta 3. Darse de Baja del MOA

Tarea 1. Petición de baja.

Mini Agente Reportador (MAR)

Meta 1. Generar Nombre del Agente Consumidor

Tarea 1. Generar Nombre.

Meta 2. Comunicar al MOA el Nombre del AC

Tarea 1. Enviar nombre al Agente Administrador.

Agente Administrador (AA)

Meta 1. Crear el MOA

Tarea 1. Creación del mercado.

Tarea 2. Aplicar las políticas de ingreso.

Meta 2. Crear a los Agentes Involucrados en el Sistema Multi-Agente

Tarea 1. Crea al AR, una vez que recibe un mensaje de solicitud de registro por parte de un agente consumidor o proveedor en representación de su usuario.

Tarea 2. Crea al ALP, una vez que recibe un mensaje de solicitud búsqueda de proveedores de OA por parte de consumidor en representación de su usuario.

Tarea 3. Crea al AMC de forma permanente, mientras el mercado permanezca activo.

Tarea 4. Crea al APOA, una vez que recibe un mensaje de solicitud publicación por parte de un agente proveedor.

Tarea 5. Crea al AEA, una vez que recibe un mensaje de solicitud baja de agente por parte de un agente consumidor o proveedor.

Tarea6. Crea al AEOA, una vez que recibe un mensaje de solicitud baja de OA por parte de un agente proveedor.

Meta 3. Administrar el MOA

Tarea 1. Actualizar el historial de proveedores y consumidores en base a comportamientos derivados de los contratos.

Tarea 2. Notificar suspensión de proveedores.

Tarea 3. Aplicar sanciones.

Tarea 4. Administración de la base de datos (proveedores, consumidores y contratos).

Meta 4. Entregar OA al AC

Tarea 1. Monitorear registro de nuevos OA en la BD.

Tarea 2. Verificar estado de pago de OA del AC.

Tarea 4. Proporcionar clave y password para poder extraer el OA de la BD.

Tarea 5. Borrar OA una vez que haya accesado a el, el AC.

Meta 5. Esperar Solicitud por parte del Agente Reportador de la Creación de un AC dentro del MOA.

Tarea 1. Recibir solicitud con nombre del Agente Consumidor a crear.

Tarea 2. Lanzar al agente consumidor dentro del MOA, para que pueda comunicarse con su Interfaz.

Agente Interfaz del Administrador (AIA)

Meta 1. Esperar Acciones del Usuario

Meta 2. Acceder de Manera Remota a la Base de Datos del Mercado para Responder a las Consultas del Usuario

Tarea 1. Conectarse a la Base de Datos.

Tarea 2. Recuperar la información almacenada en la BD.

Metas del Usuario Asociadas a la Interfaz del Administrador

Meta 1. Consultar Información de Proveedores Registrados en el MOA

Meta 2. Consultar Información de Consumidores Registrados en el MOA

Meta 3. Consultar Información de OA Almacenados en la BD del MOA

Agente de Registro (AR)

Meta 1. Registrar a los Agentes Proveedores y Consumidores en el MOA

Tarea 1. Proporcionar requisitos al AC/AP.

Tarea 2. Verificar cumplimiento de requisitos.

Tarea 3. Generar clave de acceso.

Tarea 4. Solicitar registro de datos al AA.

Tarea 5. Enviar información de registro al AC/AP.

Tarea 6. Enviar información de solicitud rechazada.

Agente Publicador (APOA)

Meta 1. Publicar OA

Tarea 1. Autenticar al usuario.

Tarea 2. Verificar solicitud de publicación.

Tarea 3. Generar código de OA.

Tarea 4. Solicitud de registro de datos al AA.

Tarea 5. Enviar información al AP.

Tarea 6. Enviar información de rechazo ingreso/publicación rechazada.

Agente Eliminador OAs (AEOA)

Meta 1. Dar de Baja un OA

Tarea 1. Autenticar al usuario.

Tarea 2. Verificar solicitud de baja.

Tarea 3. Solicitar baja de OA al AA.

Tarea 4. Enviar información de rechazo de ingreso al AP.

Tarea 5. Enviar información de Baja al AP.

Tarea 6. Enviar información de rechazo baja al AP.

Agente Eliminador de Agentes (AEA)

Meta 3. Dar de Baja a un Consumidor/ Proveedor

Tarea 1. Autenticar al usuario.

Tarea 2. Verificar solicitud de baja del consumidor/proveedor.

Tarea 3. Solicitar baja de consumidor/proveedor al AA.

Tarea 4. Enviar información de baja al AC/AP.

Tarea 5. Enviar información de rechazo de ingreso/rechazo de baja.

Agente Localizador de Proveedores de OA (ALP)

Meta 1. Localizar a los Proveedores de OA

Tarea 1. Autenticar al usuario.

Tarea 2. Realiza búsquedas en la base de datos del mercado de un proveedor de OA, mediante la descripción del OA.

Tarea 3. Proveer al AC la lista de proveedores de OA que solicitó.

Tarea 4. Enviar información de rechazo de ingreso/proveedor no encontrado.

Agente Manejador de Contratos (AMC)

Meta 1. Enviar Contrato de OA al Agente Proveedor

Tarea 1. Autenticar al usuario.

Tarea 2. Adjuntar folio al contrato.

Tarea 3. Enviar documento de contrato al AP.

Tarea 4. Enviar información de rechazo de ingreso al AP.

Meta2. Registrar Contratos en el MOA

Tarea 1. Autenticar al usuario.

Tarea 2. Analizar contratos (proveedor/consumidor).

Tarea 3. Comparar contratos.

Tarea 4. Solicitar guardado de contrato en la BD local del MOA al AA.

Tarea 5. Enviar información de rechazo de ingreso.

Tarea 6. Envío de información a los agentes (éxito/fracaso).

3.3 Interacción en el Sistema Multi-Agente

Para el buen funcionamiento del MOA debe existir un proceso de interacción entre los agentes, ya que las tareas que realizan cada uno de ellos no están aisladas, es decir, la ejecución de una implica la culminación o iniciación de otra. A continuación se describe dicho proceso de interacción.

El usuario creador de OA es representado por un AP, este creador pone a disposición de consumidores sus objetos a través de dicho agente. Tanto el AP como el usuario creador de objetos interactúan con un agente de interfaz denominado AIP para comunicarse entre sí. El AP y el AIP no forman parte del Mercado, generalmente se encuentran en una máquina remota, cualquier acción o solicitud que desee realizar el usuario creador es enviada al AP mediante el AIP y será el AP quien se comunique directamente con el Mercado y viceversa, el AP se comunica con el AIP para hacerle llegar al usuario creador los resultados obtenidos a dichas acciones o solicitudes.

Por otra parte, un usuario consumidor de OA es representado por un AC que se encarga de obtener los objetos que el consumidor requiere para crear sus cursos. Tanto el AC como el usuario consumidor de objetos interactúan con un agente de interfaz denominado AIC para comunicarse entre sí. El AC radica dentro del MOA mientras que su AIC se localiza en la máquina del usuario consumidor por lo que la comunicación entre ambos agentes se realiza de forma remota; cualquier acción o solicitud que desee realizar el usuario creador es enviada al AC mediante el AIC y será el AC quien se comunique directamente con el Mercado y viceversa, el AC se comunica con el AIC para hacerle llegar al usuario consumidor los resultados obtenidos a dichas acciones o solicitudes.

Los objetos que son proporcionados o publicados por el AP para su uso son almacenados en un repositorio, el control y administración de los OA está a cargo del AA.

El AA se encuentra en estado de ejecución en todo momento, debido a que es el encargado de recibir solicitudes de AP y AC, de tal forma que verifica el tipo de petición y crea al agente correspondiente (agente de registro, agente localizador de OA, agente manejador de contratos, agente publicador, agente eliminador de agentes, y agente eliminador de OAs) para atender a esa solicitud.

En la estructura social del MOA es posible que AP y AC de OA tengan diferentes intereses, por lo que antes de que un OA sea entregado a un consumidor, se iniciará un proceso de negociación sobre el uso que se le dará al mismo y que culminará con un contrato, en donde se especificarán los términos y condiciones del uso del OA; también las obligaciones, los beneficios y las posibles penalizaciones en caso de incumplimiento.

3.4 Lenguaje Unificado de Modelado (UML)

Para poder llevar a cabo el modelado y desarrollo de nuestro sistema de software se sigue la metodología del Proceso Unificado de Desarrollo de Software (PUDS), permitiéndonos realizar modelados de Análisis y Diseño. El proceso Unificado de Desarrollo de Software utiliza el Lenguaje Unificado de Modelado (UML) [7]; el cual es un lenguaje gráfico que permite visualizar, especificar, construir y documentar los artefactos de un sistema con gran cantidad de software.

UML proporciona una forma estándar de escribir los planos de un sistema, cubriendo tanto los aspectos conceptuales, tales como procesos de un protocolo y funciones del mismo, como los aspectos concretos, es decir las clases escritas en un lenguaje de programación especifico, esquemas de bases de datos y componentes de software reutilizables.

El modelo UML se basa básicamente en diagramas, donde cada diagrama es la representación gráfica de un conjunto de elementos, visualizado la mayoría de las veces como un grafo conexo de nodos (elementos) y arcos (relaciones). Estos usan la notación pertinente, y la suma de estos diagramas crean las diferentes vistas donde resumen los elementos que constituyen un sistema.

Casos de Uso

Para construir un sistema con éxito hay que conocer las necesidades y deseos de los futuros usuarios, de esta manera surgen tres características fundamentales:

- Usuarios: Personas que trabajan y necesitan el sistema.
- Interacción: El usuario interactúa con el sistema para satisfacer un servicio requerido.
- De interacción a Caso de Uso: Fragmento de funcionalidad del sistema que proporciona al usuario un resultado importante.

El modelo de casos de uso representa la entrada a nuestro sistema, permite especificar los requisitos del sistema: representan los requisitos funcionales y juntos constituyen el modelado de casos de uso que describen la funcionalidad total del sistema a desarrollar, pero el punto clave es que permiten ser una guía en el proceso de desarrollo (diseño, implementación y prueba).

3.5 Casos de Uso del Mercado de Objetos de Aprendizaje

El diagrama de casos de uso para proveer servicios a diversas entidades externas al MOA se muestra en la figura 3.4. En el se observa la existencia de dos actores, el Consumidor Agent y el Proveedor Agent; quienes harán uso de los servicios que el mercado ofrece. Así mismo, se puede observar la presencia de nueve servicios que hasta el momento ofrece el MOA.

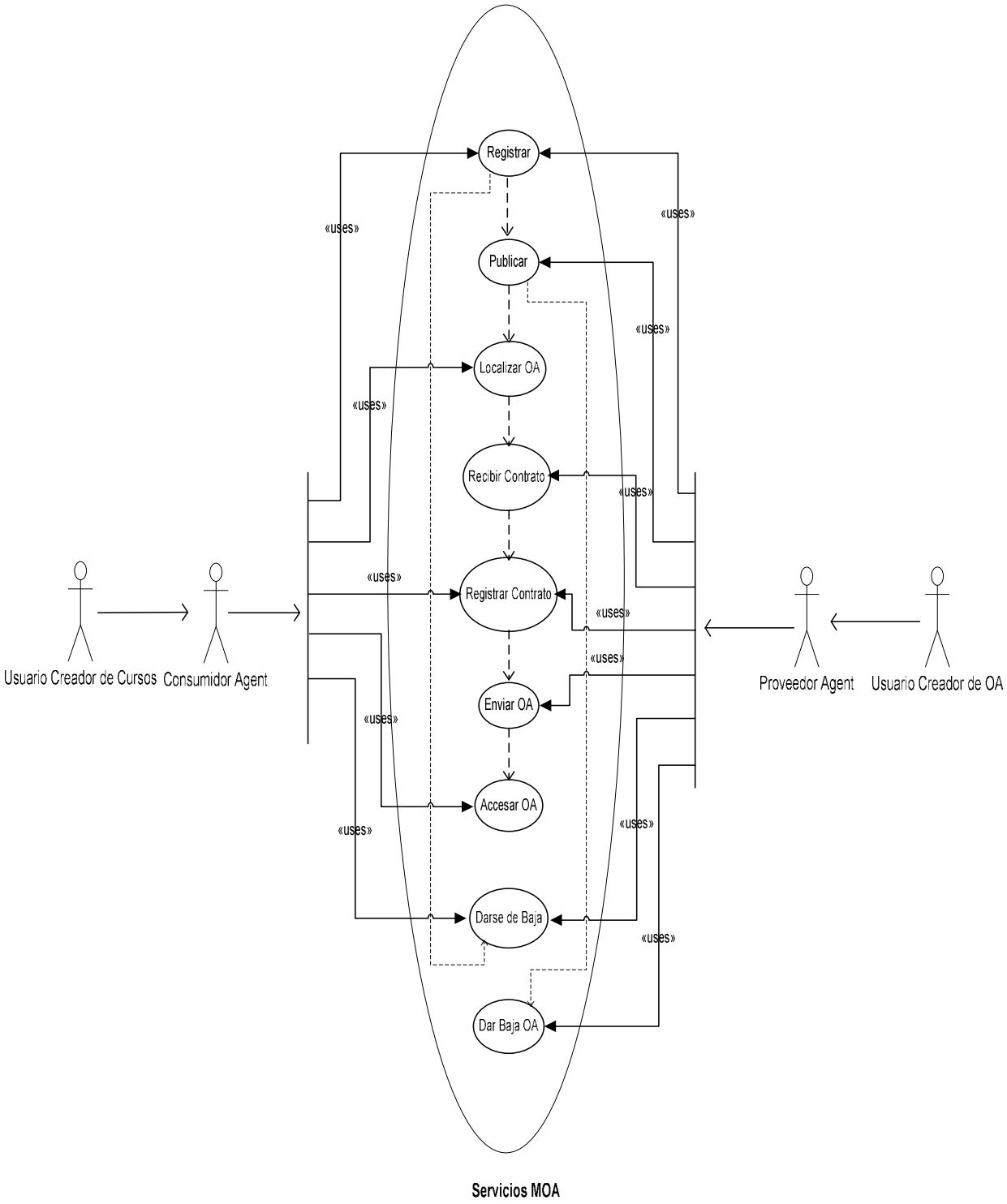

Figura 3.4 Casos de Uso del Mercado de Objetos de Aprendizaje

Caso de uso: Registrar

Descripción: Permite ofrecer el servicio de registro en el mercado de objetos de aprendizaje, a un agente consumidor o proveedor, representante de un usuario.
Actor (s): Consumidor agent, Proveedor agent.

Caso de uso: Publicar

Descripción: Permite ofrecer el servicio de publicación de un OA en el mercado, a un agente proveedor, representante de un usuario creador de OA.
Actor (s): Proveedor agent.

Caso de uso: Localizar OA

Descripción: Ofrece el servicio de localización de proveedores de OA en el mercado, a un agente consumidor, representante de un usuario creador de cursos. Este caso de uso presenta como resultado la lista de proveedores que poseen el OA solicitado.
Actor (s): Consumidor agent.

Caso de uso: Recibir Contrato

Descripción: Permite ofrecer el servicio de envío de contrato al agente proveedor, este contrato a su vez será llenado por los agentes que hayan llegado a un acuerdo de negociación por un OA deseado.
Actor (s): Proveedor agent.

Caso de uso: Registrar Contrato

Descripción: Ofrece el servicio de registro de contratos en el mercado, al agente proveedor y al agente consumidor representantes de un usuario. El propósito de este

servicio es tener un control sobre la recepción y la entrega del OA negociado, una vez que se hayan llenado y firmado los contratos.

Actor (s): Consumidor agent, Proveedor agent.

Caso de uso: Enviar OA

Descripción: Permite recibir el OA por parte del agente proveedor, para ser entregado a un agente consumidor.

Actor (s): Proveedor agent.

Caso de uso: Accesar OA

Descripción: Permite a un agente consumidor accesar al mercado para recoger su objeto de aprendizaje.

Actor (s): Consumidor agent

Caso de uso: Darse de Baja

Descripción: Este caso de uso permite dar de baja a un agente consumidor o proveedor del mercado de objetos de aprendizaje, cuando el usuario creador de cursos o creador de OA así lo desee.

Actor (s): Consumidor agent, Proveedor agent.

Caso de uso: Dar Baja OA

Descripción: Permite ofrecer el servicio de baja de un OA en el mercado, a un agente proveedor representante de un usuario creador de OA.

Actor (s): Proveedor agent.

3.6 Diagramas de Clases

El diagrama de clases forma parte de la vista estática del sistema. Son utilizados durante el proceso de Análisis y Diseño de los sistemas informáticos, donde se crea el diseño conceptual de la información que se manejará en el sistema, y de los componentes que se encargarán del funcionamiento y la relación entre uno y otro. El diagrama de clases permite definir las características de cada una de las clases, interfaces, colaboraciones y relaciones de dependencia y generalización, es donde podemos mostrar el diseño de los objetos, definiendo las clases e implementando las ya típicas relaciones de herencia y agregación. Un buen modelo de clases debe ser fácil de mantener y adaptar a futuros requisitos. En el modelo de clases, las clases están compuestas de atributos y métodos, y las estructuras estáticas son los objetos.

3.6.1 Diagrama de Clases General

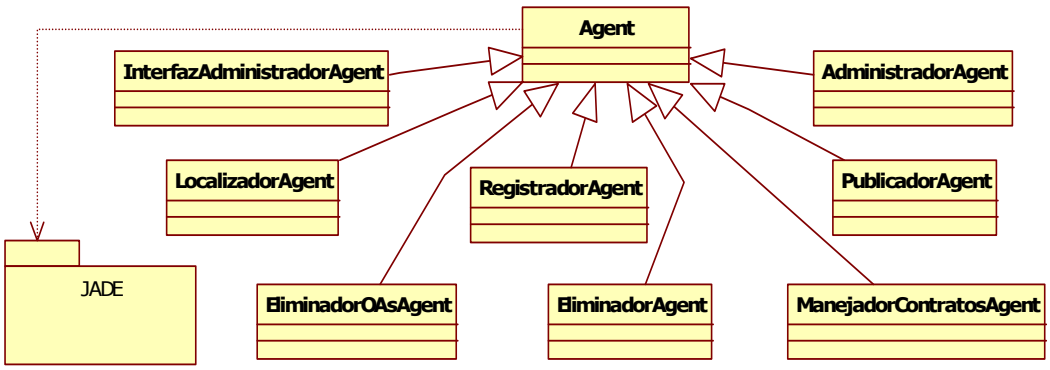

Figura 3.5 Diagrama de Clases. Sistema Multi-Agente

El diagrama de clases mostrado en la figura 3.5 representa a nuestro sistema multi-agente, cada clase personifica a un agente contenido en el mercado de objetos de aprendizaje, y a su vez, cada uno de ellos como bien puede apreciarse hereda de la clase Agent, la cual nos permite crear agentes con determinados tipos de comportamientos, según sea el requerimiento del sistema multi-agente, así también, se puede observar el uso del paquete JADE, plataforma que nos permitirá simplificar el desarrollo del sistema.

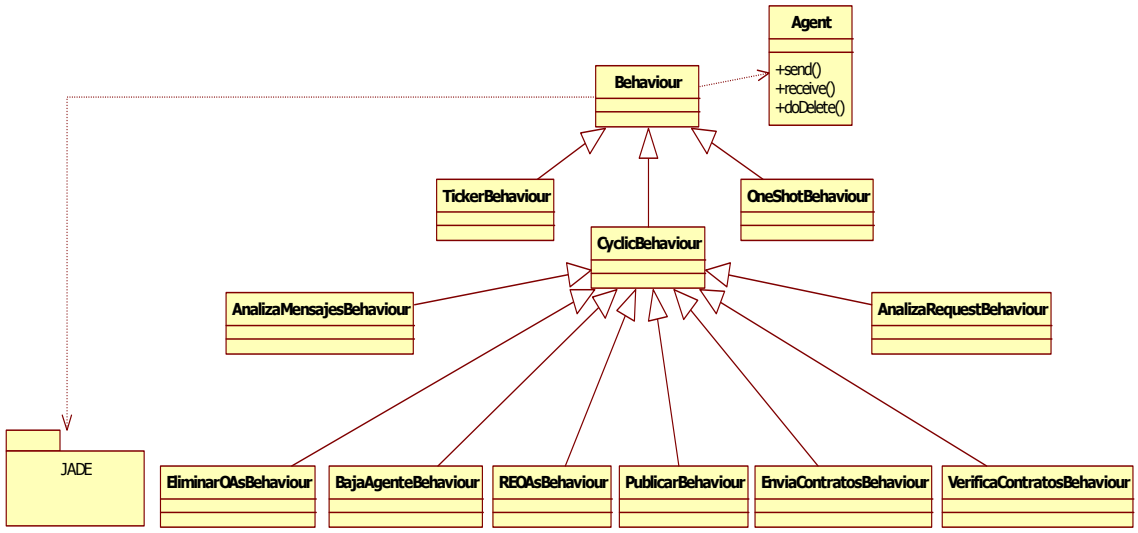

Figura 3.6 Diagrama de Clases. Comportamientos del Sistema Multi-Agente

Los comportamientos que dan vida al sistema multi-agente heredan de los comportamientos: CyclicBehaviour, TickerBehaviour y OneShotBehaviour. Por su parte estos comportamientos a su vez heredan de la clase padre Behaviour. Los comportamientos de nuestro sistema básicamente son un gestor de eventos, es decir, la forma de como un agente reacciona a un evento por la recepción de un mensaje, ver figura 3.6.

