ORIENTACIÓN A OBJETOS Y ORIENTACIÓN A AGENTES: UNA PROPUESTA DE UNIFICACION

Adriana Giret, Luca Cernuzzi

Departamento de Ingeniería Electrónica e Informática
Universidad Católica Nuestra Señora de la Asunción – Paraguay
http://www.uc.edu.py
{agiret,lcernuzz}@uca.edu.py

Oscar Pastor, Emilio Insfrán

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia- España
http://www.dsic.upv.es/
{opastor,einsfran}@dsic.upv.es

Palabras Claves: Orientación a Agentes, Orientación a Objetos, Ingeniería del Software

Resumen

La introducción de la tecnología de agentes a la industria, requiere de metodologías que asistan en todas las fases del ciclo de vida del sistema agente. Sin técnicas adecuadas para soportar este proceso, tales sistemas probablemente no serán lo suficientemente confiables, mantenibles o extensibles, serán difíciles de comprender y sus elementos serán más difícilmente reusables. En lugar de crear métodos completamente nuevos, creemos que métodos de ingeniería de software ya existentes, y que han probado ser exitosos para el modelado de sistemas complejos, podrían constituir un soporte conceptual y organizativo adecuado en la praxis del proceso de construcción de sistemas basados en agentes. Presentamos en este artículo OO-Method for Agents un método de desarrollo automático de software orientado a objetos ampliado para el modelado de agentes.

1 Introducción

Los métodos de producción de software convencionales que parten de un modelo conceptual orientado a objetos no suelen tener la expresividad necesaria para especificar adecuadamente un sistema en el que los agentes son el componente básico. Es necesario aprovechar toda la experiencia adquirida en esos ámbitos de modelado conceptual OO, pero enriqueciéndola adecuadamente para que se pueda capturar ese tipo de propiedades que tan necesarias se hacen en contextos en los que una correcta definición de agentes y de sus propiedades es necesaria.

En este artículo se presenta inicialmente una amplia discusión acerca de las características de los agentes y de las necesidades de un método adecuado para modelar sistemas de agentes. Seguidamente, se presenta un método de producción automática de software –OO-Method- que a partir de un Esquema Conceptual obtiene un producto final software funcionalmente equivalente al mismo. OO-Method especifica agentes de una manera muy simple si lo comparamos con los métodos de desarrollo de aplicaciones orientadas a agentes existentes. Ello lo hace atractivo como método de desarrollo de sistemas agentes, pero al mismo tiempo hace necesario aumentar el poder expresivo de OO-Method en esa dirección, manteniendo sus propiedades formales y de generación automática de código.

2 Métodos de desarrollo de aplicaciones orientadas a agentes

En esta sección se presenta un resumen de las características de los sistemas basados en agentes, de las cualidades de los métodos de desarrollo de éstos sistemas y algunos de los métodos orientados a agentes existentes.

2.1 Sistemas Basados en Agentes

Un *sistema basado en agentes*, es aquel en el cual la principal abstracción utilizada es un *agente*. Los agentes software son probablemente, el área de la tecnología de la información que crece de manera más rápida. A pesar de esto, aún no existe un acuerdo sobre qué es exactamente un agente. En [GRAESS96] se puede encontrar un resumen de las distintas definiciones existentes. Luego de la revisión de varias definiciones de agentes, a nuestro criterio, la que puede ser considerada como la más clara y completa, es la propuesta por Wooldridge y Jennings [WOOLD95].

Existen dos nociones de agente: una débil y otra fuerte

?? Definición débil: un sistema computacional hardware o software que goza de las siguientes propiedades:

i. autonomía: los agentes operan sin una directa intervención de humanos u otros, y tienen cierto grado de control sobre sus acciones y su estado interno;
ii. habilidad social: los agentes interactúan con otros agentes (y posiblemente con humanos) vía algún tipo de lenguaje de comunicación entre agentes;
iii. reactividad: los agentes perciben su ambiente, (que puede ser el mundo físico, un usuario vía una interfaz gráfica, una colección de otros agentes, la INTERNET, o tal vez todos estos combinados), y responden a cambios que ocurren en él;
iv. pro-actividad: los agentes no actúan simplemente en respuesta a su ambiente, son capaces de exhibir comportamiento oportunista, dirigido por objetivos, tomando iniciativas cuando sea apropiado.

?? Definición fuerte: un agente, además de las características anteriores tiene una o más de las siguientes características:

v. nociones mentales: un agente tiene creencias, deseos e intenciones.
vi. racionalidad: realiza acciones a fin de lograr objetivos.
vii. adaptabilidad o aprendizaje
viii. veracidad: un agente no es capaz de comunicar información falsa de propósito.

Un agente fuerte también es llamado “reflectivo” ya que reflexiona sobre su comportamiento en lugar de simplemente reaccionar a estímulos o cambios. En este trabajo se enfatiza la definición fuerte, pero considerando sólo los puntos *i* a *vi*, sin abarcar la veracidad y aprendizaje.

