Metodologías para el desarrollo de sistemas multi-agente

Jorge J. Gómez Sanz

Departamento de Sistemas Informáticos y Programación
Facultad de Informática, Universidad Complutense,
Avda. Complutense s/n
Madrid, 28001
jjgomez@sip.ucm.es

Resumen

Apoyándose en los resultados de la investigación en agentes y Sistemas Multi-Agente, surge una línea de trabajo cuyo propósito es consolidar la experiencia adquirida en forma de metodologías. Estas metodologías proporcionan medios para construir Sistemas Multi-Agente de forma disciplinada y repetible. En este trabajo se presenta un resumen de lo que proponen las metodologías actuales. Ante la imposibilidad de revisar todas y cada una de las existentes, se ha optado por seleccionar un conjunto significativo cercano a las prácticas de ingeniería del software. La selección atiende a la presencia de un proceso de desarrollo, el combinar diferentes vistas para describir el sistema, e incorporar elementos asociados al área de los agentes. Como resultado, se tienen siete metodologías relevantes. A lo largo del artículo, se estudian las aportaciones de estas metodologías cara a un desarrollo.

Palabras clave: metodologías, agentes, sistemas multi-agente, ingeniería del software

1 Introducción

En la última década ha cobrado fuerza la hipótesis de que si los programas que componen los sistemas distribuidos fueran inteligentes, si pudiesen reorganizar sus funciones para corregir o tolerar los fallos y si su coordinación pudiese estructurarse en términos intuitivos, el diseño y el mantenimiento de dichos sistemas sería más fácil de elaborar, más adaptable y más fiable.

En la construcción de estos programas se está aplicando una tecnología íntimamente relacionada con la inteligencia artificial. Se trata de los *agentes* que, de momento, se entenderán como programas autónomos e inteligentes. Bajo el punto de vista de esta tecnología, los sistemas distribuidos pasan a ser *Sistemas Multi-Agente* (SMA). Los SMA se desarrollan sobre *middleware*[1] y proporcionan un nuevo nivel de abstracción más intuitivo. El diseño de SMA, generalmente, se aborda pensando en los agentes como entes con motivación. En lugar de modelar un sistema con componentes que ejecutan métodos, el desarrollador tiene que pensar en los objetivos que los componentes deben alcanzar y en las tareas necesarias para que lo consigan. Al desarrollar los componentes así, se espera que el proceso sea más intuitivo ya que esta forma de modelar y de razonar se halla más cerca del pensamiento humano que los paradigmas de programación tradicionales [Bratman 87].

La construcción de SMA integra tecnologías de distintas áreas de conocimiento: técnicas de *ingeniería del software* para estructurar el proceso de desarrollo; técnicas de *inteligencia artificial* para dotar a los programas de capacidad para tratar situaciones imprevistas y tomar decisiones, y *programación concurrente y distribuida* para tratar la coordinación de tareas ejecutadas en diferentes máquinas bajo diferentes políticas de planificación. Debido a esta combinación de tecnologías, el desarrollo de SMA se complica.

Existen plataformas de desarrollo que dan soluciones parciales al modelado de comportamiento y a la coordinación de agentes. El rango de estas soluciones va desde proporcionar servicios básicos (gestión de agentes, librerías de algoritmos, localización de agentes o movilidad), como JADE [Bellifemine et al. 01], Grasshopper [Baümer et al. 00] o ABLE [IBM 03], hasta entornos de desarrollo donde se parametrizan armazones (*framework*) software, como ZEUS [Nwana et al. 99] o AgenTool [Wood et al. 00].

Aunque facilitan el proceso, las plataformas de desarrollo quedan incompletas sin un proceso de desarrollo de software especializado para agentes que haga similar la creación de SMA a la producción de software convencional. Las técnicas convencionales de ingeniería (*Unified Process* [Jacobson et al. 00], CommonKADS [Tansley et al. 93]) no tienen en cuenta las necesidades de especificación de los SMA, como la especificación de planificación de tareas, intercambio de información con lenguajes de comunicación orientados a agentes, movilidad del código o motivación de los componentes del sistema. Por ello, se plantean nuevas metodologías basadas en agentes (BDI [Kinny et al. 97], *Vowel Engineering* [Ricordel 01], MAS-CommonKADS [Iglesias 98][Iglesias et al. 98b], GAIA [Wooldridge et al. 00]). Estas metodologías parten de un modelo, informal en la mayoría de casos, de cómo debe ser un SMA y dan guías para su construcción. En las primeras metodologías, las guías consistían en una lista breve de pasos a seguir. Las más modernas, aunque han progresado en la integración con la ingeniería del software clásica, aún no muestran la madurez que se puede encontrar en metodologías convencionales como el *Unified Process*. El motivo principal es que siguen faltando herramientas de soporte y un lenguaje para la especificación del SMA que permitan trabajar de forma similar a como se trabaja en Rational Rose, TogetherJ o Paradigm+.