Capítulo 4

Modelo de Diseño

4.1 Diagrama de Clases Detallado

Conforme se van encontrando los objetos en el sistema, estos pueden ser agrupados por tipos y clasificados en un diagrama de clases. El diagrama de clases refinado identifica todos los métodos y atributos de cada clase, a partir de ello, mostraremos los diagramas de clases refinados del mercado de objetos de aprendizaje.

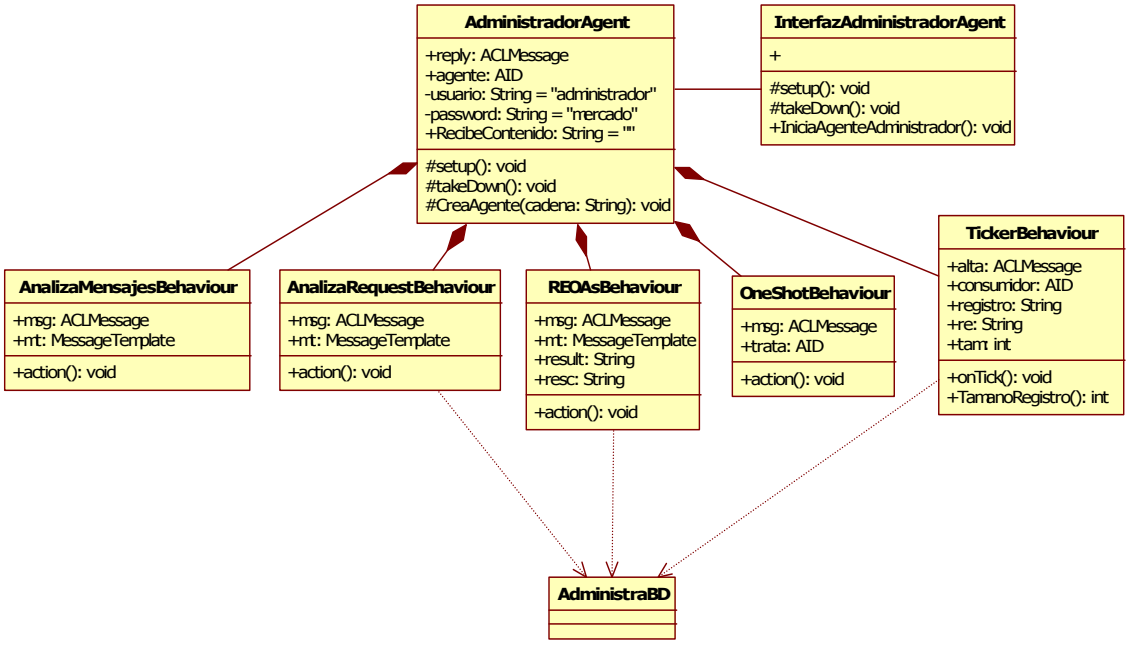

Figura 4.1 Diagrama de Clases. Agente Administrador

El diagrama de clases mostrado en la figura 4.1 modela los objetos que forman parte de la creación del agente administrador con sus respectivos comportamientos. Por tanto, la finalidad del diagrama de clases del Agente Administrador es proveer servicios a todos

aquellos agentes externos al MOA para su registro, publicación, baja, registro de contratos, envío de contratos, etc. Para llevar a cabo lo anterior, la clase AdministradorAgent es la encargada de crear, de acuerdo a la solicitud recibida, el agente que atenderá esa petición. Por otro lado, hablando de la parte interna del mercado, la clase AdministradorAgent también atenderá las peticiones de todos los agentes internos que conforman el sistema multi-agente, para interactuar con la base de datos local a través de la clase AdministraBD.

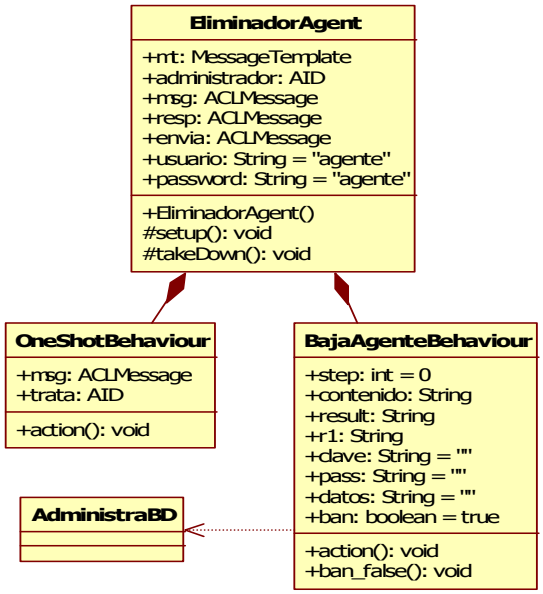

Figura 4.2 Diagrama de Clases. Agente Eliminador de Agentes

El diagrama de clases Agente Eliminador de Agentes, mostrado en la figura 4.2, permite ofrecer la baja de un agente consumidor y de un agente proveedor de objetos de aprendizaje en el mercado, cuando estos envíen su correspondiente solicitud.

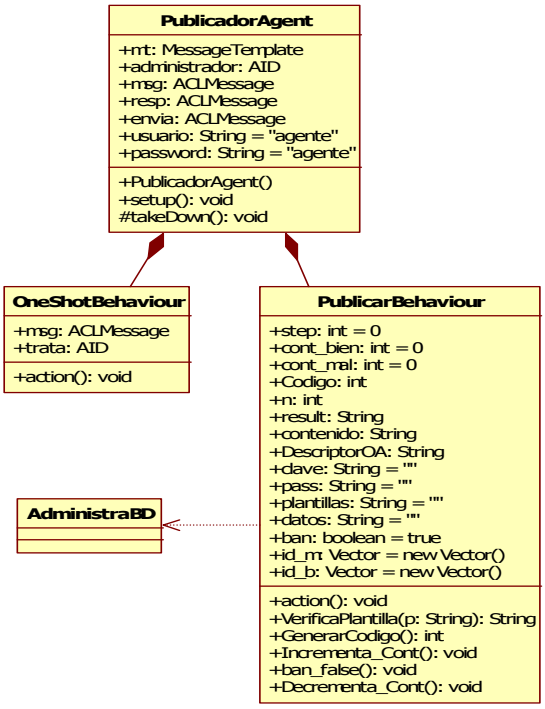

Figura 4.3 Diagrama de Clases. Agente Publicador de OAs

El diagrama de clases del Agente Publicador de OAs, mostrado en la figura 4.3, permite ofrecer el servicio de publicación de un OA en el mercado de objetos de aprendizaje, cuando se reciba una solicitud de publicación de un determinado OA por parte de un agente proveedor.

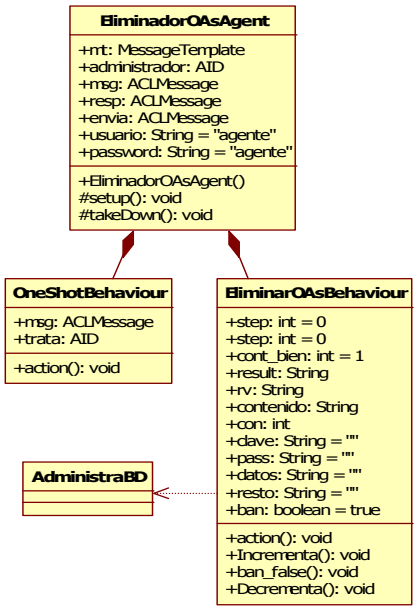

Figura 4.4 Diagrama de Clases. Agente Eliminador de OAs

La figura 4.4 ilustra el diagrama de clases del Agente Eliminador de OAs, este provee el servicio de eliminar un objeto de aprendizaje del mercado cuando recibe una solicitud de baja de OA por parte de algún agente proveedor.

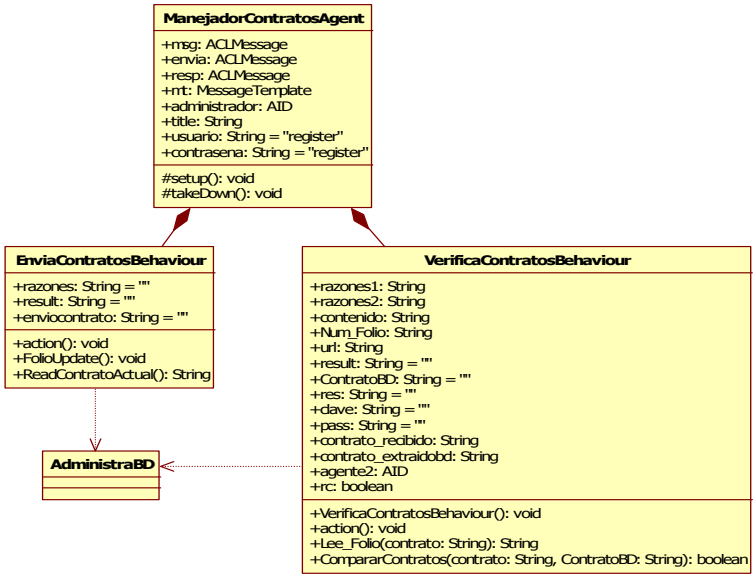

Figura 4.5 Diagrama de Clases. Agente Manejador de Contratos

El diagrama de clases mostrado en la figura 4.5 permite ofrecer el servicio de envío de contrato a un agente proveedor y el servicio de registro de contratos enviados por un agente consumidor y un agente proveedor, para que estos sean almacenados en la base de datos local del MOA.

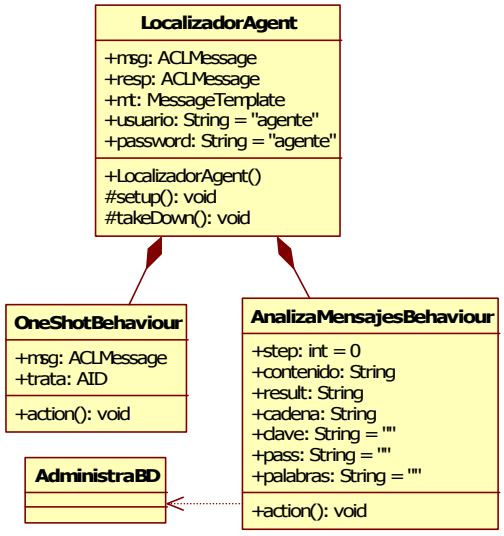

Figura 4.6 Diagrama de Clases. Agente Localizador de Proveedores de OAs

El diagrama de clases del Agente Localizador de Proveedores de OAs, mostrado en la figura 4.6, permite ofrecer el servicio de localización de proveedores de OA, a un agente consumidor, cuando este solicite dicho servicio.

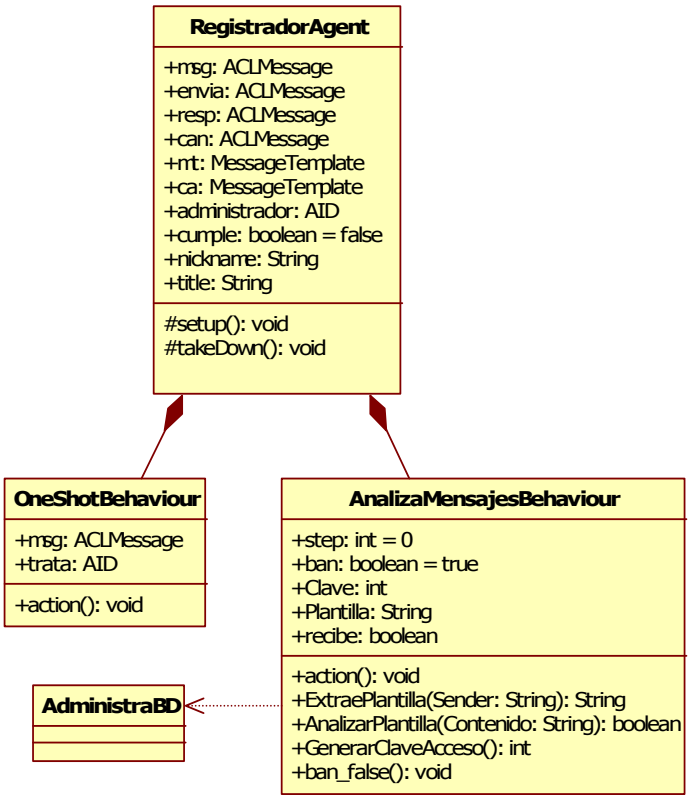

Figura 4.7 Diagrama de Clases. Agente Registro

El diagrama de objetos mostrado en la figura 4.7, permite ofrecer el servicio de registro tanto a un agente proveedor como a un agente consumidor en el mercado de objetos de aprendizaje, para que posteriormente estos agentes puedan ingresar al MOA, solicitando algún tipo de servicio requerido para satisfacer sus necesidades.

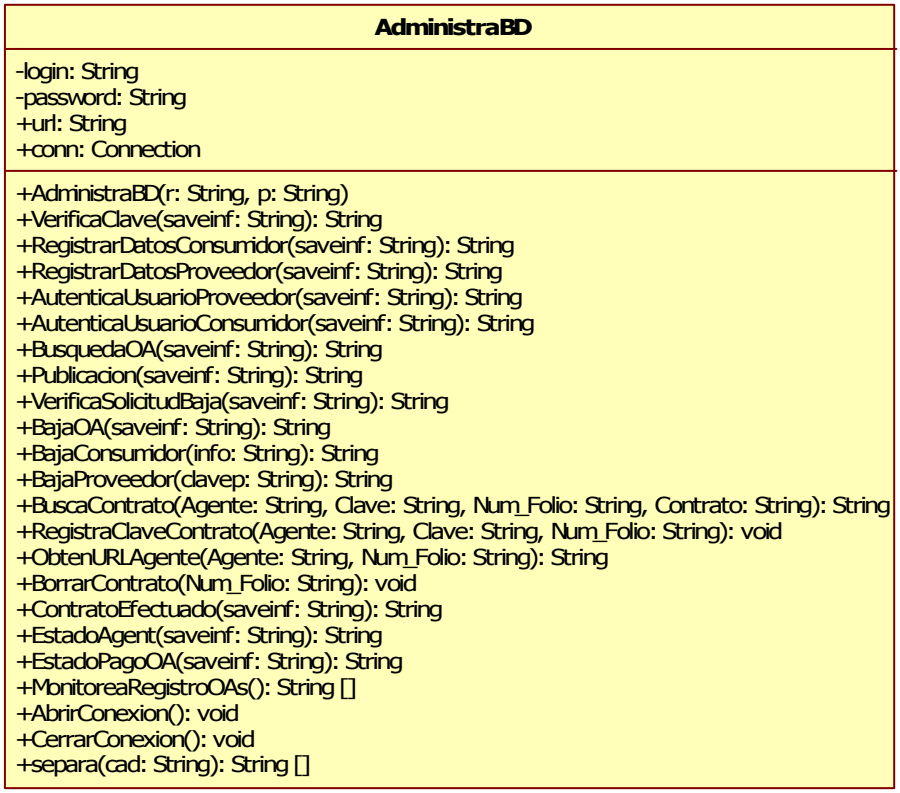

Figura 4.8 Diagrama de Clases. Administración Base de Datos

La figura 4.8 muestra el diagrama de clases Administración Base de Datos, que utiliza el sistema multi-agente del mercado de objetos de aprendizaje para consultar, guardar, borrar y extraer información, de acuerdo al servicio que se este proporcionando en un determinado momento.

4.2 Diagramas de Secuencia

Muestran las interacciones que constan de un conjunto de objetos arregladas en una secuencia de tiempo. El diagrama muestra a los objetos participando en la interacción, así como la secuencia de los mensajes intercambiados por estos.

4.2.1 Diagrama de Secuencia: Creación del Mercado de Objetos de Aprendizaje

En esta sección se muestra el diagrama de secuencia requerido, para la creación del mercado de objetos de aprendizaje.

Creación del Agente Administrador

En la figura 4.9 se muestra la creación del agente administrador a través del objeto InterfazAdministradorAgent.

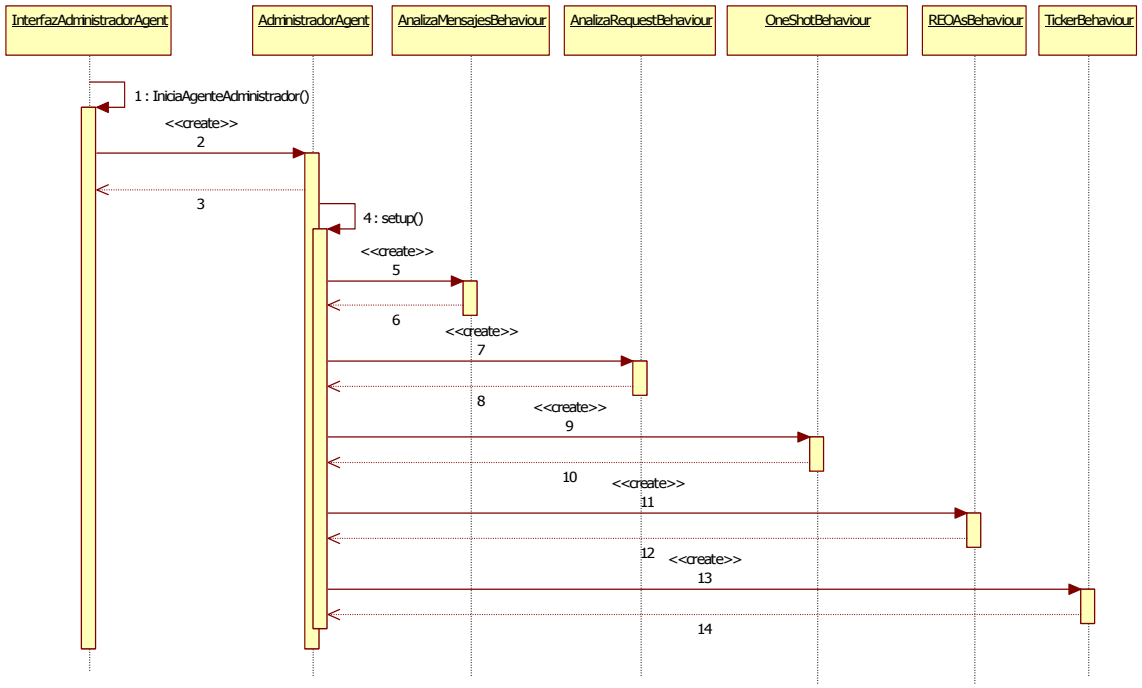

Figura 4.9 Diagrama de Secuencia. Creación del Agente Administrador

Descripción:

1. La operación IniciaAgenteAdministrador del objeto InterfazAdministradorAgent se ejecuta en primera instancia.
2. La operación IniciaAgenteAdministrador del objeto InterfazAdministradorAgent crea al objeto AdministradorAgent.
3. El objeto AdministradorAgent devuelve el control al objeto InterfazAdministradorAgent.