2.2 Propiedades a especificar en el modelado de sistemas basados en agentes.

Siendo que en los sistemas agentes la principal abstracción utilizada es un agente, el método de desarrollo adoptado debe proveer guías que permitan identificar a las entidades del dominio de aplicación que posean características de agentes, proveer técnicas y formalismos que faciliten la especificación de los mismos, en un modelo del dominio del problema, y transformar este modelo abstracto del sistema en un modelo computacional concreto que satisfaga los requisitos del sistema.
En este sentido, podemos identificar dos macro componentes altamente interrelacionadas entre si que permiten cubrir los aspectos de modelado de un sistema basado en agentes; por un lado el modelado del sistema como un todo y por el otro la especificación de las características de cada uno de los agentes componentes del sistema.

2.2.1 Modelado del sistema agente como un todo

Para modelar el sistema agente, en primer lugar es necesario identificar las entidades (agentes, objetos y usuarios humanos) que participan en el dominio del problema. Para ello, el método de desarrollo debe proveer guías de identificación que según las características de las entidades puedan orientar la manera

más conveniente o adecuada para modelarlas. Una vez identificadas las clases presentes en el dominio de la aplicación se deben modelar las relaciones que existen entre ellas. Podemos decir que existen básicamente dos tipos de relaciones: estáticas y dinámicas.

La primera captura aspectos estructurales y evidencia la arquitectura del sistema en términos de agentes, objetos y usuarios humanos y la relación estática entre estos; el modelado de estas relaciones nos proporciona una *vista estructural abstracta* del sistema. Para lograr capturar las relaciones estáticas y estructurales en forma precisa es necesario que el método de desarrollo incluya formalismos y técnicas que permitan especificar relaciones de jerarquía (herencia), dependencia semántica (asociación) y parte-de (agregación).

La otra de tipo dinámica que evidencia la habilidad social como característica de las entidades agentes. Dicha habilidad social implica la necesidad de interacción en forma de comunicación entre agentes, y entre agentes y objetos y/o usuarios humanos. Llamaremos colaboración, a la interacción en la cual las entidades implicadas son agentes; y coordinación, a aquella en la que interactúan agentes y usuarios humanos. El modelado de las relaciones de colaboración y coordinación constituye la *vista de comunicación*.

2.2.2 Modelado de las características de los agentes

Además de la identificación de las clases, el modelado de la arquitectura del sistema y de las interacción dinámicas entre las clases del sistema, el método de desarrollo adoptado debería permitir especificar una tercera vista más detallada en término de características internas de las clases agentes y no agentes identificadas en la vista estructural abstracta. Es decir, el método debería representar la arquitectura interna de cada clase.

En este trabajo, nos ocuparemos de analizar únicamente las características necesarias para el modelado de agentes, ya que el modelado de objetos y usuarios del sistema (actores) puede ser abordado utilizando técnicas orientadas a objetos ampliamente aceptadas. El análisis de las características lo realizaremos utilizando como marco referencial la definición de agente propuesta por Wooldridge y presentada en la sección 2.

En la definición de agentes, hemos observado que los agentes están situados en un ambiente, y son capaces de percibir este ambiente a través de sensores de algún tipo. Por tanto los agentes deben poseer *información* acerca de su ambiente. Esto nos conduce al siguiente requerimiento: el marco de especificación de agentes debe ser capaz de representar tanto el estado del ambiente en sí (vista estructural abstracta) como la información que cada agente tiene sobre este ambiente, sus *creencias*.

Un agente autónomo es uno que es capaz y está autorizado a trabajar (razonar, actuar y reaccionar) de forma independiente, en lugar de ser dirigido por otros agentes. El marco de desarrollo de agentes debe proveer la facilidad de especificar el hecho que los agentes operan de forma activa, sin la directa intervención de los humanos u otros agentes y con cierto grado de dominio sobre su flujo de control.

Consideremos ahora la noción de reactividad. Los sistemas reactivos a diferencia de los funcionales (sistemas que simplemente reciben alguna entrada, realizan algún tipo de computación sobre ésta y eventualmente producen alguna salida), no finalizan, sino que mantienen una interacción constante con su ambiente, lo que hace imposible utilizar formalismos que empleen pre- y post- condiciones para razonar sobre sus características. El siguiente requerimiento para el marco de especificación de agentes es que debe ser capaz de representar la naturaleza reactiva inherente de los agentes y sistemas

multiagentes en general. Un formalismo útil para la especificación de esta característica podría ser la lógica temporal o el álgebra de procesos.

Los agentes afectan su ambiente en lugar de dejar en forma pasiva que su ambiente les afecte. En este sentido podríamos decir que los agentes poseen dos tipos de atributos: los atributos de información (creencias y conocimiento) que están relacionados con lo que el agente conoce del mundo que ocupa; y los atributos pro-activos (deseos, intenciones, obligaciones, acuerdos, elecciones, etc.) que son aquellos que en cierta medida guían las acciones del agente. No existe un consenso claro tanto en la comunidad AI o filosófica acerca de precisamente qué combinación de atributos de información y pro-atributos son más adecuados para caracterizar a los agentes [WOOLD95]. Un enfoque que consideramos particularmente interesante y que adoptamos en este trabajo, [KINNY96], afirma que son necesarias: creencias, deseos e intenciones (Beliefs, Desires and Intentions - BDI) [KINNY96]; mientras que otros defienden otras combinaciones. Las creencias representan hechos sobre el ambiente del agente, se distinguen de los conocimientos en el sentido que los conocimientos deben ser siempre verdaderos en cambio un agente puede creer algo falso. Los deseos representan los objetivos del agente, que son estados que un agente desea alcanzar, verificar o mantener. Las intenciones otorgan deliberación al agente, podríamos considerar que el comportamiento pro-activo de los agentes está representado en términos de una librería de planes, el agente selecciona un plan de la librería sobre la base de los objetivos que desea satisfacer. Un plan se instancia cuando ocurre el disparo de un evento que satisface su invocación y condiciones de contexto. Un plan instanciado es una intención. El cuerpo de un plan es un conjunto de tareas que pueden ser sub-objetivos, acciones, aserciones de la base de creencias, y consultas y mensajes a otros agentes. Cuando se forma una intención, estas tareas son ejecutadas por el agente en un esfuerzo por alcanzar un objetivo dado. La racionalidad, por su parte, es la asunción que un agente actuará a fin de lograr sus objetivos, y no lo hará de una manera tal que prevenga que sus objetivos sean alcanzados – al menos en la medida que sus creencias lo permitan. El marco de desarrollo de sistemas basados en agentes debe cubrir el modelado de todas estas características.