De entre las metodologías existentes, se ha seleccionado un conjunto utilizando tres criterios: utilización de diferentes vistas para la especificación del sistema, incorporar la idea de proceso de desarrollo, e integrar técnicas de ingeniería y teoría de agentes. De acuerdo con estos criterios, se han identificado siete metodologías. La primera es la ingeniería de vocales (*vowel engineering*) [Demazeau 95] que fue una de las primeras en considerar diferentes aspectos (agentes, entorno, interacciones y organización) en el desarrollo de SMA. La segunda es MAS-CommonKADS [Iglesias 98][Iglesias et al. 98b] que, debido a su origen (CommonKADS [Tansley et al. 93]), se trata de una metodología orientada al desarrollo utilizando experiencia en sistemas expertos. A continuación el diseño basado en BDI [Kinny et al. 97] que ha influido notablemente en la forma de concebir el control de los agentes. Después, se estudian dos metodologías soportadas por herramientas: ZEUS [Nwana et al. 99] y MaSE

[1] *Se denomina middleware a software cuyo cometido es unir o mediar entre varios programas aislados*

[DeLoach 01]. Seguidamente, se estudia GAIA [Wooldridge al. 00], de gran influencia, que estudia la definición de vistas en una metodología y trata de integrarse en un ciclo de vida de software tipo cascada. Por último, INGENIAS [Gomez et al. 02][Gomez 02], creada a partir del trabajo de MESSAGE [Caire et al. 02] en la que el autor de este trabajo participó activamente.

Tras la presentación de estas metodologías, se presenta una sección para orientar en la elección de una metodología, otra en la que se dan referencias de trabajos en la evaluación de metodologías y una última orientada a proporcionar más información y otros puntos de vista acerca de las metodologías existentes.

2 Vowel Engineering

Se trata de la metodología seguida en el grupo MAGMA [MAGMA 03]. El término *vowel engineering* viene de que el sistema final depende de la ordenación y agrupamiento de aspectos identificados por cuatro vocales: A (por agentes), E(por entorno), I (por interacciones) y O (por organización). Para cada aspecto, se utilizan componentes y técnicas específicas. Para representar agentes se u desde simples autómatas hasta complejos sistemas basados en conocimiento. La forma de ver las interacciones van desde modelos físicos (propagación de onda en el medio físico) hasta los actos del habla (*speech acts*). Las organizaciones van desde aquellas inspiradas en modelos biológicos hasta las gobernadas por leyes sociales basadas en modelos sociológicos.

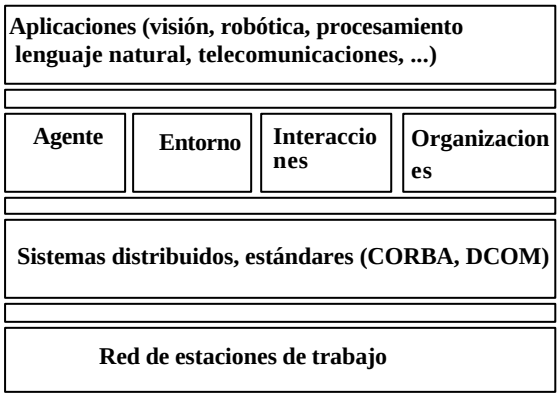

Figura 1. Arquitectura de capas del Sistema Multi-Agente en *Vowel Engineering*

El propósito de *Vowel engineering* es lograr librerías de componentes que den soluciones al diseño de cada uno de estos aspectos, para que posteriormente, el diseñador seleccione un modelo de agente, un modelo de entorno, un modelo de interacciones y modelos de organización a instanciar (ver figura 1).