4. El AdministradorAgent ejecuta su setup, para el lanzamiento de sus comportamientos.
5. El AdministradorAgent crea el objeto AnalizaMensajesBehaviour a través de la operación setup, este objeto tendrá la función de recibir aquellas solicitudes enviadas por agentes externos al mercado de objetos de aprendizaje, para ser atendidas.
6. El objeto AnalizaMensajesBehaviour devuelve el control al objeto AdministradorAgent.
7. El AdministradorAgent crea el objeto AnalizaRequestBehaviour a través de la operación setup, que recibirá aquellas solicitudes enviadas por agentes internos del mercado de objetos de aprendizaje, para ser atendidas.
8. El objeto AnalizaRequestBehaviour devuelve el control al objeto AdministradorAgent.
9. El AdministradorAgent crea el objeto OneShotBehaviour a través de la operación setup, este objeto tendrá la función de crear a nuestro agente manejador de contratos.
10. El objeto OneShotBehaviour devuelve el control al objeto AdministradorAgent.
11. El AdministradorAgent crea el objeto REOAsBehaviour a través de la operación setup, este objeto se encargara del envío y recepción de los OA negociados por parte de los agentes consumidores y proveedores.
12. El objeto REOAsBehaviour devuelve el control al objeto AdministradorAgent.
13. El AdministradorAgent crea el objeto TickerBehaviour a través de la operación setup, el cual tendrá la tarea de monitorear cada media hora la base local del MOA y hacer un recuento de cuantos proveedores han enviado sus OA para ser entregados a los agentes consumidores.
14. El objeto TickerBehaviour devuelve el control al objeto AdministradorAgent.

4.2.2 Diagramas de Secuencia: Registro

En los siguientes diagramas de secuencia se modela el servicio de registro de un agente consumidor y de un agente proveedor en el mercado de objetos de aprendizaje.

Parte 1: Creación del Agente de Registro

En la figura 4.10 se puede observar la creación del agente de registro por parte del agente administrador a través del comportamiento AnalizaMensajesBehaviour.

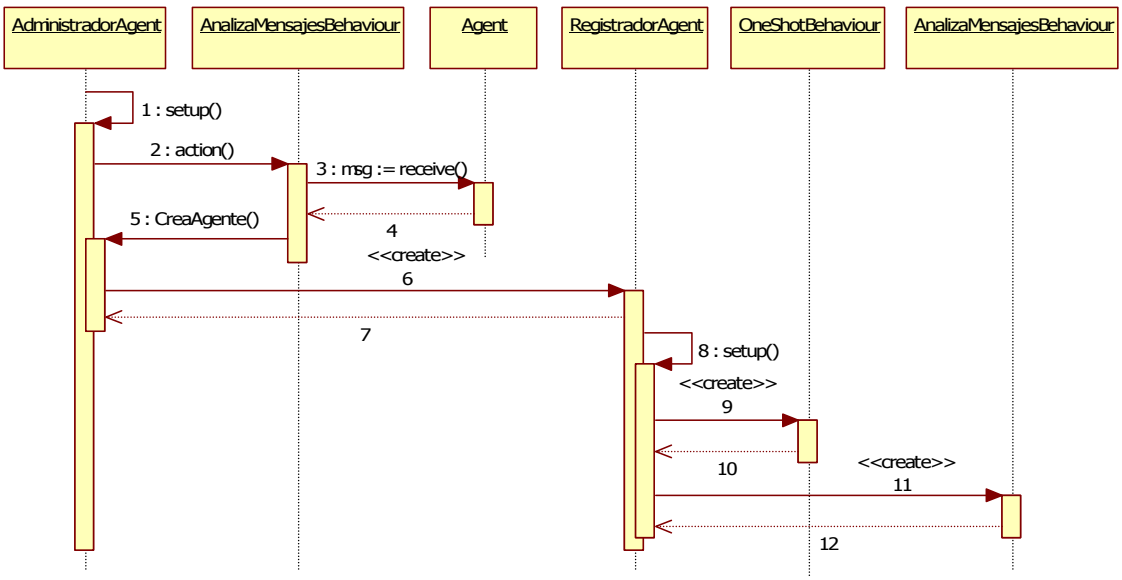

Figura 4.10 Diagrama de Secuencia. Creación del Agente de Registro

Descripción:

1. El AdministradorAgent se inicializa a través del método setup.
2. El AdministradorAgent ejecuta la operación action del objeto AnalizaMensajesBehaviour, método donde se reciben todas las peticiones de los agentes externos al MOA.
3. El objeto AnalizaMensajesBehaviour invoca la operación receive del objeto Agent, esta operación recibirá la petición de solicitud registro por parte de un agente consumidor o un agente proveedor.

4. El objeto Agent retorna el control al objeto AnalizaMensajesBehaviour.
5. El objeto AnalizaMensajesBehaviour invoca la operación CreaAgente contenida en el método setup del objeto AdministradorAgent.
6. El objeto AdministradorAgent a través del setup crea al objeto RegistradorAgent, este objeto es el que se encargara de realizar el registro correspondiente de un determinado agente (consumidor/proveedor).
7. El objeto RegistradorAgent regresa el control al objeto AdministradorAgent.
8. El objeto RegistradorAgent ejecuta su setup.
9. El objeto RegistradorAgent crea al objeto OneShotBehaviour.
10. OneShotBehaviour regresa el control al objeto RegistradorAgent.
11. RegistradorAgent crea al objeto AnalizaMensajesBehaviour.
12. AnalizaMensajesBehaviour retorna el control al objeto RegistradorAgent.

Parte 2: Servicio Registro

En la figura 4.11 se muestran las interacciones que ocurren cuando el agente de registro recibe una solicitud de registro por parte de un agente proveedor o consumidor.

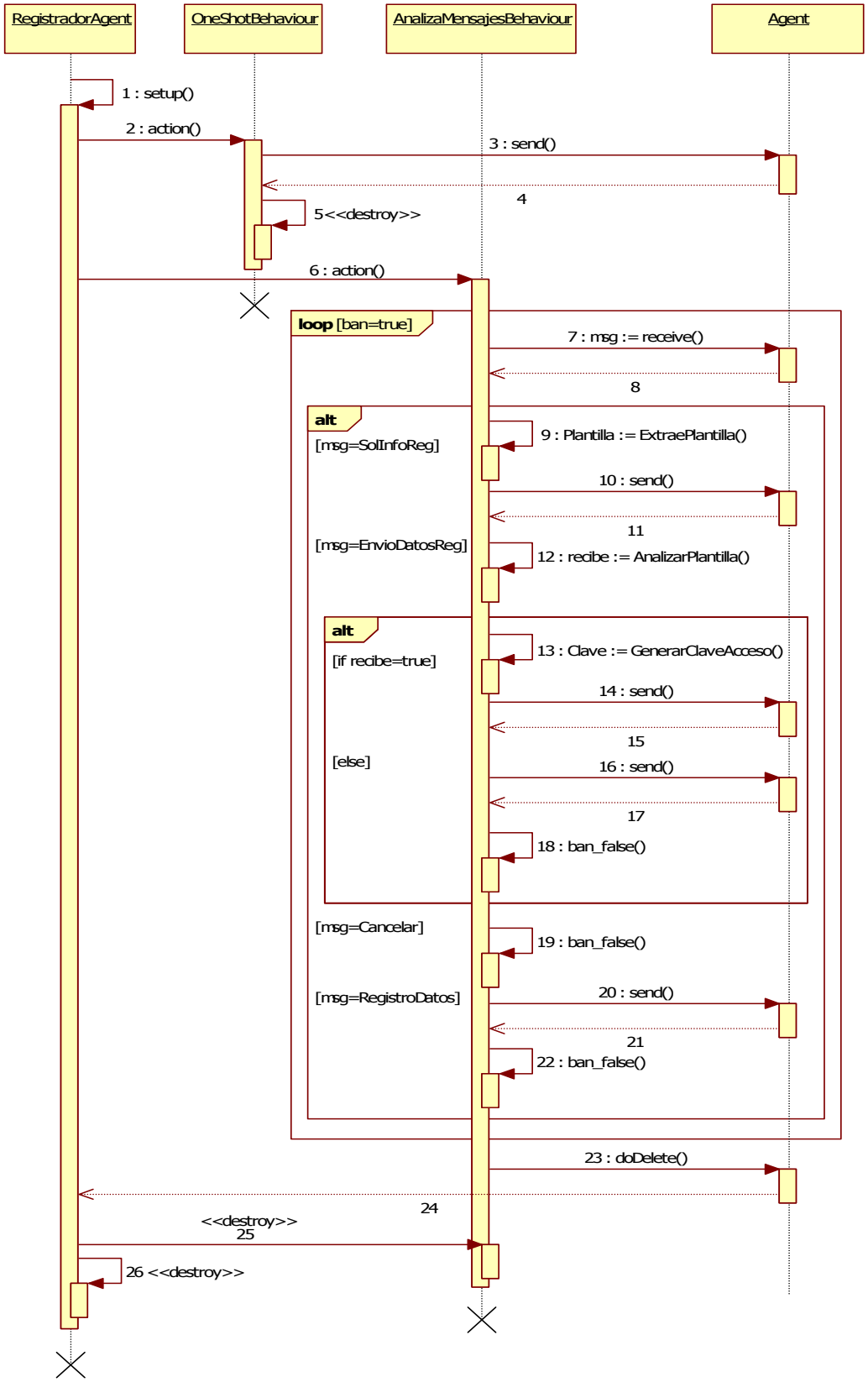

Figura 4.11 Diagrama de Secuencia. Servicio Registro

Descripción:

1. El RegistradorAgent ejecuta el método setup.
2. La clase RegistradorAgent invoca la función action del objeto OneShotBehaviour.
3. La clase OneShotBehaviour ejecuta la operación send del objeto Agent, para enviar un mensaje al agente solicitante, sobre quien le atenderá durante el proceso de registro en el MOA.
4. El objeto Agent devuelve el control al objeto OneShotBehaviour.
5. Se destruye el objeto OneShotBehaviour.
6. El objeto RegistradorAgent ejecuta la operación action del objeto OneShotBehaviour.

Ciclo: Mientras ban = true

7. La clase AnalizaMensajesBehaviour invoca la operación receive del objeto Agent, con esta función se recibirán solicitudes por parte del agente consumidor o proveedor y los resultados de las operaciones efectuadas en la base de datos local por parte del agente administrador.
8. El objeto Agent retorna el control a la clase AnalizaMensajesBehaviour.

Alternativa: Si msg= SolInfoReg

9. La clase AnalizaMensajesBehaviour extrae la plantilla y adjunta folio.
10. El objeto AnalizaMensajesBehaviour ejecuta la función send del objeto Agent, para enviar la plantilla de registro al agente correspondiente.
11. El objeto Agent retorna el control al objeto AnalizaMensajesBehaviour.

Alternativa: Si msg=EnvioDatosReg

12. La clase AnalizaMensajesBehaviour analiza la plantilla recibida.

Alternativa: Si recibe=true

13. La clase AnalizaMensajesBehaviour ejecuta la operación GenerarClaveAcceso.
14. La clase AnalizaMensajesBehaviour ejecuta la función send del objeto Agent para enviar al agente administrador los datos extraídos de la plantilla más la clave de acceso generada, para ser almacenados en la base de datos local del MOA.

15. El objeto Agent retorna el control a la clase AnalizaMensajesBehaviour.

Alternativa: Si recibe=false

16. La clase AnalizaMensajesBehaviour ejecuta la operación send del objeto Agent, a través del cual enviará un mensaje de solicitud rechazada al agente solicitante.
17. La clase Agent retorna el control al objeto AnalizaMensajesBehaviour.
18. La clase AnalizaMensajesBehaviour ejecuta el método ban_false.

Alternativa: Si msg=Cancelar

19. El objeto AnalizaMensajesBehaviour ejecuta la operación ban_false.

Alternativa: Si msg=RegistroDatos

20. El objeto AnalizaMensajesBehaviour ejecuta la función send del objeto Agent para enviar al agente solicitante su clave de acceso o una solicitud de rechazo.
21. El objeto Agent retorna el control al objeto AnalizaMensajesBehaviour.
22. El objeto AnalizaMensajesBehaviour ejecuta el método ban_false.
23. El objeto AnalizaMensajesBehaviour ejecuta la función doDelete del objeto Agent.
24. El objeto Agent retorna el control a la clase RegistradorAgent.
25. El RegistradorAgent destruye a la clase AnalizaMensajesBehaviour.
26. El objeto RegistradorAgent se autodestruye.

Parte 3: Almacenamiento de Datos del Registro

En la figura 4.12 se observan las interacciones que ocurren cuando el agente administrador recibe una solicitud de registro de datos, de un agente consumidor o de un agente proveedor por parte del agente de registro.

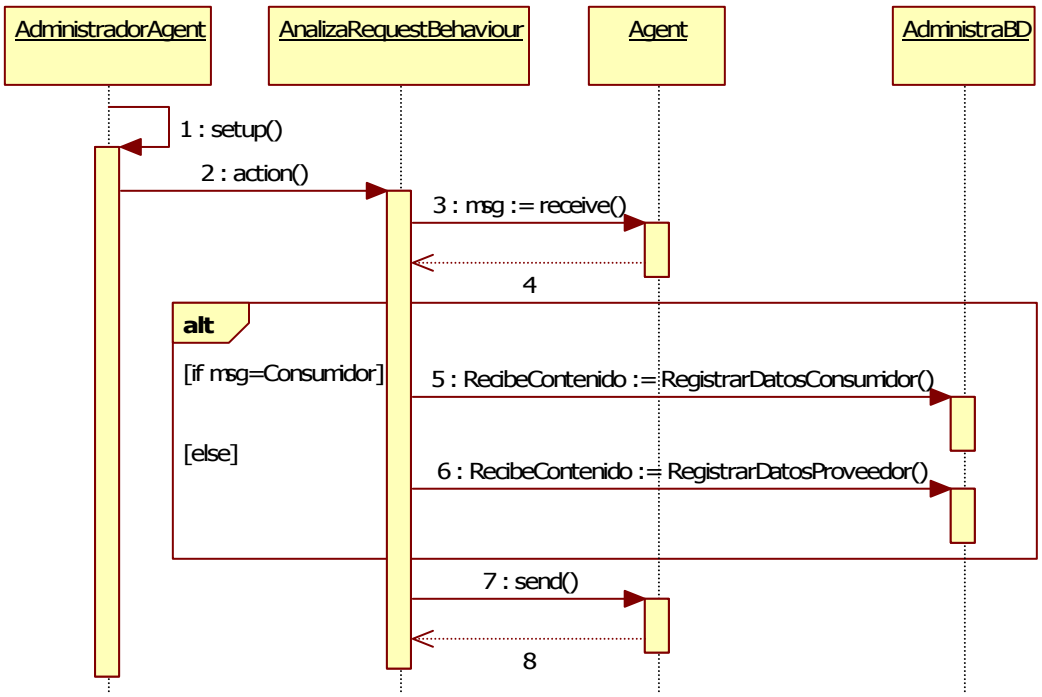

Figura 4.12 Diagrama de Secuencia. Almacenamiento de Datos del Registro

Descripción:

1. El AdministradorAgent ejecuta el método setup.
2. La clase AdministradorAgent invoca la función action del objeto AnalizaRequestBehaviour.
3. La clase AnalizaRequestBehaviour ejecuta la operación receive del objeto Agent, para recibir un mensaje de solicitud registrar datos por parte del agente registrador.
4. La clase Agent retorna el control al objeto AnalizaRequestBehaviour.

Alternativa: Si msg=Consumidor

5. La clase AnalizaRequestBehaviour invoca el método RegistrarDatosConsumidor del objeto AdministraBD.

Alternativa: else

6. La clase AnalizaRequestBehaviour invoca el método RegistrarDatosProveedor del objeto AdministraBD.
7. La clase AnalizaRequestBehavior ejecuta el método send de la clase Agent, para enviar al agente de registro el resultado del almacenamiento de los datos.
8. El objeto Agent regresa el control al objeto AnalizaRequestBehavior.

4.2.3 Diagramas de Secuencia: Publicación

En los siguientes diagramas de secuencia se modela el servicio de solicitud publicación de OA por parte de un agente proveedor en el mercado de objetos de aprendizaje.

Parte 1: Creación del Agente Publicador

En la figura 4.13 se muestra la creación del agente publicador por parte del agente administrador a través de su comportamiento AnalizaMensajesBehaviour.

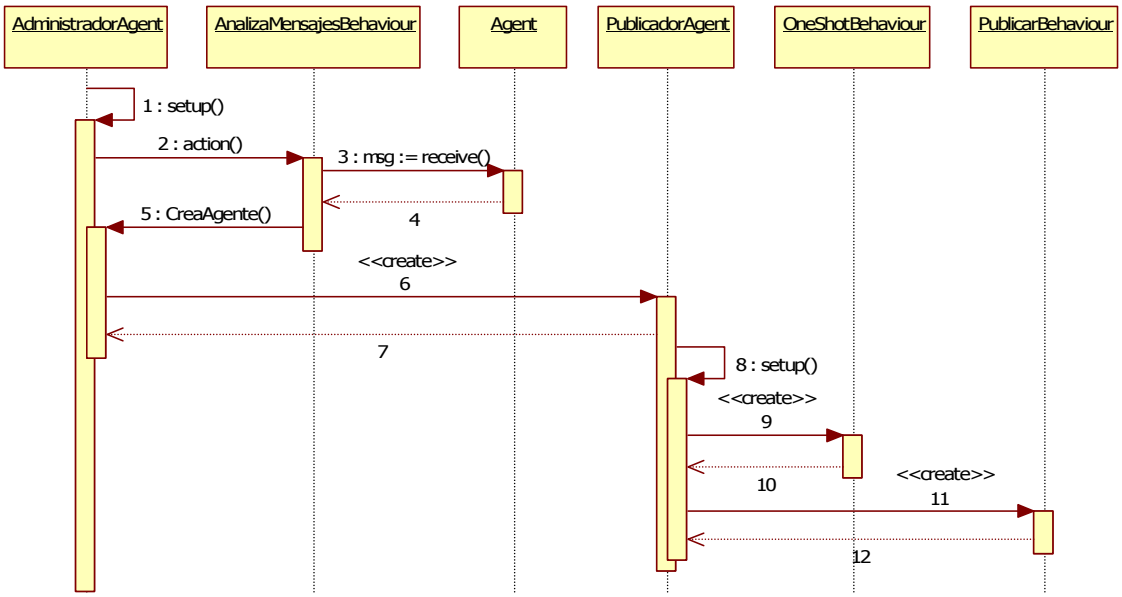

Figura 4.13 Diagrama de Secuencia. Creación del Agente Publicador

Descripción:

1. El AdministradorAgent ejecuta la operación setup.
2. El AdministradorAgent invoca la operación action del objeto AnalizaMensajesBehaviour, para recibir las peticiones de los agentes externos al MOA.
3. El objeto AnalizaMensajesBehaviour ejecuta la función receive del objeto Agent, esta función recibirá las peticiones de solicitud publicación por parte de un agente proveedor.
4. El objeto Agent retorna el control al objeto AnalizaMensajesBehaviour.
5. El objeto AnalizaMensajesBehaviour ejecuta la operación CreaAgente del objeto AdministradorAgent.
6. El objeto AdministradorAgent a través del setup crea al objeto PublicadorAgent, este objeto se encargara de realizar la publicación de OA en el mercado, cuando se reciba una solicitud de publicación de un agente proveedor.
7. La clase PublicadorAgent regresa el control al objeto AdministradorAgent.
8. La clase PublicadorAgent ejecuta su setup.
9. La clase PublicadorAgent crea al objeto OneShotBehaviour.
10. OneShotBehaviour regresa el control al objeto PublicadorAgent.
11. La clase PublicadorAgent crea al objeto PublicarBehaviour.
12. PublicarBehaviour retorna el control al objeto PublicadorAgent.

Parte 2: Servicio Publicación

En la figura 4.14 se pueden observar las interacciones que ocurren cuando el agente publicador recibe una solicitud de publicación por parte de un agente proveedor para ofertar sus OA.