Finalmente, cabe destacar que no todos los agentes necesitan tener movilidad. En efecto, un agente puede simplemente comunicarse con su ambiente por medios convencionales. Esto incluye varias formas de llamada a procesos remotos y mensajes [LANGE98]. Sin embargo, consideramos la movilidad (que no es una característica presente en la definición de Wooldridge) una característica interesante de los agentes, particularmente de aquellos orientados a Web que constituyen una parte importante de los sistemas existentes y que se encuentran en etapas de rápida evolución. Un agente móvil no está limitado al sistema donde inicia su ejecución. Es libre de viajar a través de los nodos (hosts) de una red de computadoras. Creado en un ambiente de ejecución, puede transportar su estado y código a otro ambiente de la red, donde reanuda la ejecución. Los agentes móviles pueden ser interesantes porque, entre otras cosas:

- Reducen la sobrecarga de la red: los agentes móviles permiten empaquetar el intercambio de mensajes y enviarlos al destino donde se lleva a cabo la interacción en forma local.
- Encapsulan protocolos: cuando se intercambian datos en un sistema distribuido, cada nodo posee el código que implementa el protocolo necesario para interpretar los mensajes intercambiados. Cuando los protocolos evolucionan, también se necesita actualizar el código que lo implementa. En cambio, los agentes móviles, son capaces de moverse a los nodos estableciendo canales de comunicación basados en protocolos propios.

El marco de especificación de agentes debe proveer la facilidad para especificar los posibles ambientes de ejecución a los que puede migrar el agente y cuáles serían las eventuales restricciones o permisos de acceso necesarios.

Dada la especificación, se debe implementar un sistema software que sea correcto con respecto a dicha especificación. El siguiente paso por tanto es traducir la especificación abstracta del sistema a un modelo computacional concreto. El método de desarrollo de sistemas basados en agentes debe asistir al ingeniero de software en esta etapa del proceso de construcción, para ello puede incluir guías para el refinamiento manual de la especificación a un modelo ejecutable utilizando algún proceso de refinamiento informal; ejecutar o animar directamente la especificación abstracta; o, traducir la especificación a un modelo computacional empleando alguna técnica de traducción automática.

2.3 Métodos orientados a agentes

En esta sección presentamos un breve resumen del estado del arte de los métodos de desarrollo de sistemas basados en agentes.

Como lo afirma Iglesias, en [IGLESI97], existen dos enfoques principales que siguen los métodos orientadas a agentes:

?? *extensión de métodos orientadas a objetos*, a fin de incluir aspectos relevantes de los agentes. Podemos citar varias razones que justifican la extensión de las métodos orientadas a objetos. Entre estas tenemos: existen similitudes entre el paradigma orientado a objetos y el paradigma orientado a agentes; el uso común de los lenguajes orientados a objetos para la implementación de los agentes; y la popularidad de los métodos orientadas a objetos.
Ejemplos de este enfoque son: Análisis y Diseño Orientado a Agentes [BURMEIS96], Técnicas de Modelado de Agentes para Sistemas de agentes BDI [KINNY96], Método para Multi-Agentes basado en Escenarios [MOULIN97], Metodología Orientada a agentes para el Modelado de Empresas [KENDAL96], Ingeniería de Sistemas Multi-Agentes [DELOAC95], Una metodología a nivel de organización para el diseño de multi-agentes [GUTKNE95], entre otros.

?? *extensión de métodos de ingeniería del conocimiento* (KE, Knowledge Engineering), tratando de aprovechar las técnicas de la KE para modelar características cognitivas de los agentes. Según [GLASER96] los métodos de ingeniería del conocimiento pueden proveer una buena base para el modelado de sistemas multi-agentes, dado que ellos se ocupan del desarrollo de sistemas basados en el conocimiento. Y como los agentes poseen características cognitivas, estos métodos pueden proveer las técnicas para el modelado de estos agentes. Ejemplos de este enfoque son: CoMoMAS [GLASER96], MAS-CommonKADS [IGLESI97].

3 Métodos de desarrollo automático de aplicaciones OO: OO-Method

En esta sección presentamos el método de producción automática de software OO-Method. Este método proporciona un enfoque metódico que abarca la construcción del modelo conceptual y su representación en un entorno de desarrollo comercial como un producto de software de calidad.