Como ejemplo, [Demazeau 95] propone para aspectos de interacción un lenguaje para la descripción de protocolos de interacción basados en procesos de comunicación síncronos o asíncronos donde la semántica es muy similar a la de los actos del habla. La representación en sí se hace mediante redes de transición en las que los arcos se corresponden con los mensajes intercambiados y los estados reflejan la situación global (i.e. no hay posibilidad de que un estado se refiera al estado de un agente concreto).

El proceso de desarrollo consiste en tener en cuenta los elementos señalados por las vocales en un cierto orden. El orden se decide en función del tipo de sistema que queramos tener. Por ejemplo, si se empieza por la vocal O, se tendrá un sistema en el que las relaciones sociales son lo más importante. Si se empieza por la A, se tendrá un sistema en el que probablemente la organización surja como resultado de la interacción de agentes aislados.

Una de las últimas publicaciones acerca de *Vowel Engineering* [Ricordel 01] propone la implementación mediante la plataforma *Volcano*. La plataforma utiliza el ensamblaje de componentes utilizando lenguajes de composición arquitecturas, concretamente UniCon [Carnegie 03]. El desarrollo consiste en ensamblar componentes que pueden ser desarrolladas *ad-hoc* o proceder de una librería. Cada componente pertenece a una categoría concreta de las cuatro consideradas.

Pese a que en la literatura se indica que existe un entorno de desarrollo basado en estas ideas, éste no es público. De todas formas, el trabajo desarrollado dentro de MAGMA con esta metodología es reseñable debido a su variedad en los dominios de aplicación (Sistemas de información geográfica, *Robocup*, simulaciones de mercados o agentes en tiempo real).

2.1 Comentarios

Vowel Engineering ha sido una de las primeras metodologías en modelar sistemas a partir de la combinación de diferentes aspectos. La combinación de aspectos se indicaba inicialmente en forma de instrucciones en lenguaje natural, pero en los últimos trabajos ha evolucionado hacia la utilización de lenguajes de arquitecturas. Como consecuencia de esta forma de desarrollo, se facilita la reutilización de código.

Aunque es prometedor, el trabajo en *Vowel Engineering* aún no está completo ya que faltan herramientas de soporte. Además, no existen instrucciones acerca de cómo describir cada uno de los aspectos considerados.

El proceso de desarrollo es el punto debil de esta metodología. *Vowel Enginering* solo proporciona las vocales como resumen de elementos a considerar en el desarrollo y un conjunto de tecnologías aplicadas.

3 MAS-CommonKADS

Esta metodología extiende CommonKADS [Tansley et al. 93] aplicando ideas de metodologías orientadas a objetos para su aplicación a la producción de SMA [Iglesias 98][Iglesias et al. 98b]. La metodología CommonKADS gira alrededor del modelo de experiencia y está pensada para desarrollar sistemas expertos que interactuen con el usuario. De hecho considera sólo dos agentes básicos: el usuario y el sistema. MAS-CommonKADS extiende los modelos de CommonKADS para tener en cuenta la posibilidad de que dos o más componentes del sistema interactúen.

MAS-CommonKADS ha sido la primera en plantear un desarrollo de SMA integrado con un ciclo de vida de software, concretamente el espiral dirigido por riesgos [Pressman 82]. Propone siete modelos para la definición del sistema: agente, tareas, experiencia, coordinación, comunicación, organización y diseño. Cada modelo presenta referencias a la teoría sobre la que se basa. El modelo en sí parte de una descripción gráfica que luego se complementa con explicaciones en lenguaje natural de cada elemento. Existe por cada modelo una descripción de las dependencias respecto de otros modelos y de las actividades involucradas. Estos modelos se hayan descritos ampliamente en [Iglesias 98] en lenguaje natural, complementándose con otras notaciones como SDL (*Specification and Description Language*) [ITU 99b] o MSC (*Message Sequence Chart*) [ITU 99a] para describir el comportamiento de los agentes cuando interaccionan.

Según la web de MAS-CommonKADS existen herramientas de soporte desarrolladas dentro del proyecto [MIX 99]. Estas herramientas aparecen bajo el epígrafe MAST (*MultiAgent Systems Tool*). Sin embargo, la única que se puede descargar no es una herramienta para apoyo al análisis y diseño, sino un conjunto de arquitecturas y frameworks de agentes.

3.1 Comentarios

Esta metodología ha sido la primera en incorporar la idea de *proceso de ingeniería* en el sentido de Pressman [Pressman 82]. De hecho, describe con bastante detalle cómo se debe definir el sistema teniendo en cuenta las dependencias entre los modelos.