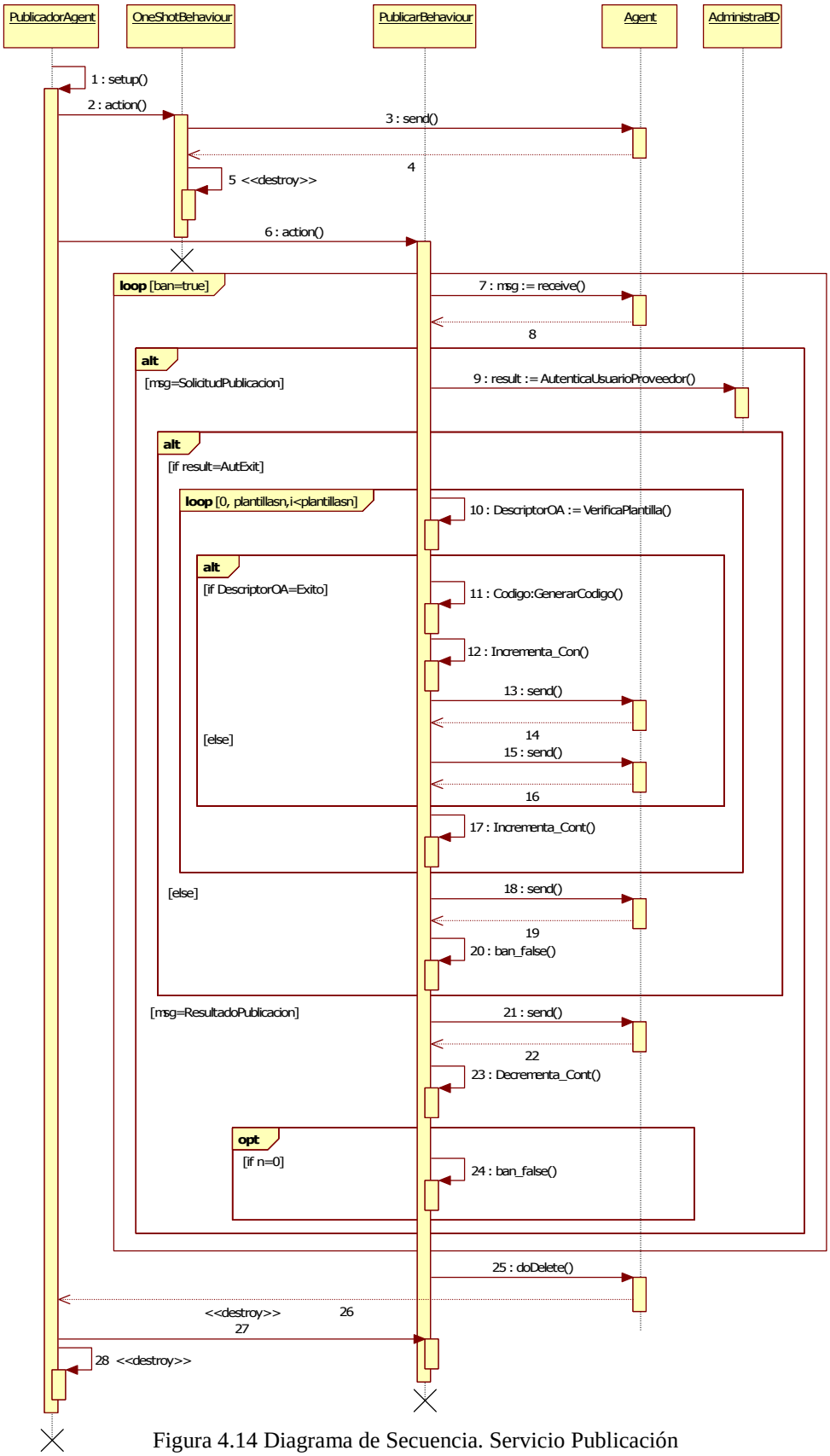

Figura 4.14 Diagrama de Secuencia. Servicio Publicación

Descripción:

1. El objeto PublicadorAgent se inicializa.
2. El objeto PublicadorAgent ejecuta el método action del objeto OneShotBehaviour.
3. El objeto OneShotBehaviour ejecuta la operación send del objeto Agent, para enviar un mensaje al agente proveedor, sobre quien le atenderá durante el proceso de publicación.
4. El objeto Agent regresa el control al objeto OneShotBehaviour.
5. El objeto OneshotBehaviour se autodestruye.
6. La clase PublicadorAgent ejecuta el método action de la clase PublicarBehaviour.

Ciclo: Mientras ban=true

7. La clase PublicarBehaviour ejecuta la operación receive del objeto Agent, esta operación recibirá las solicitudes de publicación por parte de un agente proveedor y el resultado de publicación por parte del agente administrador.
8. El objeto Agent retorna el control al objeto PublicarBehaviour.

Alternativa: Si msg=SolicitudPublicacion

9. El objeto PublicarBehaviour ejecuta la operación AutenticaUsuarioProveedor de la clase AdministraBD.

Alternativa: Si result=AutExit

Ciclo: Desde i=0, mientras i<plantillasn

10. El objeto PublicarBehaviour verifica la plantilla de publicación.

Alternativa: DescriptorOA=Exito

11. PublicarBehaviour genera código de OA.
12. PublicarBehaviour ejecuta la operación Incrementa_Con.
13. PublicarBehaviour ejecuta la operación send del objeto Agent, para enviar los datos de publicación a almacenar al agente administrador.
14. El objeto Agent devuelve el control a la clase PublicarBehaviour.

Alternativa: else

15. PublicarBehaviour ejecuta la operación send del objeto Agent, para enviar un mensaje al agente proveedor de verificación plantilla no exitosa.
16. Agent retorna el control al objeto PublicarBehaviour.

17. La clase PublicarBehaviour ejecuta la función Incrementa_Con.

Alternativa: else

18. El objeto PublicarBehaviour ejecuta la operación send del objeto mAgent, donde se enviara el mensaje fallo de autenticación al agente proveedor.
19. El objeto Agent retorna el control al objeto PublicarBehaviour.
20. PublicarBehaviour ejecuta la operación ban_false.

Alternativa: Si msg=ResultadoPublicacion

21. El objeto PublicarBehaviour ejecuta send del objeto Agent, para el envío del resultado de la publicación al agente proveedor.
22. El objeto Agent devuelve el control al objeto PublicarBehaviour.
23. PublicarBehaviour ejecuta la operación Decrementa_Cont.

Opción: Si n= 0

24. El objeto PublicarBehaviour ejecuta la operación ban_false.

25. La clase PublicarBehaviour ejecuta el método doDelete del objeto Agent.
26. EL objeto Agent devuelve el control al objeto PublicadorAgent.
27. El objeto PublicadorAgent destruye al objeto PublicarBehaviour.
28. El PublicadorAgent se autodestruye.

Parte 3: Almacenamiento de Datos de la Publicación

En la figura 4.15 se muestran las interacciones para realizar el registro de publicación del OA en la base de datos, solicitado por el agente publicador.

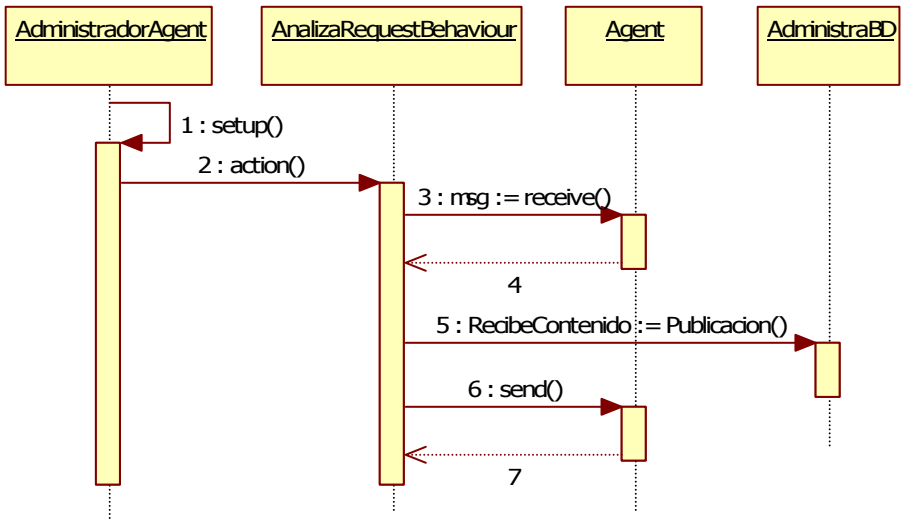

Figura 4.15 Diagrama de Secuencia. Almacenamiento de Datos de la Publicación

Descripción:

1. La clase AdministradorAgent se inicializa.
2. La clase AdministradorAgent ejecuta el método action del objeto AnalizaRequestBehaviour.
3. La clase AnalizaRequestBehaviour ejecuta la operación receive del objeto Agent, para recibir los datos del OA que será publicado en la base de datos local del MOA (ClaveP, Código del OA, Título del OA, Identificador del OA de la BD local del proveedor y la Plantilla).
4. El objeto Agent devuelve el control al objeto AnalizaRequestBehaviour.
5. El objeto AnalizaRequestBehaviour ejecuta la operación Publicación de la clase AdministraBD, para guardar los datos de publicación del OA.
6. El objeto AnalizaRequestBehaviour ejecuta un send del objeto Agent para enviar el resultado de la publicación al agente publicador.
7. El objeto Agent retorna el control al objeto AnalizaRequestBehaviour.

4.2.4 Diagramas de Secuencia: Localización de Proveedores de OA

Los siguientes diagramas de interacción modelan el servicio de localización de proveedores, que publican sus OA en el mercado.

Parte 1: Creación del Agente Localizador

En la figura 4.16 se muestra la creación del agente localizador de proveedores de OA por parte del agente administrador, cuando este recibe una solicitud a través del comportamiento AnalizaMensajesBehaviour.

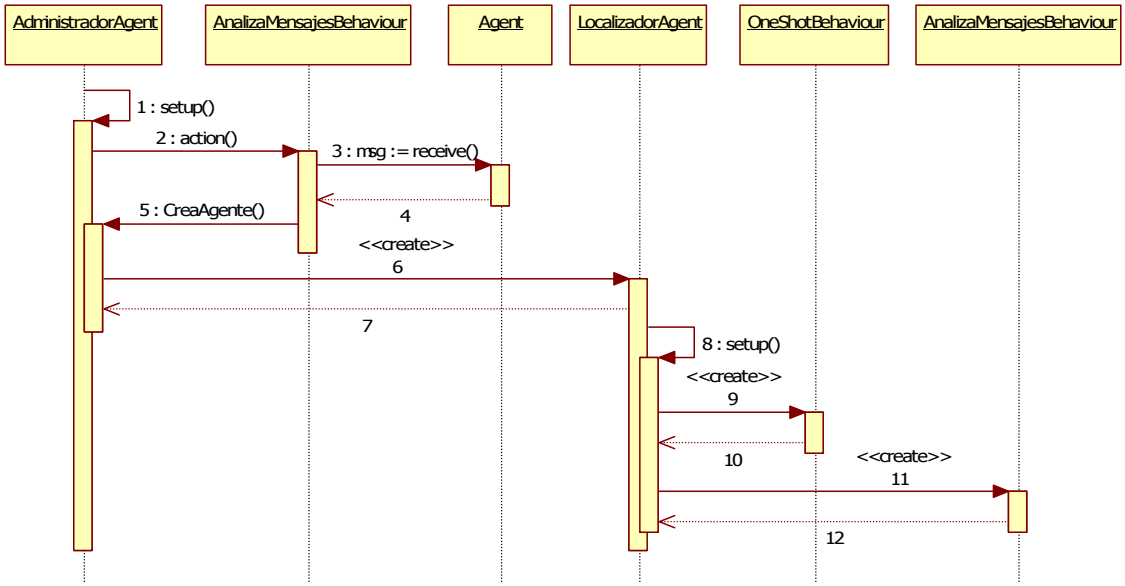

Figura 4.16 Diagrama de Secuencia. Creación del Agente Localizador

Descripción:

1. La clase AdministradorAgent se inicializa.
2. La clase AdministradorAgent invoca la operación action de la clase AnalizaMensajesBehaviour.
3. La clase AnalizaMensajesBehaviour ejecuta la operación receive de la clase Agent, donde recibirá la solicitud localizar OA por parte del agente consumidor.
4. La clase Agent retorna el control a la clase AnalizaMensajesBehaviour.

5. La clase AnalizaMensajesBehaviour ejecuta la operación CreaAgente de la clase AdministradorAgent.
6. La clase AdministradorAgent crea la clase LocalizadorAgent, quien atenderá todo el proceso de localización de proveedores de OA, en el mercado.
7. La clase LocalizadorAgent retorna el control a la clase AdministradorAgent.
8. La clase LocalizadorAgent se inicializa.
9. La clase LocalizadorAgent crea a la clase OneShotBehaviour.
10. La clase OneShotBehaviour retorna el control a la clase LocalizadorAgent.
11. La clase LocalizadorAgent crea la clase AnalizaMensajesBehaviour.
12. La clase AnalizaMensajesBehaviour retorna el control a la clase LocalizadorAgent.

Parte 2: Servicio Localización de Proveedores de OA

En la figura 4.17 se modelan las interacciones que ocurren cuando el agente localizador recibe un mensaje de solicitud de localización de proveedores de OA por parte de un agente consumidor.

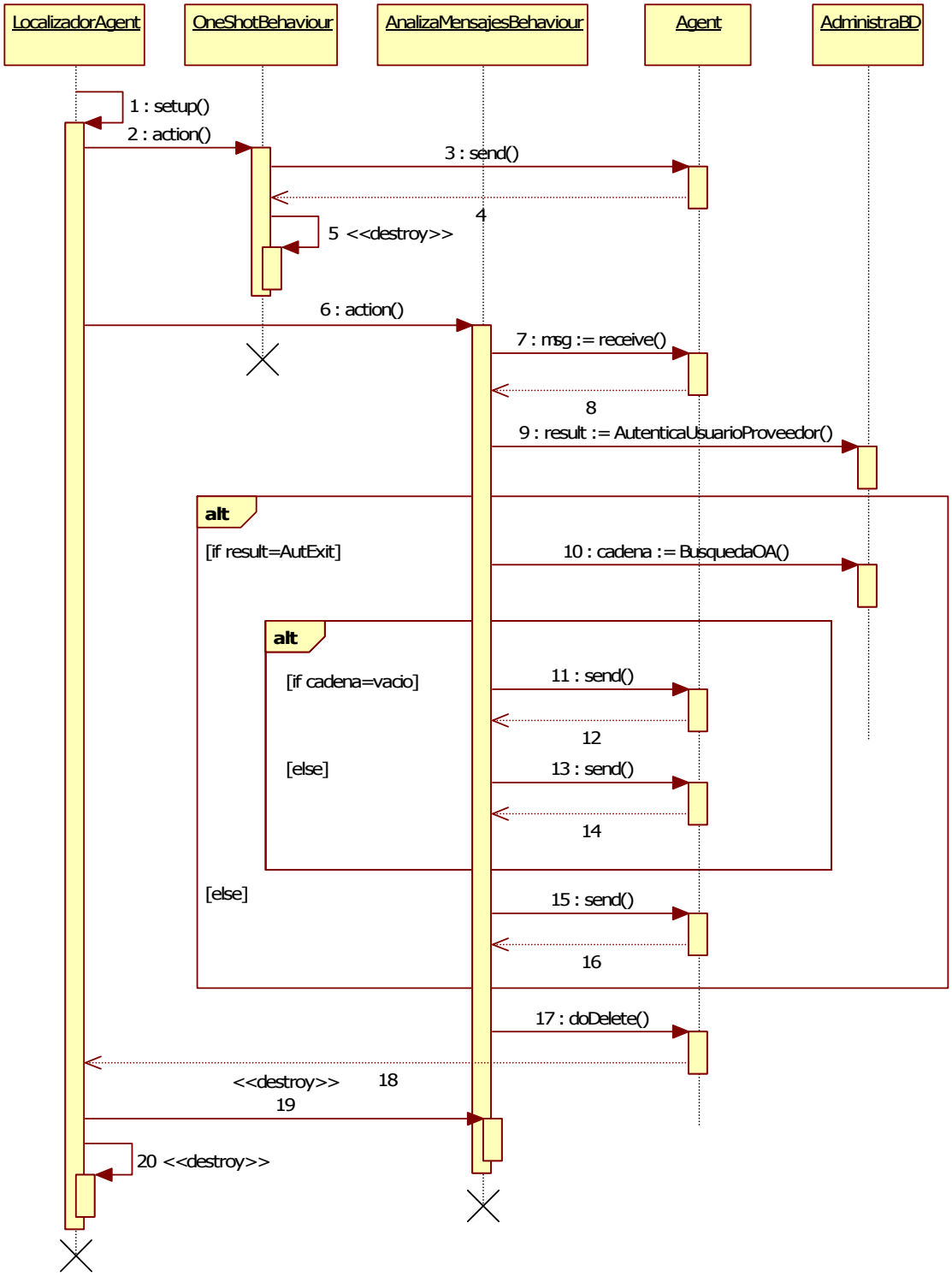

Figura 4.17 Diagrama de Secuencia. Servicio Localización de Proveedores de OA

Descripción:

1. El objeto LocalizadorAgent se inicializa.
2. El objeto LocalizadorAgent ejecuta la función action del objeto OneShotBehaviour.
3. El objeto OneShotBehaviour invoca la función send del objeto Agent, donde enviara un mensaje al agente consumidor, para infórmale quien le atenderá durante el proceso de localización de proveedores de OA.
4. El objeto Agent retorna el control a la clase OneShotBehaviour.
5. El objeto OneShotBehaviour se autodestruye.
6. El objeto LocalizadorAgent ejecuta el método action del objeto AnalizaMensajesBehaviour.
7. El objeto AnalizaMensajesBehaviour ejecuta la función receive del objeto Agent, donde recibirá un mensaje de solicitud de búsqueda de proveedores de OA por parte de un agente consumidor.
8. El objeto Agent regresa el control al objeto AnalizaMensajesBehaviour.
9. El objeto AnalizaMensajesBehaviour ejecuta la operación AutenticaUsuarioProveedor del objeto AdministraBD.

Alternativa: Si result=AutExit

10. El objeto AnalizaMensajesBehaviour ejecuta la función BusquedaOA del objeto AdministraBD.

Alternativa: Si cadena=vacio

11. El objeto AnalizaMensajesBehaviour ejecuta la función send del objeto Agent, para enviar un mensaje al agente consumidor de proveedor de OA no encontrado.
12. El objeto Agent regresa el control al objeto AnalizaMensajesBehaviour.

Alternativa: else

13. El objeto AnalizaMensajesBehaviour ejecuta la función send del objeto Agent, para enviar la lista de los proveedores de OA encontrados en el MOA.
14. El objeto Agent regresa el control al objeto AnalizaMensajesBehaviour.

Alternativa: else

15. El objeto AnalizaMensajesBehaviour ejecuta la operación send del objeto Agent, donde envía un mensaje al agente consumidor de autenticación no exitosa.
16. El objeto Agent regresa el control al objeto AnalizaMensajesBehaviour.
17. AnalizaMensajesBehaviour ejecuta doDelete de la clase Agent.
18. El objeto Agent regresa el control al objeto LocalizadorAgent.
19. El objeto LocalizadorAgent destruye al objeto AnalizaMensajesBehaviour.
20. El objeto LocalizadorAgent se autodestruye.

4.2.5 Diagramas de Secuencia: Manejo de Contratos

En esta sección se muestran los diagramas que modelan la creación del agente manejador de contratos y aquellos que modelan las interacciones que permiten atender solicitudes tales como: envío de un contrato vacío a un agente proveedor, y el servicio de registro contrato, tanto de un agente consumidor como de un agente proveedor.

Parte 1: Envío Información

En la figura 4.18 se modela el servicio de envío de información a un agente externo solicitante, sobre quien lo atenderá, para que esté pueda solicitar el envío de un contrato vacío o el registro de un contrato efectuado en la base de datos local del mercado.

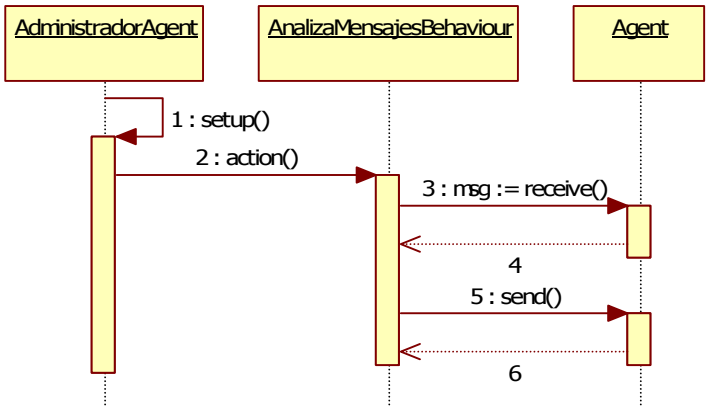

Figura 4.18 Diagrama de Secuencia. Envío Información

Descripción:

1. La clase AdministradorAgent ejecuta su setup.
2. La clase AdministradorAgent invoca el método action de la clase AnalizaMensajesBehaviour.
3. La clase AnalizaMensajesBehaviour ejecuta la función receive de la clase Agent, donde se recibirá la petición de solicitud contrato o registro de contrato por parte de algún agente.
4. La clase Agent retorna el control a la clase AnalizaMensajesBehaviour.
5. La clase AnalizaMensajesBehaviour ejecuta la operación send del objeto Agent, para enviar la información correspondiente al agente externo solicitante.
6. La clase Agent retorna el control a la clase AnalizaMensajesBehaviour.