El proceso de desarrollo que propone OO-Method consiste en recoger las propiedades esenciales del sistema a desarrollar (modelo conceptual) por parte del ingeniero de software y construir de forma automática en cualquier momento (mediante un proceso de conversión gráfico-textual) la especificación formal orientada a objetos en OASIS [PAS92][PAS95][LET98] que constituirá un repositorio de alto nivel del sistema. Además, mediante la definición de un preciso modelo de ejecución, que incluye una estrategia de generación de código (que define cómo se corresponden ciertos patrones de análisis con patrones de diseño e implementación en un lenguaje de programación específico), permite la construcción de un producto software que incluye todos los aspectos estáticos y

dinámicos recogidos en el modelo conceptual. Podemos decir que el modelo de ejecución representa cómo los conceptos recogidos en el modelo conceptual son representados en un entorno computacional.

La propuesta OO-Method está basada en los siguientes principios: dar soporte a las nociones del modelado conceptual orientado a objetos, integrar los métodos formales con métodos convencionales de aceptación industrial, proporcionar un entorno de producción de software avanzado que incluya la generación completa de código (estática y dinámica) en entornos de desarrollo comerciales.

OO-Method aborda la tarea de construcción de software utilizando los siguientes modelos:

1. Modelo Conceptual, que se posiciona en el espacio del problema y para cuya elaboración se proporciona un lenguaje gráfico con la expresividad necesaria para representar adecuadamente los conceptos del modelo OO usado como soporte formal. El Modelo Conceptual permite recoger las propiedades estáticas y dinámicas de los requisitos del sistema en desarrollo. El problema a este nivel consiste en obtener una definición precisa del sistema independientemente de criterios de implementación. Debido a la correspondencia existente entre estos elementos gráficos y el lenguaje de especificación subyacente, una Especificación Formal del sistema en el lenguaje OASIS se puede obtener en cualquier momento. Esta especificación formal actúa, como se ha dicho anteriormente, como repositorio de alto nivel del sistema. Para captar toda esta información OO-Method utiliza tres modelos que describen una Sociedad de Objetos desde tres puntos de vista complementarios. Cada modelo, proporciona una serie de técnicas (con sus correspondientes diagramas) para introducir la información relevante. En concreto tenemos los siguiente modelos:
    a. Modelo de Objetos: define la estructura y relaciones estáticas de las clases identificadas en el dominio del problema.
    b. Modelo Dinámico: define las secuencias posibles de servicios (vidas posibles) y los aspectos relacionados con la interacción entre objetos.
    c. Modelo Funcional: captura la semántica asociada a los cambios de estado de los objetos motivados por la ocurrencia de eventos.
2. El Modelo de Ejecución está situado en el espacio de la solución y fija los detalles para la implementación de una representación concreta del Modelo Conceptual en un entorno de desarrollo. Para ello utiliza una estrategia de generación de código de acuerdo a ciertos patrones específicos de diseño e implementación: control de acceso, interfaz de usuario, persistencia de objetos, etc. Cada elemento del modelo conceptual tienen su correspondiente representación en el lenguaje de programación elegido.

4 OO-Method como método de desarrollo de sistemas basados en agentes.

En la sección 2 indicamos cuáles serían los requisitos de un método de desarrollo de software para el modelado de sistemas en los cuales la abstracción principal es un agente. En esta sección presentamos un análisis de OO-Method considerando dichos requisitos.

OO-Method incluye como concepto el agente pero con una semántica muy limitada y sencilla con respecto a la definición que hemos adoptado. En OO-Method las clases tienen una doble perspectiva cliente/servidor, como servidora una clase oferta servicios, mientras que como cliente, los objetos de la clase actúan como activadores de servicios. Esta perspectiva cliente es la asociada al concepto de agente de OO-Method, por tanto un agente para este método es un objeto activador de los servicios ofrecidos por las clases de la sociedad. En cambio, a pesar de esta limitación, el método incluye una

serie de formalismos interesantes que permiten el modelado de algunas de las características analizadas en la sección 2.2.

Consideremos el modelado de la vista *estructural abstracta*, OO-Method cubre la identificación de objetos y usuarios humanos dentro del dominio del problema (Modelo de Objetos); sin embargo, denomina agentes a todas aquellas clases activadoras de servicios, sin contar (como era de esperarse al ser un método orientado a objetos) de guías para determinar si estas entidades constituyen o no agentes según la definición de Wooldridge. En el Modelo de Objetos es posible modelar la estructura estática del dominio de aplicación mediante las relaciones de herencia y agregación, entre las entidades identificadas.

Hemos señalado (sección 2.2) que en la vista *de comunicación* es necesario modelar colaboración (interacción agente-agente) y coordinación (interacción agente-usuario humano). OO-Method incluye en su Modelo Conceptual un Modelo Dinámico en el cual es posible modelar los aspectos del sistema referentes al control, a las posibles secuencias de eventos que pueden ocurrir en la vida de los objetos, y la interacción entre éstos. Con este modelo es posible especificar tanto colaboración como coordinación para sistemas basados en agentes. La comunicación entre agentes se puede especificar mediante el intercambio de mensajes con sintaxis y semántica común a todos los agentes del sistema.

Analizamos a continuación por cada característica, que hemos señalado forma parte de la vista interna de cada agente, las técnicas y formalismos que OO-Method dispone para modelar algunas de ellas y las limitaciones que presenta para otras.