El principal inconveniente de MAS-CommonKADSEn es que el nivel de detalle alcanzado en la descripción no es realizable sin el apoyo de herramientas de soporte. Lo que propone MAS-CommonKADS, entre otras cosas, no es una notación sino una lista detallada de elementos y relaciones a identificar en el sistema. Un desarrollador puede seguir la lista y generar la documentación requerida de forma manual, sin embargo el proceso es demasiado costoso y dado a errores.

En la versión actual de MAS-CommonKADs, la información que hay que obtener se expresa con lenguaje natural. A pesar del apoyo de una herramienta de soporte, al tener la especificación en lenguaje natural, se dificulta el análisis automático de la especificación generada . Así pues, para averiguar si existen elementos no utilizados o si existen contradicciones hay que revisar la documentación a mano. Para lograr lo mismo en MAS-CommonKADS habría que restringir el uso de lenguaje natural o bien incluir formalismos que logren una definición más precisa y menos ambigua del SMA.

A pesar de estos inconvenientes, MAS-CommonKADs constituye un ejemplo a seguir en lo que a metodologías se refiere. Es exhaustiva como pocas a la hora de detallar el sistema y además es consecuente con que el proceso de desarrollo en la mayoría de los casos es más complejo que un conjunto de pasos. De hecho, las ideas de MAS-CommonKADS han estado presentes en casi todos los trabajos referentes a metodologías desde hace años.

4 BDI

Las arquitecturas BDI se inspiran en un modelo cognitivo del ser humano [Bratman 87]. Según esta teoría, los agentes utilizan un modelo del mundo, una representación de cómo se les muestra el entorno. El agente recibe estímulos a través de sensores ubicados en el mundo. Estos estímulos

modifican el modelo del mundo que tiene el agente (representado por un conjunto de creencias). Para guiar sus acciones, el agente tiene *Deseos*. Un *deseo* es un estado que el agente quiere alcanzar a través de *intenciones*. Éstas son acciones especiales que pueden abortarse debido a cambios en el modelo del mundo. Aunque la formulación inicial es de Bratman, fueron Georgeff, Rao y Kinny [Kinny et al. 97] quienes formalizaron este modelo y le dieron aspecto de metodología. De hecho es su trabajo, y no el original de Bratman, lo que aquí se comenta.

Para especificar el sistema de agentes, se emplean un conjunto de modelos que operan a dos niveles de abstracción: externo e interno. Primero, desde un punto de vista externo, un sistema se modela como una jerarquía de herencia de clases de agentes, de la que los agentes individuales son instancias. Las clases de agente se caracterizan por su propósito, sus responsabilidades, los servicios que ejecutan, la información acerca del mundo que necesitan y las interacciones externas. Segundo, desde un punto de vista interno, se emplean un conjunto de modelos (modelos internos) que permiten imponer una estructura sobre el estado de información y motivación de los agentes y las estructuras de control que determinan su comportamiento (creencias, objetivos y planes en este caso).

En esta metodología, la integración con el ciclo de vida de software es reducida. Los autores proponen una serie concreta de pasos para generar los modelos. Estos pasos se repiten haciendo que los modelos, que capturan los resultados del análisis, sean progresivamente elaborados, revisados y refinados. Además, el refinamiento de los modelos internos conlleva la realimentación de los modelos externos. Por ejemplo, el construir los planes y creencias de una clase de agente clarifica qué nivel de detalle se requiere en la representación del mundo que posee el agente.

```
inicializar -estado();
repetir
        opciones:= generadorOpciones(colaEventos);
        opcionesSeleccionadas:=deliberar(opciones);
        actualizarIntenciones(opcionesSeleccionadas);
        ejecutar();
        obtenerNuevosEventosExternos();
        eliminarAccionesTerminadasConExito();
        eliminarAccionesImposiblesDeTerminarConExito();
fin repetir
```

Figura 2. Control interno de los agentes BDI

Se imponen varias restricciones sobre la arquitectura que soporte estos modelos: que asegure que los eventos se responden en su momento, que las creencias se mantengan consistentemente, y que la selección de planes y ejecución se desarrolle de manera que refleje ciertas nociones de racionalidad. El control de cada agente en esta arquitectura sería el expuesto en la figura 2.