Parte 2: Creación del Agente Manejador de Contratos

En la figura 4.19 se muestra la creación del agente manejador de contratos, a través del comportamiento OneShotBehaviour, mismo que forma parte del agente administrador.

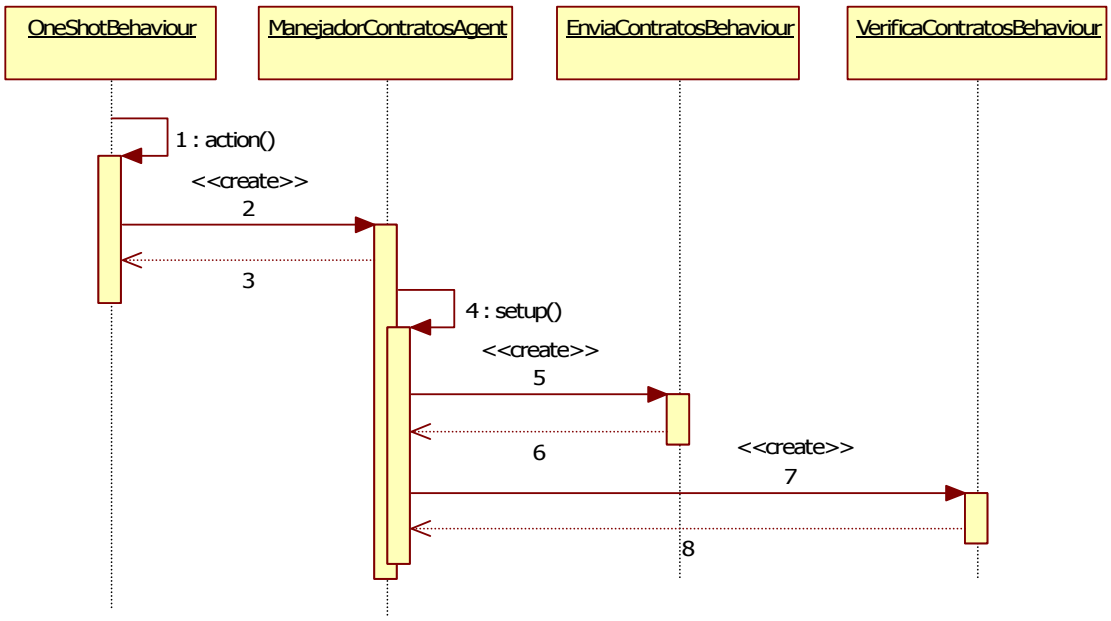

Figura 4.19 Diagrama de Secuencia. Creación del Agente Manejador de Contratos

Descripción:

1. El objeto OneShotBehaviour ejecuta su action.
2. El objeto OneShotBehaviour crea al objeto ManejadorContratosAgent.
3. El objeto ManejadorContratosAgent devuelve el control al objeto OneShotBehaviour.
4. El objeto ManejadorContratosAgent se inicializa a través del setup.
5. El objeto ManejadorContratosAgent crea al objeto EnviaContratosBehaviour.
6. El objeto EnviaContratosBehaviour devuelve el control al objeto ManejadorContratosAgent.
7. El objeto ManejadorContratosAgent crea al objeto VerificaContratosBehaviour.
8. El objeto VerificaContratosBehaviour devuelve el control al objeto ManejadorContratosAgent.

Parte 3: Servicio Envío Contrato

El diagrama mostrado en la figura 4.20 modela la serie de mensajes intercambiados entre las clases, para realizar el envío de contrato solicitado por un agente proveedor.

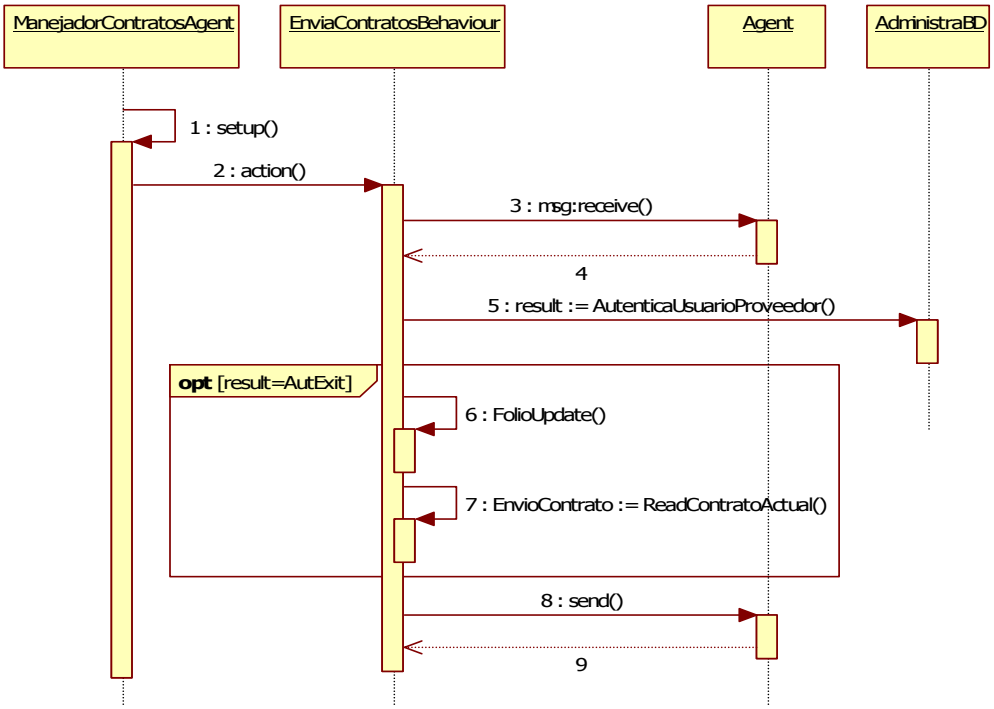

Figura 4.20 Diagrama de Secuencia. Envío Contrato.

Descripción:

1. La clase ManejadorContratosAgent se inicializa.
2. La clase ManejadorContratosAgent ejecuta la operación action de la clase EnviaContratosBehaviour.
3. La clase EnviaContratosBehaviour ejecuta la operación receive de la clase Agent, para recibir las peticiones de envío de contrato por parte de los agentes proveedores.
4. La clase Agent devuelve el control a la clase EnviaContratosBehaviour.
5. La clase EnviaContratosBehaviour ejecuta la operación AutenticaUsuarioProveedor de la clase AdministraBD.

Opción: Si result=AutExit

6. La clase EnviaContratosBehaviour actualiza los datos del contrato a ser enviado.
7. La clase EnviaContratosBehaviour lee el contrato actual.

8. La clase EnviaContratosBehaviour ejecuta la operación send de la clase Agent, para enviar al agente solicitante la plantilla de contrato vacía o un mensaje de autenticación no exitosa.
9. La clase Agent devuelve el control a la clase EnviaContratosBehaviour.

Parte 4: Almacenamiento de Contrato

La serie de mensajes intercambiados entre las clases del sistema, al realizar el registro del contrato en la base de datos, se muestran en la figura 4.21.

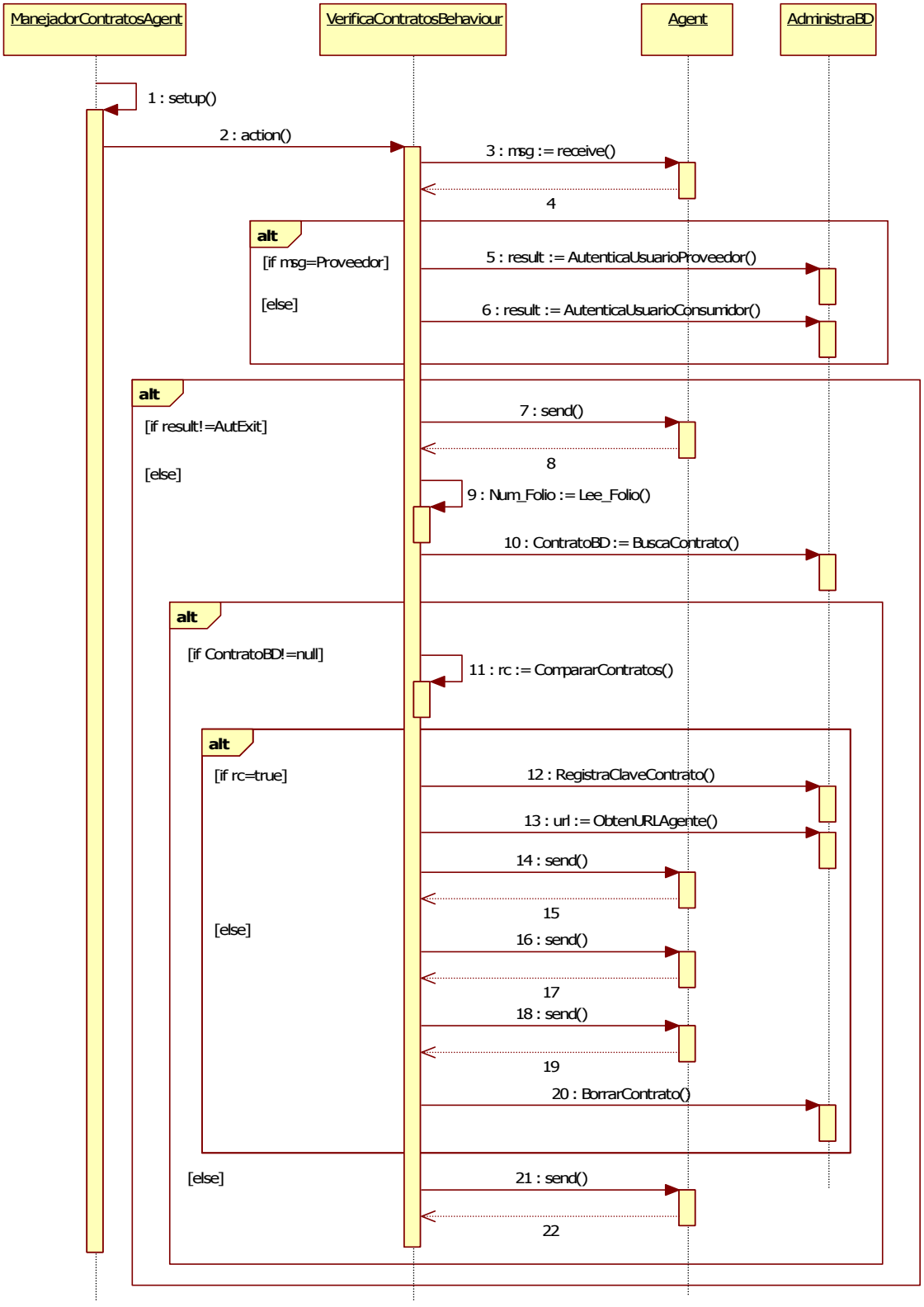

Figura 4.21 Diagrama de Secuencia. Almacenamiento de Contrato

Descripción:

1. El objeto ManejadorContratosAgent ejecuta el setup.
2. El objeto ManejadorContratosAgent ejecuta el método action del objeto VerificaContratosBehaviour.
3. El objeto VerificaContratosBehaviour ejecuta el método receive del objeto Agent, para recibir las solicitudes de registro de contrato de un agente consumidor o de un agente proveedor.
4. El objeto Agent retorna el control al objeto VerificaContratosBehaviour.

Alternativa: Si msg=Proveedor

5. El objeto VerificaContratosBehaviour invoca la operación AutenticaUsuarioProveedor del objeto AdministraBD.

Alternativa: else

6. El objeto VerificaContratosBehaviour invoca la operación AutenticaUsuarioConsumidor del objeto AdministraBD.

Alternativa: Si result!=AutExit

7. El objeto VerificaContratosBehaviour invoca la operación send del objeto Agent, para enviar un mensaje de autenticación no exitosa.
8. El objeto Agent devuelve el control al objeto VerificaContratosBehaviour.

Alternativa: else

9. El objeto VerificaContratosBehaviour lee el folio de la plantilla.
10. EL objeto VerificaContratosBehaviour ejecuta el método BuscaContrato del objeto AdministraBD, con la finalidad de buscar si existe ya un contrato almacenado en la base de datos con el folio del contrato que acaba de recibir, si no existe, almacena el contrato nuevo que acaba de recibir.

Alternativa: Si ContratoBD!=null

11. El objeto VerificaContratosBehaviour compara el contrato que acaba de recibir con el contrato almacenado en la base de datos, a través del método CompararContratos.

Alternativa: Si rc=true

12. El objeto VerificaContratosBehaviour ejecuta el método RegistraClaveContrato del objeto AdministraBD, para almacenar la clave faltante ya sea del agente consumidor o del agente proveedor.
13. El objeto VerificaContratosBehaviour ejecuta el método ObtenURLAgente del objeto AdministraBD, para obtener la dirección url del agente que se anticipó al registro del contrato.
14. El objeto VerificaContratosBehaviour ejecuta la operación send del objeto Agent, para enviar el mensaje de guardado de contrato exitoso a los agentes que solicitaron el servicio de registro.
15. El objeto Agent retorna el control al objeto VerificaContratosBehaviour.

Alternativa: else

16. El objeto VerificaContratosBehaviour ejecuta la operación send del objeto Agent, para enviar contratoBD al último agente en pedir la solicitud de registro de contrato.
17. El objeto Agent devuelve el control al objeto VerificaContratosBehaviour.
18. El objeto VerificaContratosBehaviour ejecuta la operación send del objeto Agent, para enviar contrato al primer agente en pedir solicitud registro de contrato.
19. El objeto Agent devuelve el control al objeto VerificaContratosBehaviour.
20. El VerificaContratosBehaviour ejecuta el método BorrarContrato del objeto AdminsitraBD.

Alternativa: else

21. El objeto VerificaContratosBehaviour ejecuta la operación send del objeto Agent, a través del cual enviará un mensaje de guardado de contrato exitoso.
22. El objeto Agent retorna el control al objeto.

4.2.6 Diagramas de Secuencia: Baja de OAs

En los modelados siguientes, se mostrará la forma en como interaccionan las diferentes clases para poder ofrecer el servicio de baja de un objeto de aprendizaje solicitado por un agente proveedor.

Parte 1: Creación del Agente Eliminador de OAs

En la figura 4.22 se muestra la creación del agente eliminador de OAs, a través de la función setup del agente administrador.

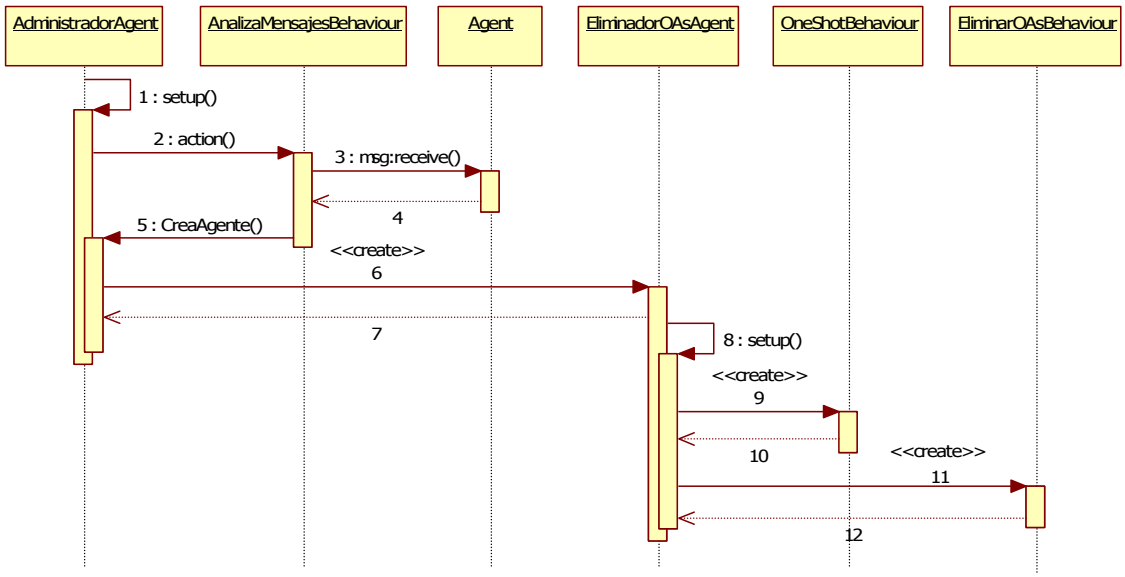

Figura 4.22 Diagrama de Secuencia. Creación del Agente Eliminador de OAs

Descripción:

1. El objeto AdministradorAgent ejecuta su setup.
2. El objeto AdministradorAgent ejecuta la función action de la clase AnalizaMensajesBehaviour.
3. El objeto AnalizaMensajesBehaviour ejecuta la función receive del objeto Agent, para recibir la solicitud baja de un OA.
4. El objeto Agent retorna el control al objeto AnalizaMensajesBehaviour.

5. El objeto AnalizaMensajesBehaviour ejecuta la función CreaAgente del objeto AdministradorAgent.
6. El objeto AdministradorAgent crea el objeto EliminadorOAsAgent, quien atenderá las peticiones de baja de un OA solicitadas por un agente proveedor.
7. El objeto EliminadorOAsAgent retorna el control al objeto AdministradorAgent.
8. El objeto EliminadorOAsAgent se inicializa.
9. El objeto EliminadorOAsAgent crea el objeto OneShotBehaviour.
10. El objeto OneShotBehaviour retorna el control al objeto EliminadorOAsAgent.
11. El objeto EliminadorOAsAgent crea al objeto EliminarOAsBehaviour.
12. El objeto EliminarOAsBehaviour retorna el control al objeto EliminadorOAsAgent.

Parte 2: Servicio de Eliminación de OAs

El diagrama de secuencia que provee el servicio de eliminación de OA, puede ser visualizado en la figura 4.23, para que se lleve a cabo la baja de un OA, la clase EliminadorOAsAgent debe verificar a través de su comportamiento EliminarOAsBehaviour que el agente proveedor no tenga adeudos en el MOA, en caso de estar todo en perfectas condiciones se procederá a dar de baja el OA solicitado.

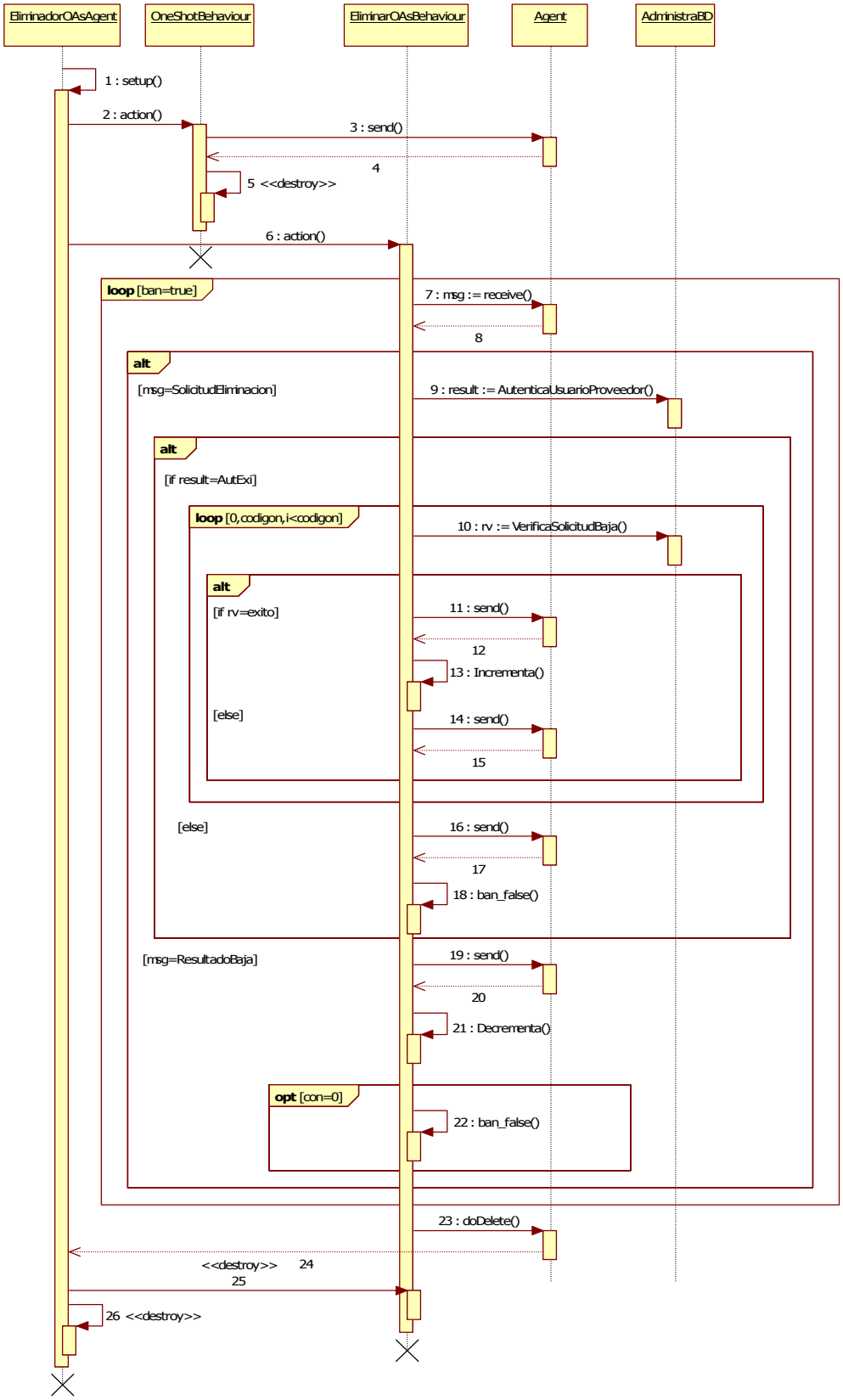

Figura 4.23 Diagrama de Secuencia. Servicio de Eliminación de OAs

Descripción:

1. La clase EliminadorOAsAgent ejecuta el setup.
2. La clase EliminadorOAsAgent ejecuta el método action de la clase OneShotBehaviour.
3. La clase OneShotBehaviour ejecuta un send de la clase Agent, par enviar al agente que requiere el servicio, la información de quien lo atenderá.
4. La clase Agent devuelve el control a la clase OneShotBehaviour.
5. La clase OneShotBehaviour se autodestruye.
6. La clase EliminadorOAsAgent ejecuta el método action de la clase EliminarOAsBehaviour.