Para modelar autonomía es necesario especificar actividad y control del flujo de ejecución por parte de la propia clase, OO-Method puede especificar que un objeto es capaz de iniciar la ejecución de una interacción, mediante la relación agente o con los disparos (triggers) cuando se cumple una condición determinada. La reactividad puede ser modelada con el álgebra de procesos que OO-Method incluye para la definición del comportamiento de objetos. Las creencias, deseos y planes, de cada agente pueden ser modeladas en forma de objetos, que mantienen la representación y los procedimientos para el manejo de estas nociones (crear, eliminar, modificar, etc.). Las intenciones, que corresponden a los planes instanciados del agente, puede ser modelados utilizando los diagramas de transición de estados, capturando de esta manera la dinámica interna del agente, definible en tiempo de especificación. Finalmente en OO-Method no es posible especificar clases o módulos software que pueden migrar a distintos ambientes de ejecución.

5 Extensión de OO-Method para especificación de agentes: OO-Method for Agents

Como lo indicamos en la sección 2.2. son necesarias tres vistas o modelos para la especificación de sistemas basados en agentes. En esta sección presentamos por cada vista las extensiones a los modelos de OO-Method para poder modelarlas.

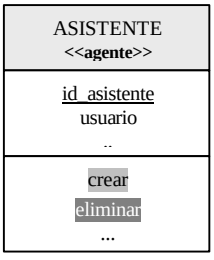

Figura 5.1: Notación Gráfica para una clase agente

Vista *estructural abstracta*:

- Identificación de Agentes: los agentes son las entidades activas del sistema, que pueden cambiar su propio estado y cuyas acciones pueden afectar su ambiente; el ambiente, por su parte, consiste de

elementos pasivos (objetos) cuyos estados cambian sólo por las acciones de los agentes o de los actores.

- Modelo de Objetos: una clase agente, como se muestra en la figura 5.1, se representará gráficamente como una clase, con el estereotipo **<<agente>>** en la cabecera. Se podrán establecer las relaciones (agregación, agente, herencia) convencionales de OO-Method entre las clases del sistema.

Vista de *comunicación*:

- Modelo Dinámico: en el Diagrama de Interacción las instancias de las clases agentes serán representadas en los disparos e interacciones globales por cajas con la etiqueta **<<agente>>**, figuras 5.2a y 5.2b. La sintaxis y semántica de estos diagramas serán las convencionales de OO-Method.

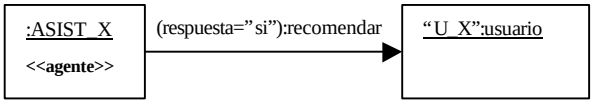

Figura 5.2a: Colaboración

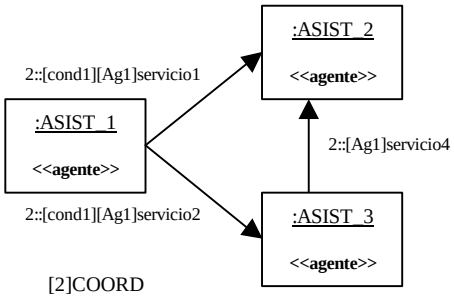

Figura 5.2b: Coordinación

Vista *interna* de cada agente:

- Modelo de Objetos: por cada clase agente declarada se proveerá la definición de las nociones mentales (creencias, deseos e intenciones) del agente.
  - ?? Creencias: serán modeladas por medio de clases componentes de la clase agente. En la figura 5.3, se muestra la estructura de la clase Creencias. Los atributos *a1,...,an*, representarán las creencias, mientras que los operaciones *s1,...,sn*, corresponderán a los métodos para el manejo de estas creencias.

  - ?? Deseos u Objetivos: serán modelados por medio de clases componentes de la clase agente. En la figura 5.4, se muestra la estructura de la clase Objetivos. El atributo *objetivo* será el objetivo propiamente dicho.

  - ?? Intenciones: un plan instanciado es una intención, por tanto en un modelo conceptual es preciso modelar por cada objetivo cuáles son los planes del agente que podrán ser intenciones. Los planes serán clases componentes de la clase Objetivos. Por cada Objetivo podrá existir más de un plan. El cuerpo de un plan (secuencia de eventos, servicios o transacciones) será modelado como una transacción

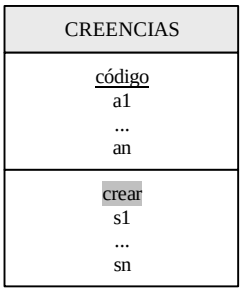

Figura 5.3: Clase Creencias del agente

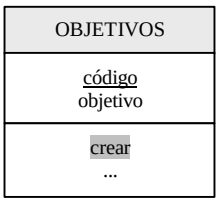

Figura 5.4: Clase Objetivos del agente

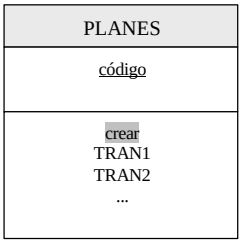

Figura 5.5: Clase Planes del agente

global, que será ofertada como un servicio de la clase Planes. La ejecución de un plan específico está determinada por el valor de verdad de la condición asociada al plan. Este hecho será modelado por medio de Disparos que ejecutan la transacción, según la condición asociada.