4.1 Comentarios

Metodológicamente, la propuesta de Georgeff, Kinny y Rao es consecuente con la dificultad de generar los modelos que proponen. Como ellos admiten, existen dependencias entre los diferentes modelos que dificultan el proceso de generación de los modelos que describen el SMA. Como solución proponen un proceso iterativo e incremental (idea también reflejada en el Proceso Racional Unificado [Jacobson et al. 99]) con realimentaciones. Sin embargo, la forma en que tienen lugar estas realimentaciones no se llega a mostrar con detalle.

Como en las anteriores metodologías, se echa de menos la presencia de herramientas de soporte. Sin embargo, los modelos formales que se pueden encontrar tras esta metodología constituyen una referencia obligada para aquellos interesados en la integración de teorías de agentes y las prácticas de ingeniería. Quizá, lo único comparable sea el trabajo realizado por [Brazier et al. 97] en DESIRE.

5 MaSE

MaSE (Multi-agent systems Software Engineering) [DeLoach 01] parte del paradigma orientado a objetos y asume que un agente es sólo una especialización de un objeto. La especialización consiste en que los agentes se coordinan unos con otros vía conversaciones y actúan proactivamente para alcanzar metas individuales y del sistema.

En MaSE los agentes son sólo una abstracción conveniente, que puede o no poseer inteligencia. En este sentido, los componentes inteligentes y no inteligentes se gestionan igualmente dentro del mismo armazón.

El proceso de desarrollo en MaSE es un conjunto de pasos, la mayoría de los cuales se ejecutan dentro de la herramienta que soporta MaSE, AgentTool [DeLoach 01] (ver figura 3). El análisis en MaSE consta de tres pasos: capturar los objetivos, capturar los casos de uso y refinar roles. Como productos de estas etapas se esperan: diagramas de objetivos, que representan los requisitos funcionales del sistema; diagramas de roles, que identifica roles, tareas

asociadas a roles y comunicaciones entre roles y entre tareas; y casos de uso, mostrados no como diagrama sino como una enumeración de los casos de uso considerados con la posibilidad de usar diagramas de secuencia para detallarlos.

El diseño consta de cuatro pasos: crear clases de agentes, construir conversaciones, ensamblar clases de agentes y diseño del sistema. Como productos de estas etapas, MaSE espera: diagramas de clases de agentes, que enumeran los agentes del sistema, roles jugados e identifican conversaciones entre los mismos;descomposición del sistema (agente) en subsistemas (componentes del agente) e interconexión de los mismos (definición de la arquitectura del agente mediante componentes); diagramas UML de despliegue para indicar cuántos agentes habrá en el sistema y de qué tipo.

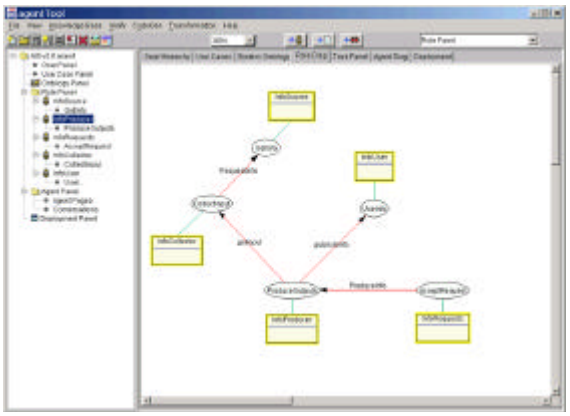

Figura 3. AgentTool, la herramienta de soporte de MaSE

Las comunicaciones entre diferentes elementos (componente-componente, agente-agente, rol-rol, tarea-tarea) se refieren al envío de estímulos desde una máquina de estados a otra. En el caso de las tareas, a estas máquinas las denominan en MaSE *diagramas de tareas concurrentes*.

La herramienta de soporte, agentTool [DeLoach 01], permite generar código automáticamente a partir de la especificación del sistema, que en MaSe es el conjunto final de diagramas. La generación de código es independiente del lenguaje de programación utilizado, ya que se realiza recorriendo las estructuras de datos generadas en la especificación y generando texto como salida.

Dentro de la misma herramienta, MaSE incorpora utilidades para verificar la corrección de los protocolos que utilicen los agentes.

5.1 Comentarios

MaSE usa intensivamente máquinas de estados para especificar el comportamiento de diversos elementos de la especificación. Aunque en MaSe no se menciona, esta idea no es original. Además de su aplicación en UML, se puede revisar la línea de investigación en *Abstract State Machines* (ASM) y el conocido SDL [ITU 99b] que lleva utilizándose en industria desde hace dos décadas. En estas líneas existen multitud de herramientas, métodos de verificación, y desarrollos completos.