Ciclo: Mientras ban=true

7. La clase EliminarOAsBehaviour ejecuta un receive de la clase Agent, para recibir la petición de un agente proveedor de baja de OA, o la respuesta del agente administrador, acerca del resultado de la eliminación de un determinado OA de la base de datos local.
8. La clase Agent devuelve el control a la clase EliminarOAsBehaviour.

Alternativa: Si msg=SolicitudEliminacion

9. La clase EliminarOAsBehaviour ejecuta el método AutenticaUsuarioProveedor de la clase AdministraBD.

Alternativa: Si result=AutExit

Ciclo: Desde i=0, mientras i<codigon

10. La clase EliminarOAsBehaviour ejecuta el método VerificaSolicitudBaja de la clase AdministraBD.

Alternativa: Si rv=éxito

11. La clase EliminarOAsBehaviour ejecuta un send de la clase Agent, para enviar al agente administrador una petición de baja del OA.
12. La clase Agent devuelve el control a la clase EliminarOAsBehaviour.
13. La clase EliminarOAsBehaviour ejecuta el método Incrementa.

Alternativa: else

14. La clase EliminarOAsBehaviour ejecuta un send de la clase Agent, para enviar al agente proveedor un mensaje de rechazo de baja.
15. La clase Agent devuelve el control a la clase EliminarOAsBehaviour.

Alternativa: else

16. La clase EliminarOAsBehaviour ejecuta un send de la clase Agent, para enviar al agente proveedor un mensaje de rechazo ingreso.
17. La clase Agent devuelve el control a la clase EliminarOAsBehaviour.
18. La clase EliminarOAsBehaviour ejecuta la opracion ban_false.

Alternativa: Si msg=ResultadoBaja

19. La clase EliminarOAsBehaviour ejecuta un send de la clase Agent, para enviar al agente proveedor un mensaje del resultado de la baja de un OA.
20. La clase Agent devuelve el control a la clase EliminarOAsBehaviour.
21. La clase EliminarOAsBehaviour ejecuta la operación Decrementa.

Opción: Si con= 0

22. La clase EliminarOAsBehaviour ejecuta la operación ban_false.

23. La clase EliminarOAsBehaviour ejecuta el método doDelete de la clase Agent.
24. La clase Agent devuelve el control a la clase EliminadorOAsAgent.
25. La clase EliminadorOAsAgent destruye la clase EliminarOAsBehaviour.
26. La clase EliminadorOAsAgent se autodestruye.

Parte 3: Baja de OAs de la Base de Datos

El diagrama de secuencia mostrado en la figura 4.24 modela las interacciones necesarias para llevar a cabo la baja de un OA de la base de datos local del mercado de objetos de aprendizaje.

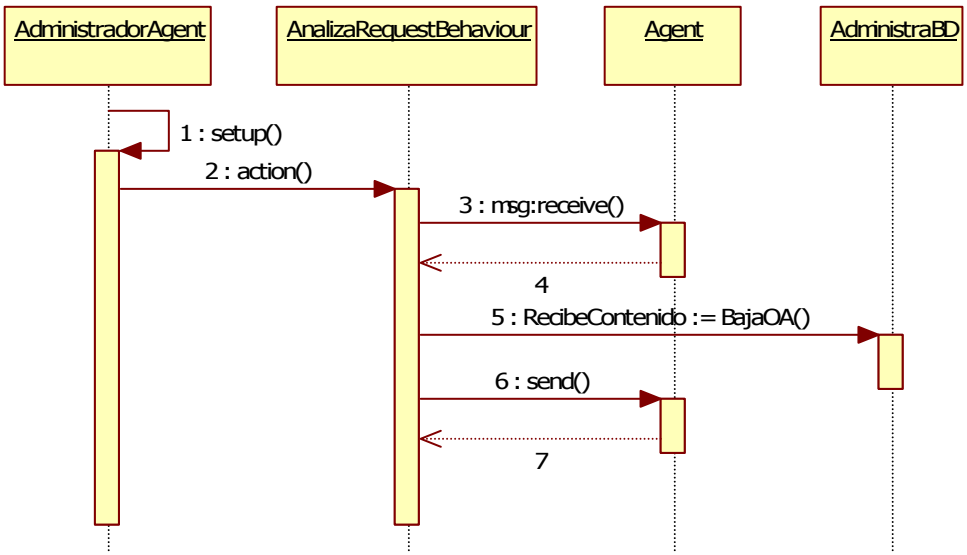

Figura 4.24 Diagrama de Secuencia. Baja de OAs de la Base de Datos

Descripción:

1. El objeto AdministradorAgent ejecuta el método setup.
2. El objeto AdministradorAgent ejecuta la función action del objeto AnalizaRequestBehaviour.
3. El objeto AnalizaRequestBehaviour ejecuta la operación receive del objeto Agent, esta operación recibe un mensaje de solicitud baja de OA, por parte de un agente eliminador de OAs.
4. El objeto Agent devuelve el control al objeto AnalizaRequestBehaviour.
5. El objeto AnalizaRequestBehaviour ejecuta el método BajaOA del objeto AdministraBD.
6. El objeto AnalizaRequestBehaviour ejecuta la operación send del objeto Agent, para enviar al agente eliminador de OAs, el resultado de la baja.
7. El objeto Agent devuelve el control al objeto AnalizaRequestBehaviour.

4.2.7 Diagramas de Secuencia: Baja Agente

Para llevar a cabo el servicio de baja de un agente, ya sea consumidor o proveedor, del mercado de objetos de aprendizaje, es necesario modelarlo en tres pasos fundamentales, el primero y más importante, el modelado para la creación del agente eliminador de agentes, quien es el que proporcionará el servicio de baja solicitado, el segundo y sin ser el menos significativo; el servicio de baja de agente, donde se muestra el modelado de todas las interacciones de comunicación para atender dichas solicitudes de baja, y por último el modelado de baja del agente en la base de datos local del mercado.

Parte 1: Creación del Agente Eliminador de Agentes

El modelado mostrado en la figura 4.25 muestra la creación del agente eliminador de agentes.

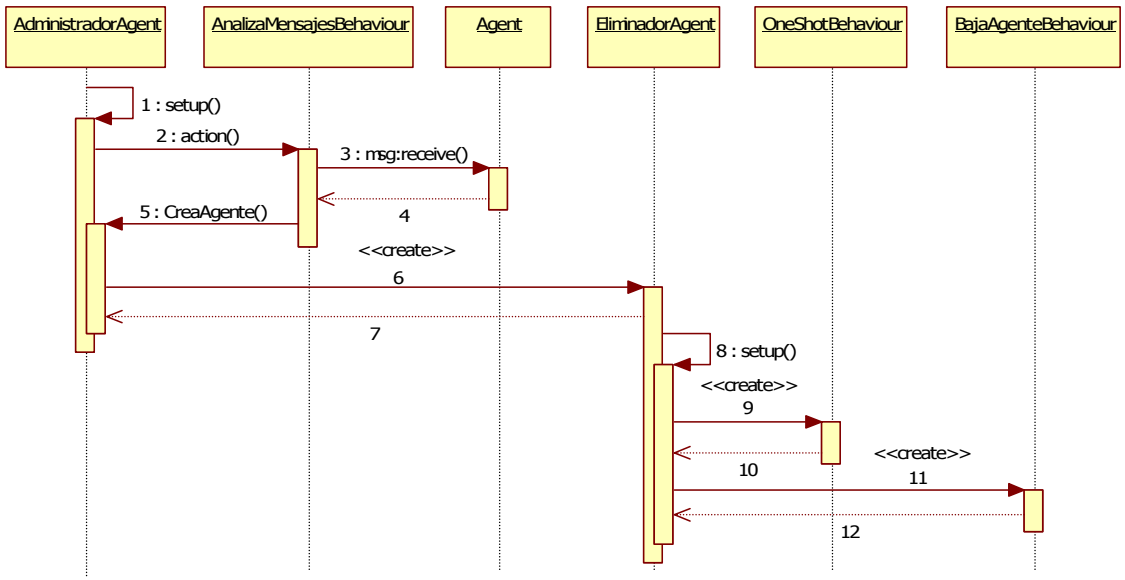

Figura 4.25 Diagrama de Secuencia. Creación del Agente Eliminador de Agentes

Descripción:

1. El objeto AdministradorAgent se inicializa a través del setup.
2. El AdministradorAgent invoca el método action del objeto AnalizaMensajesBehaviour.
3. El objeto AnalizaMensajesBehaviour invoca el método receive del objeto Agent, donde se recibirá la solicitud de un agente proveedor o consumidor para solicitar su baja del mercado.
4. El objeto Agent retorna el control al objeto AnalizaMensajesBehaviour.
5. AnalizaMensajesBehaviour invoca el método CreaAgente contenido en el setup del objeto AdministradorAgent.
6. El AdministradorAgent crea al objeto EliminadorAgent.
7. El objeto EliminadorAgent devuelve el control al objeto AdministradorAgent.
8. El EliminadorAgent se inicializa a través del setup.
9. El EliminadorAgent crea al objeto OneshotBehaviour.
10. El objeto OneShotBehaviour devuelve el control al objeto EliminadorAgent.
11. El EliminadorAgent crea al objeto BajaAgenteBehaviour.
12. El objeto BajaAgenteBehaviour devuelve el control al objeto EliminadorAgent.

Parte 2: Servicio Baja Agente

La figura 4.26 muestra los mensajes entre las clases que permitirán proporcionar el servicio de baja de un agente en el mercado.

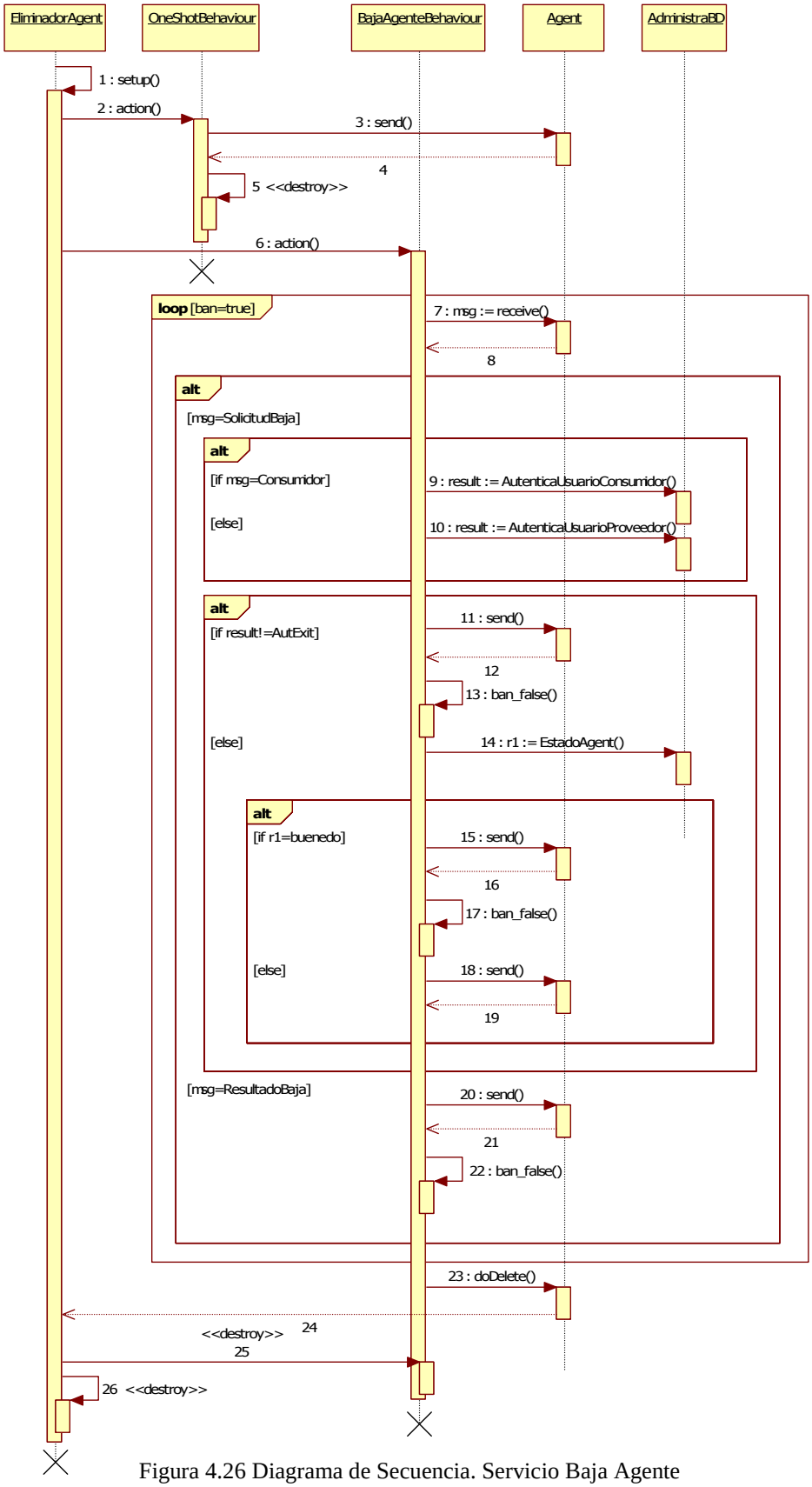

Figura 4.26 Diagrama de Secuencia. Servicio Baja Agente

Descripción:

1. La clase EliminadorAgent se inicializa a través del método setup.
2. El EliminadorAgent ejecuta la operación action de la clase OneShotBehaviour.
3. OneShotBehaviour ejecuta un send de la clase Agent, para enviar la información al agente solicitante, de quien lo atenderá en el proceso de su baja.
4. La clase Agent retorna el control a la clase OneShotBehaviour.
5. La clase OneshotBehaviour se autodestruye.
6. El EliminadorAgent ejecuta la operación action de la clase BajaAgenteBehaviour.

Ciclo: Mientras ban=true

7. La clase BajaAgenteBehaviour ejecuta un receive de la clase Agent, a través de esta operación la clase BajaAgenteBehaviour podrá recibir una solicitud de baja agente o el resultado de la baja por parte del agente administrador.
8. La clase Agent retorna el control a la clase BajaAgenteBehaviour.

Alternativa: Si msg=SolicitudBaja

Alternativa: Si msg=Consumidor

9. La clase BajaAgenteBehaviour invoca la operación AutenticaUsuarioConsumidor de la clase AdministraBD.

Alternativa: else

10. La clase BajaAgenteBehaviour invoca la operación AutenticaUsuarioProveedor de la clase AdministraBD.

Alternativa: Si result!=AutExit

11. La clase BajaAgenteBehaviour invoca el método send de la clase Agent, para enviar un mensaje de rechazo ingreso.
12. La clase Agent retorna el control a la clase BajaAgenteBehaviour.
13. BajaAgenteBehaviour ejecuta ban_false.

Alternativa: else

14. La clase BajaAgenteBehaviour ejecuta el método EstadoAgent, con la finalidad de obtener el historial del agente.

Alternativa: Si r1=buenedo

15. La clase BajaAgenteBehaviour ejecuta el método send de la clase Agent, para enviar un mensaje de rechazo baja.
16. La clase Agent devuelve el control a la clase BajaAgenteBehaviour.
17. BajaAgenteBehaviour ejecuta la operación ban_false.

Alternativa: else

18. La clase BajaAgenteBehaviour ejecuta el método send de la clase Agent, para enviar una solicitud de baja agente al agente administrador.
19. La clase Agent devuelve el control a la clase BajaAgenteBehaviour.

Alternativa: Si msg=ResultadoBaja

20. BajaAgenteBehaviour ejecuta el método send de la clase Agent, para enviar el resultado de la baja al agente solicitante.
21. La clase Agent devuelve el control a la clase BajaAgenteBehaviour.
22. BajaAgenteBehaviour ejecuta la operación ban_false.
23. BajaAgenteBehaviour ejecuta el método doDelete de la clase Agent.
24. La clase Agent devuelve el control a la clase EliminadorAgent.
25. El EliminadorAgent destruye la clase BajaAgenteBehaviour.
26. El EliminadorAgent se autodestruye.

Parte 3: Baja Agente del Mercado

El diagrama mostrado en la figura 4.27 modela las interacciones que ocurren cuando se da de baja a un agente consumidor o proveedor.

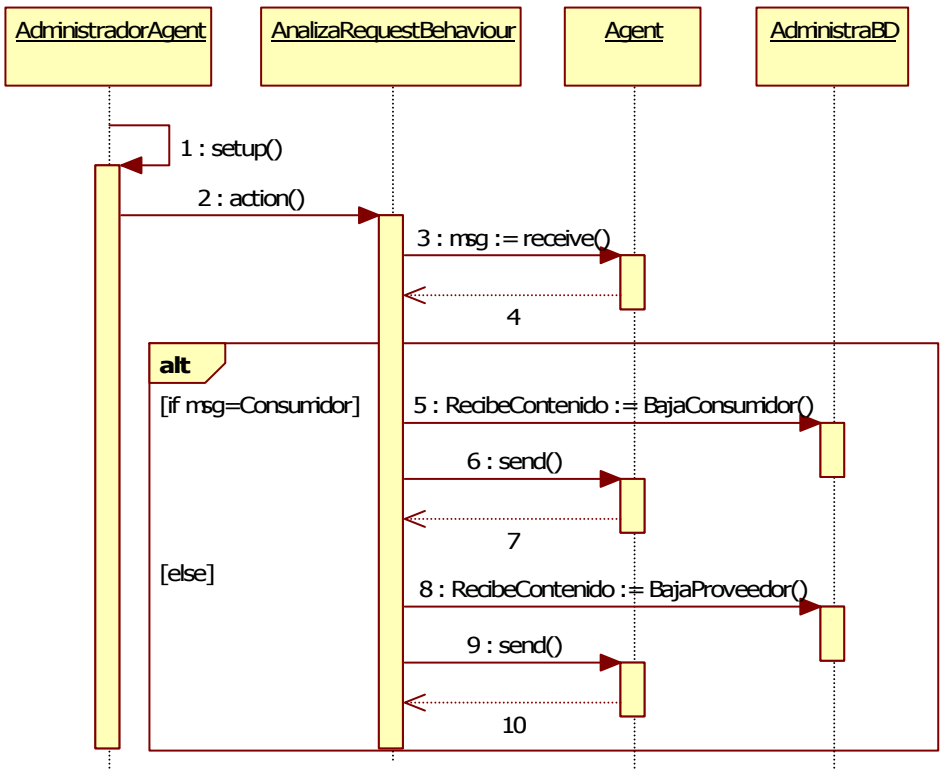

Figura 4.27 Diagrama de Secuencia. Baja Agente del Mercado

Descripción:

1. El objeto AdministradorAgent se inicializa.
2. El AdministradorAgent ejecuta el método action del objeto AnalizaRequestBehaviour.
3. AnalizaRequestBehaviour ejecuta el método receive del objeto Agent, para recibir las solicitudes de baja, por parte del agente eliminador de agentes.
4. El objeto Agent retorna el control al objeto AnalizaRequestBehaviour.