En resumen, la estructura resultante de la declaración de una clase agente en el Modelo de Objetos de OO-Method será la que se presenta en la figura 5.6.

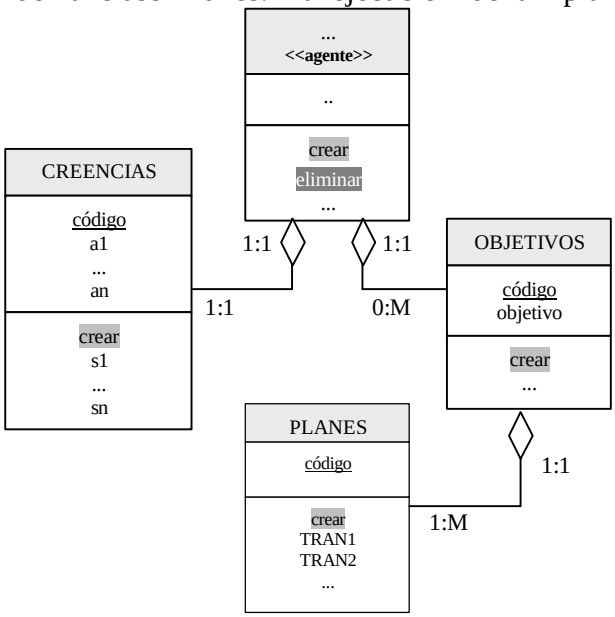

Figura 5.6: Estructura de las Nociones Mentales de la clase agente

- Incluir un Diagrama de Procesadores: para poder especificar movilidad de las clases agentes a distintos procesadores del sistema. En la figura 5.7 se muestra un diagrama de procesadores similar al del AUML (Agent UML) propuesto por Odell [ODELL00]. En el diagrama de procesadores, una caja sombreada es un procesador, una caja sin sombra es un dispositivo o una red, las líneas que unen cajas representan conexión física (cableado), los rectángulos internos a los procesadores constituyen módulos software que representan a los agentes del sistema, el hecho de que un agente pueda migrar a un procesador del sistema se modela por medio de un arco con flecha en los extremos, en la figura por ejemplo, el *asistenteX* puede migrar de un *servidor_x* a cualquier otro servidor, en cambio no lo puede hacer a algún *cliente_y*.

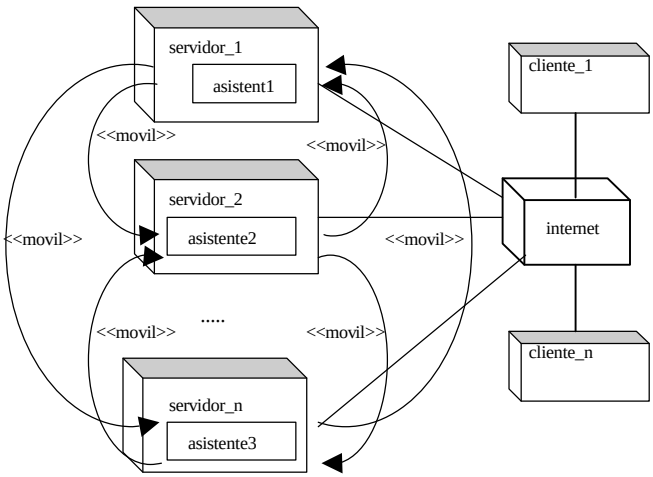

Figura 5.7: Diagrama de Procesadores

Presentamos a continuación un caso de estudio simplificado en el que podremos observar el diagrama de clases del sistema. Utilizaremos como ejemplo un sistema simplificado de Alquiler de Coches. El alquiler de un coche implica la creación de un contrato con el cliente y recoge los datos del alquiler.

Las clases identificadas son: Vehículo, Cliente, Contrato. Usuario humano: Usuario, empleado de la agencia de alquiler. Clase agente: Alquilador, agente encargado de encontrar la mejor opción de alquiler de vehículo (en cuanto a preferencias del cliente, por ejemplo color, marca, kilometraje, tamaño, etc.) para un cliente determinado.

El diagrama de clases resultante es el que se muestra en la figura 5.8. En ésta, se puede notar que la clase agente Alquilador es un compuesto de Creencias y Objetivos y que cada Objetivo a su vez está compuesto por Planes. Las Creencias del Alquilador representa el conocimiento de éste sobre los objetos de su entorno, que en este caso son Vehículo y Cliente. La clase Usuario es el cliente del servicio *alquilar()* que una vez invocado hace que el Alquilador ponga en marcha el plan correspondiente. El plan para alquilar un vehículo a partir de los datos que proporciona la clase Usuario, está representado por TRANS1 que está definida como la secuencia de los siguientes servicios: Planes.Objetivo.Alquilador.bus_pref.(), busca según el contexto del agente las preferencias