De todas formas, y aunque el diseño de sistemas distribuidos basándose en máquinas de estados ya ha sido aceptado industrialmente, no es tan sencillo verificar si con esta técnica se es capaz de expresar elementos característicos de la tecnología de agentes como el razonamiento de los agentes, organización de los agentes o caracterización de su estado mental. De hecho, estos son elementos que esta metodología no considera.

Respeto del proceso de desarrollo, MaSE propone un proceso más simple que el de MAS-CommonKADS. De acuerdo con los artículos publicados, la metodología podría traducirse como *tome la herramienta de soporte y rellene los diferentes apartados.* La ventaja de este enfoque es que se aprende rápido: sólo hay que experimentar con la herramienta. Sin embargo, la trampa de esta forma de modelar es que se obvia que, como en el modelo BDI, se tienen dependencias entre los diagramas propuestos (como entre los diagramas de secuencia y las conversaciones) y que no es tan sencillo el saber qué máquinas de estados definen la ejecución de una tarea en el contexto de una interacción.

En cuanto al proceso de generación de código, la versión actual integra el código de los componentes a generar con el código de la herramienta. Esto es, el código a generar aparece como cadenas de caracteres que se concatenan con información extraída de la especificación. Esto quiere decir que cambios en el código de componentes generados implican la recompilación necesaria de la herramienta completa.

6 ZEUS

ZEUS consta de una herramienta [Nwana et al. 99] (ver figura 4) y una metodología [Collis et al. 99], de forma similar a agenTool y MaSE. Desde su aparición, ZEUS se ha convertido en referencia de cómo debe ser una herramienta para el desarrollo de

SMA. Sobre todo, por la forma en que combinan los distintos resultados de investigación en agentes (planificación, ontologías, asignación de responsabilidades, relaciones sociales entre agentes) en un sistema ya ejecutable. De hecho, la aplicación genera incluso programas para arrancar el sistema especificado e incluye herramientas de monitorización como el *Visor de Sociedad* que muestra los agentes existentes y sus relaciones, la *Herramienta de Control* para ver o modificar remotamente el estado de los agentes y los *Generadores de Informes* para obtener estadísticas de funcionamiento e informes de actuación de la sociedad de agentes.

La metodología ZEUS propone un desarrollo en cuatro etapas [Collis et al. 99]: el análisis del dominio, el diseño de los agentes, la realización de los agentes y el soporte en tiempo de ejecución. Las etapas soportadas por la herramienta son la de *realización de los agentes* y la de *soporte en tiempo de ejecución*. Las etapas anteriores se basan en el uso de roles para analizar el dominio y en su asignación a agentes.

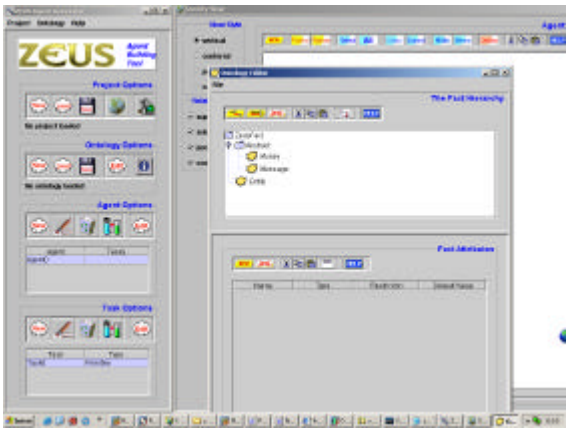

Figura 4. Entorno de desarrollo ZEUS

Los resultados a producir a lo largo del proceso de desarrollo son los siguientes:

- **Análisis del dominio.** Se orienta a obtener el modelo de roles. El modelo de roles se compone de diagramas UML de clases para representar roles, diagramas UML de colaboración para indicar qué mensajes se intercambian y fichas para describir los roles.
- **Diseño del agente.** Consiste en determinar qué necesita cada agente para poder desempeñar su cometido. Esto incluye revisar las tecnologías y disciplinas relacionadas con el diseño de agentes.
- **Implementación de los agentes y soporte en tiempo de ejecución.** Se trata de utilizar la herramienta ZEUS para trasladar los conceptos de diseño dentro de la plataforma que ZEUS tiene ya implementada. El resultado es un modelo ejecutable del sistema.