Alternativa: Si msg=Consumidor

5. AnalizaRequestBehaviour ejecuta el método BajaConsumidor del objeto AdministraBD.

6. AnalizaRequestBehaviour ejecuta el método send del objeto Agent, para enviar el resultado de la baja de una agente consumidor.
7. El objeto Agent retorna el control al objeto AnalizaRequestBehaviour.

Alternativa: else

8. AnalizaRequestBehaviour ejecuta el método send del objeto Agent, para enviar el resultado de la baja de una agente proveedor.
9. El objeto Agent retorna el control al objeto AnalizaRequestBehaviour.

4.2.8 Diagrama de Secuencia: Entrega del OA

En la figura 4.28 se muestra el diagrama que modela las interacciones que ocurren, cuando el agente administrador recibe una solicitud de entrega de OA, por parte de un agente proveedor.

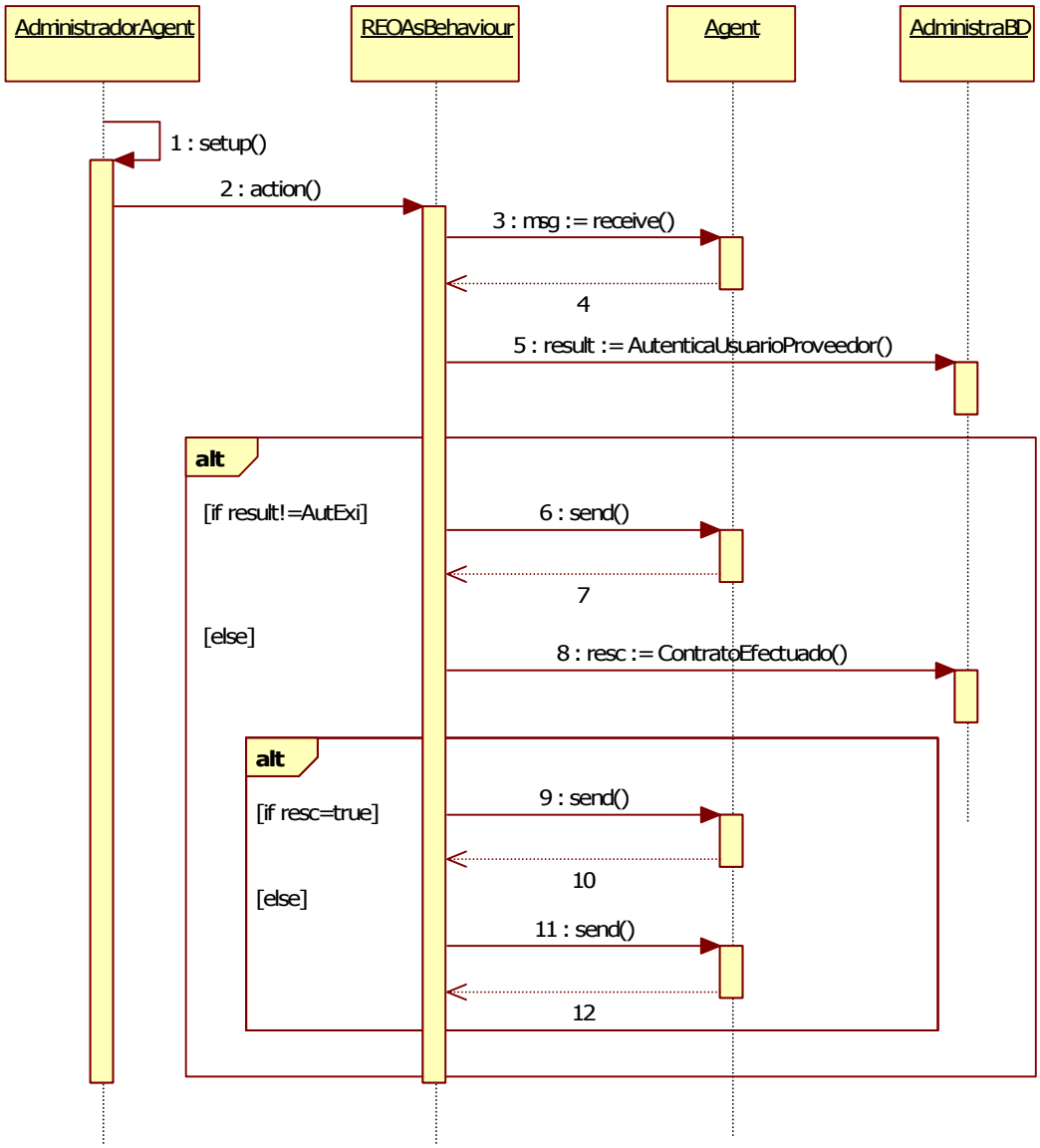

Figura 4.28 Diagrama de Secuencia. Entrega del OA

Descripción:

1. El objeto AdministradorAgent se inicializa.
2. El AdministradorAgent ejecuta el método action del objeto REOAsBehaviour.
3. El REOAsBehaviour ejecuta el método receive de la clase Agent, para recibir la petición de entrega OA.
4. El objeto Agent devuelve el control al objeto REOAsBehaviour.
5. REOAsBehaviour ejecuta la operación AutenticaUsuarioProveedor del objeto AdministraBD.

Alternativa: Si result!=AutExit

6. REOAsBehaviour ejecuta una operación send del objeto Agent, para enviar un mensaje de fallo validación.
7. El objeto Agent devuelve el control al objeto REOAsBehaviour.

Alternativa: else

8. REOAsBehaviour ejecuta el método ContratoEfectuado del objeto AdministraBD, con ello se verifica que tanto un agente consumidor como un agente proveedor hayan llevado a cabo la negociación del OA a recibir.

Alternativa: Si resc=correcto

9. REOAsBehaviour ejecuta la operación send del objeto Agent, para enviar un mensaje con los datos de password y contraseña al agente proveedor.
10. El objeto Agent devuelve el control al objeto REOAsBehaviour.

Alternativa: else

11. REOAsBehaviour ejecuta la operación send del objeto Agent, para enviar un mensaje de registro no autorizado al agente proveedor.
12. El objeto Agent devuelve el control al objeto REOAsBehaviour.

4.2.9 Diagrama de Secuencia: Acceso al OA

El diagrama modelado en la figura 4.29, muestra el conjunto de interacciones para monitorear cada cierto tiempo la base de datos local del MOA, esto con el propósito de verificar si algún agente proveedor o agentes proveedores han depositado OAs en ella, de ser así, el agente administrador se lo hará saber al agente consumidor correspondiente, para que este a su vez pueda obtenerlo.

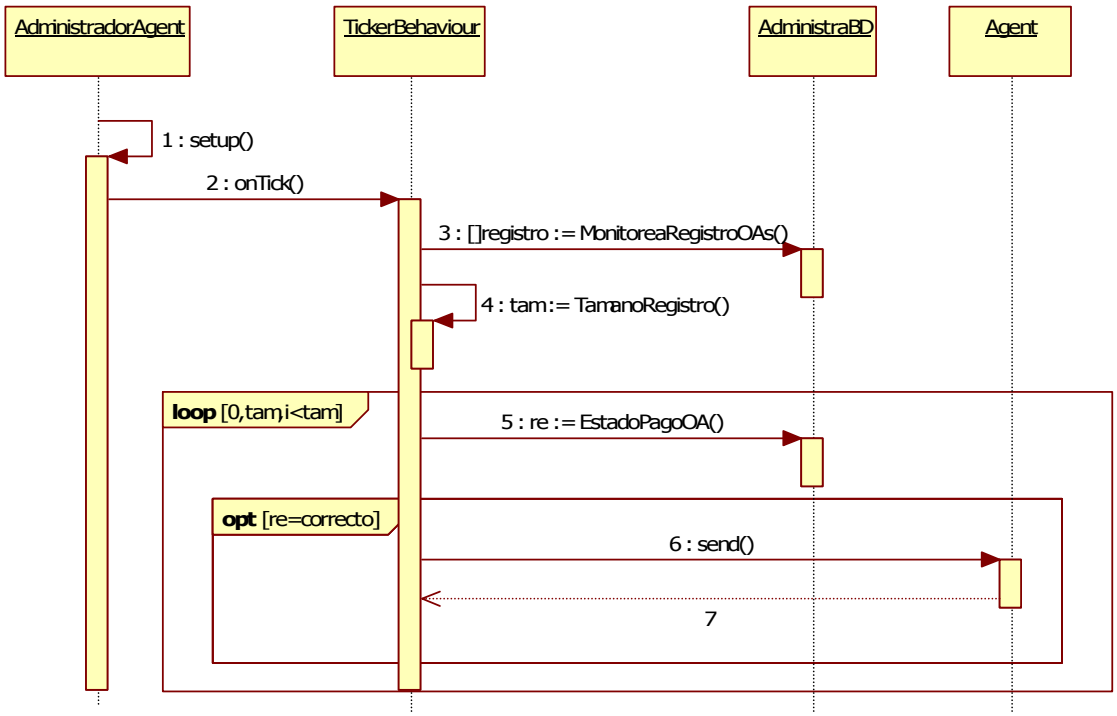

Figura 4.29 Diagrama de Secuencia. Acceso al OA

Descripción:

1. El objeto AdministradorAgent ejecuta su setup.
2. El AdministradorAgent ejecuta el método onTick del objeto TickerBehaviour, este método tiene la característica de ejecutarse cada cierto tiempo.
3. El objeto TickerBehaviour ejecuta la operación MonitoreaRegistroOAs del objeto AdministraBD.
4. TickerBehaviour ejecuta el método TamanoRegistro.

Ciclo: Desde i=0, mientras i<tam

5. TickerBehaviour ejecuta el método EstadoPagoOA del objeto AdministraBD.

Opción: Si re=correcto

6. TickerBehaviour ejecuta un send del objeto Agent, para notificar al agente consumidor la recepción en el mercado de su OA.
7. El objeto Agent devuelve el control al objeto TickerBehaviour.

4.3 Estructura de la Base de Datos Local del MOA

Retomando el concepto de bases de datos descrito en el capitulo 2 y llevándolo al contexto de nuestro mercado de objetos de aprendizaje, exponemos el porque la necesidad de hacer uso de una base de datos y no de un sistema de procesamiento de archivos.
La diferencia principal entre un sistema de procesamiento de archivos y un DBMS radica en la manera de almacenar, recuperar y actualizar los datos; lo anterior se debe a que con los sistemas de procesamiento de archivos, como su nombre lo indica, los datos se guardan en diversos archivos y muchas veces dichos archivos son de diferente formato, ocasionando problemas como la redundancia que trae consigo el almacenamiento y además de esto dificulta el acceso a los datos, por esto se justifica la necesidad de hacer uso de una base de datos y no de un sistema de procesamiento de archivos.

A partir del análisis de los diferentes diagramas de secuencia, puede apreciarse que prácticamente todos los agentes que se encuentran dentro del mercado, manipulan la base de datos, la única diferencia radica en que el agente administrador y el agente manejador de contratos tienen permiso de lectura/escritura y el resto de los agentes sólo de lectura.
La base de datos local del mercado de objetos de aprendizaje contendrá toda la información relacionada con los agentes proveedores de un servicio, así como de los agentes consumidores de los servicios proveídos, aunado a lo anterior también almacenará todos los contratos que se hayan llevado a cabo entre los agentes proveedores y consumidores registrados ante el MOA.

En la Figura 4.30 se muestra el modelo Entidad-Relación de la Base de Datos Local que contiene el MOA, donde se almacena la información apropiada tanto de los agentes proveedores como de los agentes consumidores; en el modelo Entidad-Relación el agente proveedor distribuye OAs, es decir, se realiza un registro de los proveedores que estén proporcionando algún tipo de servicio y por consecuencia se tienen las descripciones de los OAs, mismos que están relacionados al proveedor que esté publicitando el servicio. Así también, el proveedor esta asociado a un contrato de copra-venta de OA. De la misma forma, se tienen almacenados los datos de los agentes consumidores, los cuales al igual

que los agentes proveedores, se encuentran relacionados con la información de los contratos de compra-venta que han llevado a cabo cuando han adquirido algún OA, por último cada contrato esta asociado a un OA que haya enviado un agente proveedor para su entrega a un agente consumidor.

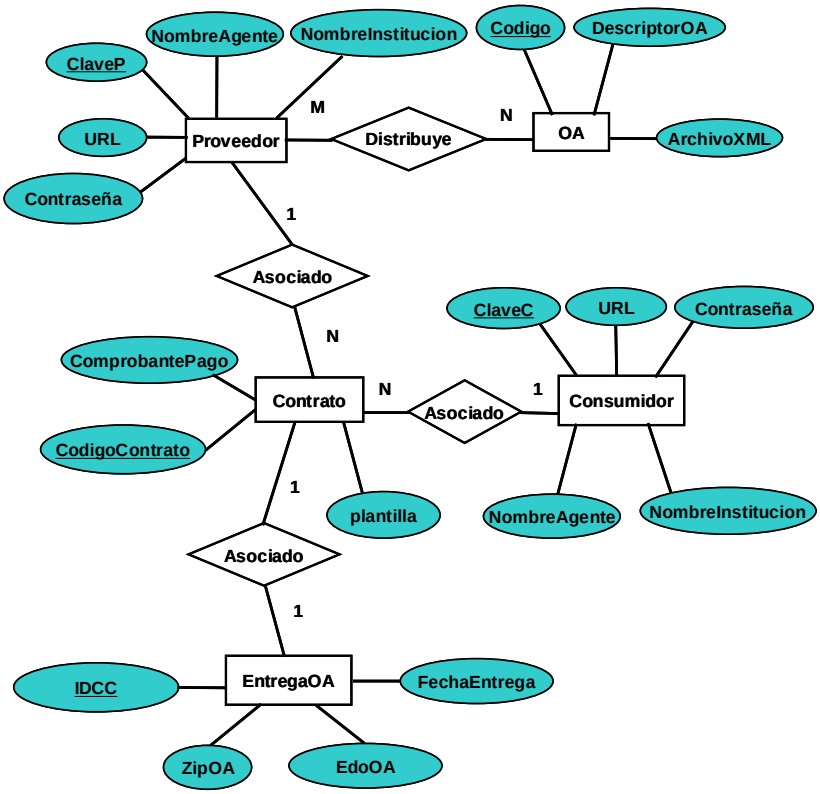

Figura 4.30 Modelo Entidad-Relación

Capítulo 5

Implementación

5.1 Lenguaje y Tecnología a Utilizar

En este capítulo se describirán las herramientas y la tecnología que se utilizó para desarrollar la primera versión del mercado de objetos de aprendizaje visto como un sistema multiagente.

5.1.1 Java Agents Development Environment (JADE)

Es una plataforma estable y eficiente que cumple con FIPA y está basada en código abierto, fue desarrollada por TILAB. JADE es ampliamente usado, tanto en investigación como en proyectos comerciales y tiene una amplia comunidad de usuarios y desarrolladores activos.
JADE (Java Agent DEvelopment Framework) es un sistema de software totalmente implementado en el lenguaje JAVA. Simplifica la implementación de sistemas multiagentes a través de un “middle-ware” que cumple con las especificaciones de la FIPA y contiene un conjunto de herramientas gráficas que soportan las fases de depurado y despliegue. La plataforma del agente puede ser distribuida a través de máquinas (que no necesariamente comparten el mismo sistema operativo) y la configuración puede ser controlada vía una GUI remota.

5.1.2 Java

Java es un lenguaje de programación de alto nivel con el que se puede escribir tanto programas convencionales como para Internet. Una de las ventajas significativas de Java sobre otros lenguajes de programación es que es independiente de la plataforma, tanto en código fuente como en binario. Esto quiere decir que el código producido por el compilador Java puede transponerse a cualquier plataforma que tenga instalada una máquina virtual Java y ejecutarse [8].

El lenguaje de programación Java fue desarrollado por Sun Microsystems en 1991. Nace como parte de un proyecto de investigación para desarrollar software para comunicación entre aparatos electrodomésticos. Durante la fase de investigación surge un problema que dificultaba el proyecto iniciado; cada aparato tenia un microprocesador diferente y muy poco espacio de memoria, esto provocó el desarrollo de un nuevo lenguaje de programación independiente del dispositivo, mismo que fue bautizado como Oak [8].

La explosión de Internet en 1994, gracias al navegador gráfico Mosaic para la Word Wide Web (WWW), no pasó desapercibida por la Sun, se dieron cuenta que los logros alcanzados en su proyecto eran aplicables a Internet, debido a que el Internet es una gran red mundial que conecta múltiples ordenadores con diferentes sistemas operativos y diferentes arquitecturas de microprocesadores, pero todos tienen en común un navegador para comunicarse entre si, así pues la Sun dirige sus investigaciones hacia el desarrollo de un lenguaje que permitiera crear aplicaciones que se ejecuten en cualquier ordenador de Internet con el único soporte de un navegador.

5.1.3 MySQL

MySQL es un gestor de base de datos sencillo de usar, e increíblemente rápido. También es uno de los motores de base de datos más usados en Internet, la principal razón de esto es que es gratis para aplicaciones no comerciales [9].

MySQL es un sistema de gestión de base de datos, multihilo y multiusuario con más de seis millones de instalaciones [10]. MySQL AB desarrolla MySQL como software libre en un esquema de licenciamiento dual. Por un lado lo ofrece bajo la GNU GPL, pero, empresas que quieran incorporarlo en productos privativos pueden comprar a la empresa una licencia que les permita ese uso. Está desarrollado en su mayor parte en ANSI C [10].

5.1.4 Java DataBase Connectivity (JDBC)

JDBC (conexión con bases de datos desde Java) es una API (Application Programming Interface – interfaz de programación de aplicaciones) de Java, proporcionada por un conjunto de clases, que permite ejecutar instrucciones SQL para manipular y gestionar bases de datos relacionales [11].

Para que una aplicación Java pueda hacer operaciones sobre una base de datos, previamente tiene que establecer una conexión con la misma. Esta conexión se realiza a través de un controlador (driver). La función de este controlador es traducir los mensajes propietarios de bajo nivel del sistema base de datos, a mensajes de bajo nivel de la API JDBC y viceversa.

Diferentes fabricantes proporcionan controladores (drivers) específicos para acceder a sus sistemas de bases de datos. Un controlador JDBC es una clase que implementa la interfaz que proporciona los métodos necesarios para acceder a los datos de la base de datos.

La API JDBC se proporciona en dos paquetes: java.sql y javax.sql. Ambos paquetes, que forman parte de J2SE, contienen las interfaces y clases Java fundamentales de JDBC que nos permiten el acceso a las bases de datos.

5.1.5 Lenguaje de Comunicación de Agentes (FIPA-ACL)

El intercambio de mensajes es el aspecto central de la comunicación entre agentes. Un mensaje FIPA-ACL encapsula el contenido de estos mensajes utilizando una serie de atributos definidos y estandarizados por la FIPA [FIPA SC00023J, 2003][12].

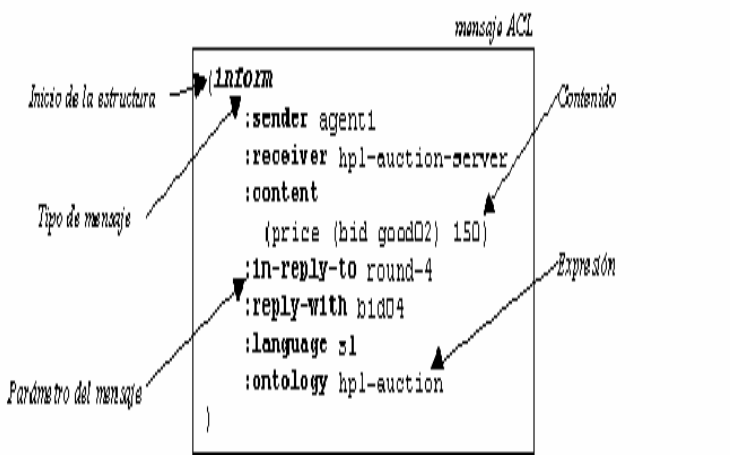

Figura 5.1 Estructura de un Mensaje ACL

Un mensaje se compone de una instrucción para indicar el tipo de acto comunicativo (pregunta, petición, etc.) y de diferentes parámetros, ver la figura 5.1.