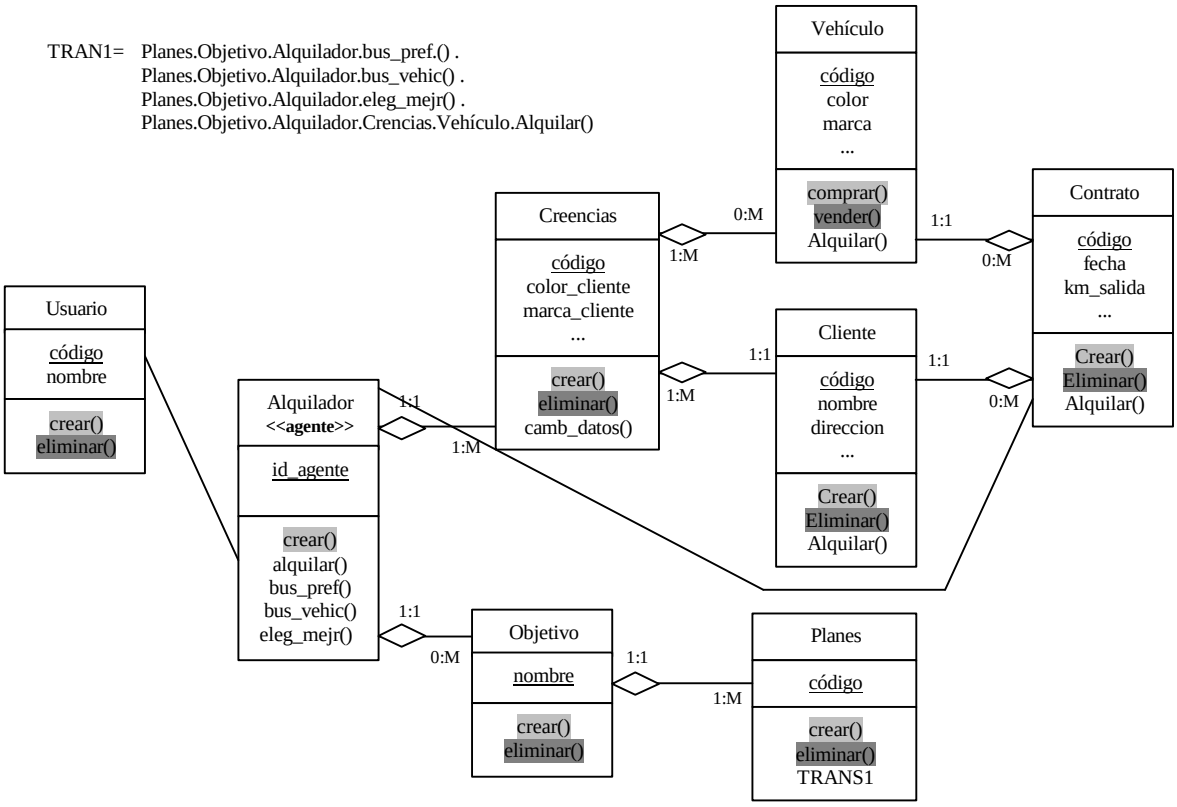

Figura 5.8: Diagrama de Clases del Sistema Alquiler de Coches

del cliente; Planes.Objetivo.Alquilador.bus_vehic(), una vez que obtiene los datos de las preferencias del cliente y mediante un sondeo de los vehículos disponibles obtiene una lista de los que podrían interesar al cliente; Planes.Objetivo.Alquilador.eleg_mejr(), con la lista de vehículos de probable elección determina el mejor de esta lista según las preferencias del cliente; Planes.Objetivo.Alquilador.Crencias.Vehículo.Alquilar() finalmente ejecuta el servicio *alquilar()*, que crea un Contrato de alquiler.

6 Conclusiones

El diseño de sistemas basados en agentes es una actividad compleja que incluye el modelado de módulos software tanto activos como pro-activos.

Técnicas, modelos y métodos de diseño específicos pueden proveer un soporte conceptual importante para la construcción de los sistemas basados en agentes. En este artículo hemos analizado las

propiedades que necesitan ser modeladas para el desarrollo de estos sistemas; en este sentido hemos definido tres vistas. La primera, una vista *estructural abstracta*, para el modelado de la arquitectura conceptual del sistema en términos de agentes, objetos y usuarios humanos, y la relación estructural entre estos. La segunda, una vista de *comunicación*, para el modelado de las relaciones de comunicación entre las clases del sistema, en esta vista hemos definido dos tipos fundamentales de relaciones, la colaboración que es aquella en la que los interlocutores son agentes, y la coordinación en la cual intervienen agentes y usuarios humanos. Y la tercera y última vista, que provee la descripción *interna* de cada agente del sistema, vista en la cual se modela la estructura del agente como una entidad autónoma, reactiva, pro-activa y racional, con creencias, objetivos y planes como nociones mentales .

En este artículo, hemos presentamos, además, un análisis de la expresividad de OO-Method para el modelado de agentes. De este análisis concluimos que OO-Method constituye un soporte conceptual poderoso para sistemas de agentes, al incluir formalismos interesantes para el modelado de algunas de las características de agentes, y al integrar métodos formales con métodos convencionales de aceptación industrial. En cambio OO-Method necesita de extensiones para abarcar de manera eficaz el modelado de las distintas vistas que hemos definido como necesarias en la especificación de agentes. Con este fin, hemos propuesto OO-Method for Agents, como una extensión para sistemas basados en agentes.

Presentamos en la tabla 1, un resumen comparativo entre OO-Method for Agents y los métodos Orientadas a Agentes MAS-CommonKADS [IGLESI97] y la Técnica de Modelado de Agentes para Sistemas de Agentes BDI [KINNY96], teniendo en cuenta qué características de agentes son abarcadas por cada método. En esta tabla podemos ver que OO-Method for Agent abarca la totalidad de las características de agentes analizadas en este artículo a diferencia de los métodos orientados a agentes analizados.