6.1 Comentarios

Hay poco que comentar acerca de ZEUS. Hasta la aparición de MaSE, ZEUS era herramienta referencia. Hoy en día se encuentran muchos más artículos referenciando MaSE que ZEUS.

Ante la similitud de enfoques, se impone una breve comparativa entre ZEUS y MaSE. Conceptualmente, ZEUS es superior a MaSE. Si bien la primera está más orientada a la aplicación de tecnología de agentes (planificación, definición de ontologías, secuenciación de tareas), la segunda se orienta más a las prácticas de ingeniería convencional.

Metodológicamente, ZEUS es más pobre que MaSE. El modelado de roles, propuesto en [Collis y Ndumu 99], no profundiza en la aplicación de la herramienta dentro del proceso de desarrollo. El ámbito de la metodología se limita a estudiar cómo agrupar la funcionalidad del sistema dentro de cada rol, dejando aparte consideraciones acerca de cómo organizar las tareas, definir las ontologías y las dependencias sociales, aspectos que son modelables dentro de la herramienta.

Detallar todos los elementos requeridos por la herramienta ZEUS requeriría un artículo entero. El uso de esta herramienta no es trivial. Presenta tantas posibilidades que a un desarrollador con pocos conocimientos en agentes puede hacerle dudar. ¿Por dónde empezar? Por ello, se hecha más en falta un proceso de desarrollo más detallado que indique en qué aspectos de la herramienta hay que concentrarse en cada momento.

La cuestión que se plantea al estudiar ZEUS y MaSE es si hay que tener una buena herramienta o una buena metodología. La opinión del autor es que las herramientas de soporte no tienen que condicionar la metodología y para ello, metodología y herramientas han de ser independientes.

7 GAIA

GAIA [Wooldridge et al. 00] [Zambonelly et al. 00] es una metodología para el diseño de sistemas basados en agentes cuyo objetivo es obtener un sistema que maximice alguna medida de calidad

global (no se llega a detallar cuál). GAIA pretende ayudar al analista a ir sistemáticamente desde unos requisitos iniciales a un diseño que, según los autores, esté lo suficientemente detallado como para ser implementado directamente.

En GAIA se entiende que el objetivo del análisis es conseguir comprender el sistema y su estructura sin referenciar ningún aspecto de implementación. Esto se consigue a través de la idea de *organización*. Una organización en GAIA es una colección de roles, los cuales mantienen ciertas relaciones con otros y toman parte en patrones institucionalizados de interacción con otros roles. Los roles agrupan cuatro aspectos: responsabilidades del agente, los recursos que se le permite utilizar, las tareas asociadas e interacciones.

GAIA propone trabajar inicialmente con un análisis a alto nivel. En este análisis se usan dos modelos, *el modelo de roles* para identificar los roles clave en el sistema junto con sus propiedades definitorias y el *modelo de interacciones* que define las interacciones mediante una referencia a un modelo institucionalizado de intercambio de mensajes, como el FIPA-Request [FIPA 03]. Tras esta etapa, se entraría en lo que GAIA considera diseño a alto nivel. El objetivo de este diseño es generar tres modelos: el *modelo de agentes* que define los tipos de agente que existen, cuántas instancias de cada tipo y qué papeles juega cada agente, el m*odelo de servicios* que identifica los *servicios* (funciones del agente) asociados a cada rol, y un *Modelo de conocidos*, que define los enlaces de comunicaciones que existen entre los agentes.

A partir de aquí, los autores de GAIA proponen aplicar técnicas clásicas de diseño orientado a objetos. Sin embargo, GAIA declara que queda fuera de su ámbito. Esta metodología sólo busca especificar cómo una sociedad de agentes colabora para alcanzar los objetivos del sistema, y qué se requiere de cada uno para lograr esto último.

7.1 Comentarios

La principal crítica que se puede hacer a GAIA es que se queda a un nivel de abstracción demasiado alto. Según los autores, con ello se consigue desacoplar GAIA de las distintas soluciones de implementación de agentes. Sin embargo, la utilidad de este desacoplamiento queda por demostrar. Dado el nivel de abstracción en que se queda es de esperar que el esfuerzo a invertir para pasar de una especificación GAIA hasta su implementación sea alto.