Existen diferentes tipos de mensajes definidos por la FIPA según el acto comunicativo, entre los cuales podemos mencionar: accept-proposal, agree, cancel, cfp, failure, inform, not-understood, propose, query-if, refuse, reject-proposal, request, request-when, request-whenever, subscribe. Para nuestro propósito los mensajes que se utilizaran son los que se muestran en tabla 5.1:

REQUEST	Pide que se realice una acción.
AGREE	Acepta petición.
REFUSE	No acepta la petición.
CANCEL	Cancela petición.
INFORM	Da información al receptor.
QUERY-IF	Pregunta algo al receptor.
SUBSCRIBE	Apuntarse a un servicio del receptor.
CFP	Pide propuestas para realizar una acción.
PROPOSE	Hace una propuesta.
REJECT-PROPOSAL	Rechaza una propuesta.
ACCEPT-PROPOSAL	Acepta la propuesta.
FAILURE	Explica el fallo de la acción.
NOT-UNDERSTOOD	Envía mensaje no entendido.

Tabla 5.1 Performativas de los Mensajes ACL

5.1.6 (Extensible Markup Language) XML

Hasta hace menos de cuatro años la Web se componía principalmente de páginas de texto que no permitían actuar sobre la información contenida en ellas, sino que sólo la mostraban. Después de varios años viendo cómo nuevos lenguajes intentaban minimizar este problema, el W3C decidió sacar a la luz la primera versión de XML, un lenguaje pensado para dar vida a la información [13].

XML se considera un lenguaje descriptivo cuyos elementos se encargan de indicar cuál va a ser la estructura del documento, indicando párrafos, citas, listas y demás elementos dentro de éste.

El lenguaje XML permite definir lenguajes de marcado propios, con sus propias etiquetas y su propia estructura. Gracias a XML los datos que se introducen en un documento construido con dicho lenguaje, pueden ser procesados, la posibilidad de manejar los datos de los documentos XML no se limita sólo a los parsers (programas encargados de interpretar los lenguajes de marcado) [13].
En nuestro caso con el lenguaje XML representaremos las diversas plantillas que deben llenar los diferentes agentes al realizar determinadas operaciones en el MOA.

5.2 Implementación del Sistema Multi-Agente

Para llevar a cabo la implementación de nuestro sistema multi-agente nos apoyamos en la tecnología JADE descrita en la sección 5.1.1; antes de introducirnos al desarrollo del sistema multi-agente es necesario hacer hincapié que la tecnología JADE responde a los estándares FIPA de organización de sistemas multi-agentes y de la comunicación entre agentes. De acuerdo con esta última, los agentes son entidades de software que se encuentran localizados en una única plataforma.

Para conocer la estructura de los agentes implementados en nuestro sistema multi-agente es necesario introducir algunos conceptos que nos ofrece dicha tecnología [14]:
Plataforma de agentes: Entorno de ejecución en donde se encuentran ejecutándose los agentes y a través de cual se dispone de dos servicios: el servicio de páginas blancas para buscar otros agentes, y el servicio de páginas amarillas para buscar servicios que ofrecen otros agentes.

Módulo de gestión: A través del modulo de gestión, se accede a las facilidades de que disponen los servicios.

Sistema de envió/recepción de mensajes: Sistema mediante el cual el AMS (Agent Management Service) puede hacer llegar a su destino todos los mensajes que generan los agentes situados en la misma plataforma.

Contendor: Un contenedor es una instancia del entorno de ejecución JADE, que permite albergar un número indeterminado de agentes. Así pues, se cuenta con un contenedor principal, uno por cada plataforma FIPA, el resto de los contenedores de la plataforma, una vez ejecutados deben subscribirse al contenedor principal.

5.2.1 Creación de Agentes JADE

Para trabajar con agentes se debe conocer la clase: jade.core.Agent; para realizar la creación de un agente JADE se debe crear la clase agente que herede de ésta y que implemente mínimamente el método setup().

```
import jade.core.Agent;

public class BetaAgent extends Agent {
        protected void setup() {
         ...
        }
}
```

Método setup(): Contiene el código para llevar a cabo las tareas de inicialización del agente, el cual se ejecuta cuando se va a lanzar el agente para su ejecución.

5.2.2 Tareas Mediante Comportamientos JADE

Para implementar los comportamientos de un agente (s) es necesario crear una clase, la cual debe heredar de la clase: jade.core.behaviours.Behaviour. Una vez que se ha implementado, es suficiente para que el agente ejecute ese comportamiento que se invoca en el cuerpo del agente. Toda clase que herede de la de comportamiento, deberá implementar el método action() , donde debe incluirse el código de las acciones correspondientes.

```
public class BetaBehaviour extends Behaviour
{
        public void action() {
                while (true) {
                        // do something
                }
        }
        public boolean done() {
        return true;
        }
}
```

5.2.3 Clasificación de Comportamientos en JADE

Podemos agrupar los comportamientos JADE en tres grandes grupos:

1. Comportamientos one-shot: Comportamiento que se ejecutan de manera casi instantánea, y solamente una vez.
2. Comportamientos cíclicos: Comportamientos que nunca son sacados del conjunto de comportamientos del agente y cuyo método action() siempre ejecuta el mismo código; por lo tanto, nunca finalizan.
3. Comportamientos genéricos: El código que se ejecuta en ellos depende del estatus del agente, y eventualmente finalizan su ejecución.

5.2.4 Implementación del Agente Administrador

El mercado de objetos de aprendizaje se encuentra implementado en un 100%. Para efectos de esta tesis se mostrará parte de la estructura de nuestro Agente Administrador, así como la estructura de los subagentes que éste crea, para responder a cada uno de los servicios que brinda el MOA.

Implementación: Agente Administrador

```
public class AdministradorAgent extends Agent {
        protected void setup(){
                addBehaviour(new OneShotBehaviour() {
                        public void action() {
                        }
                } );

                addBehaviour(new TickerBehaviour(){
                        protected void onTick() {
                        }
                } );

                addBehaviour(new AnalizaMensajesBehaviour());
                addBehaviour(new AnalizaRequestBehaviour());
                addBehaviour(new REOAsBehaviour());
        }

        private class AnalizaMensajesBehaviour extends CyclicBehaviour {
                public void action()  {
                 }
        }

        private class AnalizaRequestBehaviour extends CyclicBehaviour{
                public void action(){
                }
        }

        private class REOAsBehaviour extends CyclicBehaviour{
                public void action(){
                }
        }

         protected void takeDown(){
        }
}
```

La clase AdministradorAgent hereda de la clase Agent; permitiéndonos crear al agente administrador del mercado de objetos de aprendizaje. Dentro de la clase AdministradorAgent se encuentra el método setup() que permite establecer las tareas de inicialización del agente administrador, tales como: la creación de todas las variables para cada nombre de los agentes involucrados en el sistema multi-agente. Así también, nuestro agente está compuesto por un comportamiento OneShotBehaviour, un comportamiento TickerBehaviour y tres comportamientos cíclicos (AnalizaMensajesBehaviour, AnalizaRequestBehaviour y REOAsBehaviour), conteniendo cada uno de ellos el método action() para poder llevar a cabo las acciones pertinentes.

En el comportamiento AnalizaMensajesBehaviour el agente analiza las peticiones que recibe de los agentes externos al MOA y en dependencia de ello realiza las acciones necesarias para crear al agente que proveerá el servicio que se encuentre solicitando un agente consumidor o proveedor.

En el comportamiento AnalizaRequestBehaviour el agente recibe solamente peticiones de los agentes internos del sistema, es decir, de aquellos agentes contenidos en el mercado de objetos de aprendizaje, y en dependencia de la solicitud que recibe realiza alguna operación sobre la base de datos local del mercado a través de la clase AdministraDB.

En el comportamiento REOAsBehaviour el agente administrador sólo puede escuchar aquellas peticiones que vienen por parte del agente proveedor, al solicitarle el servicio de entrega de un OA.

El comportamiento OneShotBehaviour, le permite al agente administrador crear de manera permanente al agente Manejador de Contratos, ya que éste permanece vivo durante la ejecución del mercado.

El comportamiento TickerBehaviour, le permite al agente administrador monitorear cada cierto tiempo su base de datos local, con el propósito de percibir si un agente proveedor ha depositado algún OA en la base de datos, de ser así, el agente administrador envía el mensaje para recoger el OA al agente consumidor correspondiente.

A continuación se presenta una breve descripción de los subagentes que lanza el Agente Administrador para el buen funcionamiento del Mercado de Objetos de Aprendizaje a través de los diversos comportamientos con que cuenta el Agente Administrador. Estos subagentes siguen el mismo patrón de diseño, cuya única diferencia son los tipos de comportamientos que usan, de acuerdo al servicio que cada uno de ellos provea al mercado de objetos de aprendizaje.

Agente de Registro

```
public class RegistradorAgent extends Agent {
        protected void setup() {
                addBehaviour(new OneShotBehaviour() {
                        public void action() {
                        }
                } );
                addBehaviour(new AnalizaMensajesBehaviour());
         }

        private class AnalizaMensajesBehaviour extends CyclicBehaviour{
                public void action() {
                }
        }
}
```

De la mima forma que en el agente administrador la clase RegistradorAgent hereda de la clase Agent; permitiéndonos crear al agente de registro para poder ofrecer a los agentes solicitantes sean consumidores o proveedores, su registro en el mercado de objetos de aprendizaje. Dentro de la clase RegistradorAgent se encuentra el método setup() que permite establecer las tareas para lanzar al agente de registro. A diferencia del agente administrador, el agente de registro sólo contiene dos comportamientos, el primero un OneShotBehaviour, el cual tiene la función de enviar al agente correspondiente un mensaje, informándole quién le proveerá el servicio de registro, y el segundo comportamiento de tipo cíclico, nombrado AnalizaMensajesBehaviour, el cual contiene su correspondiente método action() para lograr el servicio de registro. Así pues, dentro del método action() se invoca a

métodos tales como ExtraePlantilla(), AnalizaPlantilla() y GenerarClaveAcceso(), todos ellos usados para proveer el registro de un agente proveedor o consumidor.

Se realizaron pruebas al agente de registro para comprobar su funcionamiento dependiendo de las solicitudes requeridas por parte de los agentes solicitantes, las tareas que realiza dicho agente de manera satisfactoria son:

- Reconocimiento del tipo de mensaje enviado por algún agente solicitante.
- Envío de la plantilla XML de registro al agente correspondiente para su registro en el MOA.
- Análisis de la plantilla XML recibida por parte del agente solicitante.
- Generación de la clave de acceso correspondiente, para el agente solicitante.
- Envió correcto de la información al agente administrador para almacenarla en la base de datos local del MOA.
- Envío de mensaje de registro exitoso al agente correspondiente una vez que reciba por parte del agente administrador un mensaje de registro exitoso.

Agente Publicador de OA

```
public class PublicadorAgent extends Agent{
        protected void setup() {
                addBehaviour(new OneShotBehaviour() {
                        public void action() {
                        }
                } );

                addBehaviour(new PublicarBehaviour());
        }

        private class PublicarBehaviour extends CyclicBehaviour{
                public void action(){
                }
        }
}
```

La implementación del agente publicador se llevó a cabo a través de la clase PublicadorAgent, la cual hereda de la clase Agent. Como todo agente, el PublicadorAgent consta del método setup() que permite realizar las tareas de inicialización del agente. Para lograr proveer el servicio de publicación el agente consta de dos comportamientos, un OneShotBehaviour para informar al agente proveedor de quién atenderá sus peticiones de publicación, y un PublicarBehaviour, que es donde se realiza todo el servicio de publicación, este comportamiento consta de métodos tales como: AutenticaUsuarioProveedor(), VerificaPlantilla(), GenerarCodigo(), entre otros, todos ellos en conjunto dan vida a nuestro agente de publicación.

Agente Localizador de Proveedores de OA

```
public class LocalizadorAgent extends Agent{
        protected void setup() {
                addBehaviour(new OneShotBehaviour() {
                        public void action() {
                        }
                 } );

                addBehaviour(new AnalizaMensajesBehaviour());
         }

        private class AnalizaMensajesBehaviour extends CyclicBehaviour{
                public void action(){
                }
        }
}
```

Para llevar a cabo la implementación del agente localizador de proveedores de OA, también se usa la clase Agent que nos ofrece jade, con ello de igual forma se crea a nuestro LocalizadorAgent, tal como se realizó con los agentes anteriormente descritos. Al igual que los agentes anteriores, este agente contiene un método setup(), un comportamiento OneShotBehaviour y un comportamiento CyclicBehaviour al cual denominamos

AnalizaMensajesBehaviour. En el método setup() lanzamos el comportamiento del agente para llevar a cabo la búsqueda de proveedores de un OA.

Por su parte en el comportamiento OneShotBehaviour como en los comportamientos anteriormente descritos, se envía un mensaje al agente correspondiente para informarle quién atenderá sus peticiones de localización de proveedores. Así también, el comportamiento cíclico AnalizaMensajesBehaviour lleva a cabo todas las acciones que involucra la búsqueda(s) de un proveedor de OA´s. Dentro de las acciones contenidas en este comportamiento, se realiza la autenticación del agente consumidor, el cual solicita el servicio de búsqueda de un OA, dicha autenticación es llevada a cabo tras la ejecución del método AuntenticaUsuarioProveedor() de la clase AdministraBD. Una vez realizada la autenticación, el agente procede a efectuar la acción de búsqueda del proveedor de OA solicitado, mediante la descripción enviada por el agente solicitante, dicha búsqueda se realiza a través del método BusquedaOA() de la clase AdministraBD.

Agente Manejador de Contratos

```
public class ManejadorContratosAgent extends Agent {
        protected void setup() {
                addBehaviour(new EnviaContratosBehaviour());
                addBehaviour(new VerificaContratosBehaviour());
        }

        private class EnviaContratosBehaviour extends CyclicBehaviour{
                public void action(){
                }
        }

        private class VerificaContratosBehaviour extends CyclicBehaviour{
                public void action(){
                }
        }
}
```

El agente manejador de contratos consta de una clase denominada ManejadorContratosAgent que extiende de la clase Agent, contiene un método setup() con el cual inicia su ejecución el agente. Consta de dos comportamientos cíclicos, el primero llamado EnviaContratosBehaviour y el segundo VerificaContratosBehaviour.

El comportamiento EnviaContratosBehaviour tiene como función autenticar al agente proveedor a través del método AutenticaUsuarioProveedor() de la clase AdministraBD, una vez realizada la autenticación, a través del método FolioUpdate() se actualizan algunos datos de la plantilla del contrato xml, una vez que los datos han sido actualizados, la plantilla es cargada a un flujo de datos usando el método ReadContratoActual() y finalmente éste es enviado al agente solicitante a través de la clase Agent.

El comportamiento VerificaContratosBehaviour se encarga de proveer el servicio de registro de contrato. Este servicio es llevado a cabo a través de los métodos que se citan a continuación: AutenticaUsuarioProveedor(), AutenticaUsuarioConsumidor(), Lee_Folio(), BuscaContrato(), CompararContratos(), RegistraClaveContrato(), ObtenURLAgente() y BorrarContrato(). Cuando un agente solicita el registro de un contrato, en primera instancia es autenticado a través del método AutenticaUsuarioProveedor() si es un agente proveedor, o a través del método AutenticaUsuarioConsumidor() si es un agente consumidor. Una vez realizada la autenticación, el agente procede a leer el folio del contrato, usando el método Lee_Folio(). Una vez obtenido el folio del contrato, el agente busca el contrato en la base de datos local del MOA, si el contrato no existe en la base de datos, el agente procede a guardarlo, de lo contrario extrae la contraparte del contrato ya almacenado en la base de datos, una vez realizado lo anterior, el manejador de contratos realiza la comparación de los contratos a través del método CompararContratos(), si como resultado se obtiene que los contratos son iguales, el agente manejador procede a registrar la clave del último agente que envió el contrato para ser registrado haciendo uso del método RegistraClaveContrato(), para finalizar el servicio el agente manejador obtiene la dirección URL del primer agente que envió el contrato para ser registrado, con la finalidad de enviar los mensajes correspondientes a los agentes de registro de contrato exitoso o no exitoso, esto a través del método ObtenURLAgente(). Por otra parte si los contratos son diferentes, entonces se procede a borrar el contrato almacenado en la base de datos, para tal vez su posterior recepción.

Agente Eliminador de OA

```
public class EliminadorOAsAgent extends Agent{
	protected void setup() {
		addBehaviour(new OneShotBehaviour() {
			public void action() {
			}
		} );
		addBehaviour(new EliminarOAsBehaviour());
	}

	private class EliminarOAsBehaviour extends CyclicBehaviour{
		public void action(){
		}
	}
}
```

El agente eliminador de OAs hereda de la clase Agent, y contiene a su vez el método setup() para inicializarse a través de éste, contiene dos comportamientos: un OneShotBehaviour y un comportamiento cíclico llamado EliminarOAsBehaviour(), el comportamiento OneShotBehaviour únicamente es usado para enviar la información correspondiente de quién atenderá al agente que solicite la baja de un OA del mercado. Por otro lado el comportamiento EliminarOAsBehaviour, se encarga de proveer el servicio de baja del OA a través del método VerificaSolicitudBaja(), claro esta, autenticando al agente previamente a través del método AutenticaUsuarioProveedor() de la clase AdministraBD.

Agente Eliminador de Agentes

```
public class EliminadorAgent extends Agent{
	protected void setup() {
		addBehaviour(new OneShotBehaviour() {
			public void action() {
			}
		} );

		addBehaviour(new BajaAgenteBehaviour());
	}

	private class BajaAgenteBehaviour extends CyclicBehaviour{
		public void action(){
		}
	}
}
```

Este agente posee el mismo diseño e implementación que el agente eliminador de Objetos de Aprendizaje.

Capítulo 6

Conclusiones

- Apoyándonos en la Arquitectura Orientada a Servicios se definieron las metas y tareas detalladas que contiene cada agente participante en el MOA visto como un sistema multiagente.
- Se identificó cada fase del funcionamiento del MOA.
- Se llevó a cabo el diseño e implementación de la base de datos local del MOA en Mysql.
- Se realizó el análisis y diseño de nuestro sistema multiagente, siguiendo el lenguaje unificado de desarrollo de software (UML), para el modelado de casos de uso, diagramas de clases y diagramas de secuencia.
- La implementación del sistema multi-agente se completó en un 100 %.
- Cada agente desarrollado posee comportamientos apropiados para poder llevar a cabo las acciones que permitan la culminación de una solicitud de servicio.
- Actualmente se cuenta totalmente desarrollada la primera versión del mercado de objetos de aprendizaje visto como un sistema multi-agente.
- Como resultado del proceso de investigación desarrollado se han publicado 2 artículos.

Recomendaciones

- Modelar todos los protocolos de comunicación, de forma confiable, remota y segura.

- Desarrollar los mecanismos de seguimiento del comportamiento de los agentes con respecto a los contratos.

Referencias

[1] C. Klingler, G.Vadillo, Psicología cognitiva estrategias en la práctica docente. McGraw-Hill/Interamericana Editores, Primera Edición, pp. 7-11.

[2] H. S. Nwana, Software agents: An overview. The Knowledye Engineering Review. 11(3):1996, pp. 205-244.

[3] S. Franklin, A. Graesser, Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents, Third International Workshop on Agent Theories, Architectures, and Languages. 1996.

[4] De Roure, N. Jennings, and N. Shadbolt, The Semantic Grid: A future e-science infraestructure. Grid Computing, 2003, pp. 437-470.

[5] C. Navarrete, Introducción a las bases de datos. Centro de Referencia Linux UAM-IBM.

[6] D. Vilariño, Notas: BASES DE DATOS CLIENTE_SERVIDOR. Verano 2003.

[7] G. Booch, et. al., El Lenguaje Unificado de Modelado. Addison Wesley, 1999, pp.5-50.

[8] F. J. Ceballos, Java 2 Curso de Programación. Alfaomega Grupo Editor, Segunda edición 2003, pp. 3-9.

[9] http://www.webestilo.com/mysql/intro.phtml

[10] http://dev.mysql.com/tech-resources/articles/dispelling-the-myths.html

[11] D. Vilariño, notas: BASES DE DATOS CON JDBC Y JAVA. 2006, pp.1-17.

[12] http://www.etse.urv.es/recerca/banzai/toni/MAS/presentacions/fipa-acl.htm

[13] A. Vázquez, “Navegar en Internet con XML”. Editorial Alfaomega, ISBN: 970 -15-0757-6, 2002.

[14] Juan A. Botía Blaya, Tutorial básico de JADE, Escuela de Primavera de Agentes Patrocinado por Agentcities.es Imaginática 2005, Universidad de Sevilla, 19 de febrero de 2005.