Característica	OO-Method for Agents	MAS-CommnonKADS	Técnica para agentes BDI
Autonomía	?	?	?
Reactividad	?	?	?
Pro-actividad	?	?	?
Sociabilidad	?		?
Racionalidad	?		
Nociones Mentales	?		?

Tabla 1: Resumen comparativo de OO-Method for Agents y dos métodos Orientadas a Agentes

Este trabajo constituye la presentación de la propuesta OO-Method for Agents. Como trabajos futuros para lograr un método de desarrollo de sistemas basados en agentes, potente y completo, se deberán seguir estudiando las extensiones introducidas, así como nuevas extensiones para el modelado de todas las características de sistemas de agentes; será necesario corresponder las extensiones incluidas al método a representaciones en OASIS; definir los módulos que implementen la traducción del Modelo Conceptual a módulos software ejecutables en el espacio de la solución, para lograr que cada elemento tenga su correspondiente representación en el lenguaje de programación elegido.

7 Referencias

[BOOCH94] Booch,G ., "Object Oriented Analysis and Design with Applications", The Benjamin/Cummings Publishing Company, 1994.

[BURMEIS96] Burmeister, B., "Models and Methodology for agent-oriented analysis and design". In K. Fischer, editor, Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems, DFKI Document D-96-06, 1996.

[DELOAC95] DeLoach S. "A. Multiagent Systems Engineering: A Methodology and Language for Designing Agent Systems", 1995.

[GLASER96] Glaser,N. "Contribution to Knowledge Modelling in a Multi-Agent Framework (the CoMoMAS Approach)". PhD thesis, L'Universitité Henri Poincaré, Nancy I, France, November 1996.

[GRAESS96] Graesser, A. and Franklin S., "Is it an Agent, or just a Program?: A taxonomy for Autonomous Agents". Proceedings of the 3º International Workshop on Agent Theories, Architectures and Lenguages, Springer-Verlag, 1996.

[GUTKNE95] Gutknecht,O. and Jacques,F.,"An organizational-level methodology for multi-agent design". 1995.

[IGLESI97] Iglesias, C. A, M., González, J. C., and Velasco, J. R., "Analysis and design of multiagent systems using MAS-CommonKADS". In AAAI'97 Workshop on Agent Theories, Architectures and Languages, Providence, RI, ATAL, July 1997.

[JACOB97] Jacobson, I., Christerson, M., Jonsson, P., and Övergaard, G., "Object-Oriented Software Engineering". A Use Case Driven Approach, ACM Press, 1997.

[KENDAL96] Kendall,E., Malkoun,M.T., and Jiang,C., "A methodology for developing agent based systems for enterprise integration" .In D.Luckose and Zhang C.,editors, Proceedings of the 1º Australian Workshop on DAI, Lecture Notes on AI. Springer-Verlag: Heidelberg, German, 1996.

[KINNY96] Kinny, D., Georgeff, M., adn Rao, A., "A methodology and modelling technique for system of BDI ageNts". In W. Van der Velde and J. Perram, editors, Agents Breaking Away: Proceedings of the MAAMAW'96. Springer-Verlag: Heidelberg, Germany, 1996.

[LANGE98] Lange,D."Programming and Deploying Java Mobile Agents with Aglets". Addison-Wesley, 1998.

[LET98] Letelier P., Sánchez P., Ramos I., and Pastor O. "OASIS 3.0: Un enfoque formal para el modelado conceptual orientado a objetos". Servicio de Publicaciones Universidad Politécnica de Valencia, Valencia, España, 1998.

[MOULIN97] Moulin, B. and Cloutier, L., "Collaborative work based on multiagent architecture: a methodological perspective". In Fred Aminzadeh and Mohammad Jamshidi, editors, Soft Computing: Fuzzy Logic, Neural Networks and Distributed Artificial Intelligence. Springer-Verlag: Heidelberg, Germany, 1997.

[ODELL00] Odell, J. and Bock C., "Suggested UML Extensions for Agents," (Response to the OMG Analysis and Design Task Force UML RTF 2.0 Request for Information), 2000.

[PAS92] Pastor O. "Diseño y Desarrollo de un Entorno de Producción de Producción Automática de Software Basado en el Modelo Orientado a Objetos. PhD thesis, DSIC – Universidad Politécnica de Valencia, Valencia, España, Mayo 1992. Supervisor: Dr. Isidro Ramos Salavert.

[PAS95] Pastor O. and Ramos I. "OASIS versión 2 (2.2): A Class-Definition Language to Model Information Systems". Servicio de Publicaciones Universidad Politécnica de Valencia, Valencia, España, 1995.

[RUMB91] Rumbaugh,J.,Blaha,M.,Premerlani,W.,Eddy,F.,and Lorensen,V.,"Object-Oriented Modeling and Design". Prentice-Hall, 1991.

[SHOHAN93] Shoham, Y., "Agent-oriented programming". Artificiall Intelligence,Vol 60, Nº1,pp 51-92, March 1993.

[WOOLD95] Wooldridge,M. and Jennings N,R.,"Agent Theories, Architectures, and Languages: a Survey",in Wooldridge and Jennings Eds., Intelligence Agents, Berlin: Springer-Verlag,Vol. 1,Nº 22, 1995.