Un aspecto discutible es el uso combinado de fórmulas lógicas con fichas de documentación clásicas en ingeniería del software [Pressman 82]. En GAIA se resalta la importancia del uso de fórmulas lógicas sin explicar los inconvenientes. Es cierto que se consigue mayor precisión, pero a costa de: invertir más trabajo, requerir desarrolladores más especializados que sepan comprenderlas, y definir la semántica de los predicados que se están empleando.

Respecto del proceso de desarrollo, GAIA omite las distintas dependencias entre los modelos propuestos, lo cual es fundamental a la hora de proponer un proceso que dé como salida la especificación del sistema.

Para terminar, solo mencionar la falta de herramientas de soporte para esta metodología. De todas las revisadas en este artículo es la que menos soporte tiene, lo cual sorprende dado el impacto que ha tenido.

8 INGENIAS

Parte del trabajo de MESSAGE [Caire et al. 02], en el que participó el autor de este artículo. MESSAGE es una metodología que ha tenido un gran impacto en la comunidad dedicada al estudio de los agente software. Como muestra de esta influencia, está la metodología que se presenta en esta sección, INGENIAS [Gomez et al. 02], y otra que aplica las ideas de MESSAGE al desarrollo de sistemas en tiempo real, llamada RT-MESSAGE [Inglada 03]. El motivo de presentar INGENIAS en lugar de MESSAGE, se debe a que el autor considera INGENIAS como evolución de las ideas de MESSAGE. INGENIAS profundiza en los elementos mostrados en el método de especificación, en el proceso de desarrollo, además de incorporar nuevas herramientas de soporte y ejemplos de desarrollo.

INGENIAS, como MESSAGE, define un conjunto de meta-modelos (una descripción de alto nivel de qué elementos tiene un modelo) con los que hay que describir el sistema. Los meta-modelos indican qué hace falta para describir: agentes aislados, organizaciones de agentes, el entorno, interacciones entre agentes o roles, tareas y objetivos. Estos meta-modelos se construyen mediante un lenguaje de meta-modelado, el GOPRR (*Graph, Object, Property, Relationship, and Role*) [Lyytinen et al. 99]. En la construcción de estos meta-modelos se integran resultados de investigación en forma de

entidades y relaciones entre entidades. La instanciación de estos meta-modelos produce diagramas, los modelos, similares a los que se usa en UML, con la diferencia de que estos diagramas se han creado exclusivamente para definir el sistema multi-agente.

El proceso de instanciación de los meta-modelos no es trivial. Existen muchas entidades y relaciones a identificar, además de dependencias entre distintos modelos. Por ello, INGENIAS [Gomez 02] define un conjunto de actividades cuya ejecución termina en un conjunto de modelos. Estas actividades a su vez se organizan siguiendo un paradigma de ingeniería del software, el Proceso Unificado [Jacobson et al. 99].

La ejecución de actividades para producir modelos se basa en la herramienta INGENIAS IDE (figura 5), una herramienta para modelado visual. Esta herramienta almacena la especificación del sistema utilizando XML.

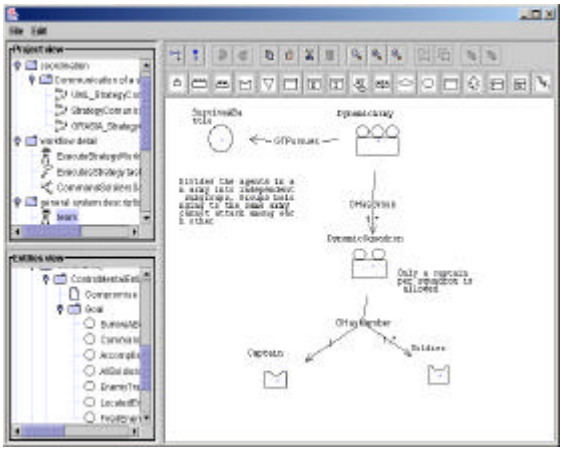

Figura 5. Herramienta de soporte para INGENIAS

Desde la especificación en XML, se plantea el procesarla :

- Para generar código. Genera código rellenando plantillas de código con información de la especificación.
- Para generar la documentación. De una forma similar a la generación de código, se parte de plantillas de documentación que se completan utilizando información de los modelos

Tanto la herramienta de análisis/diseño como el editor se pueden encontrar en http://ingenias.sourceforge.net. Desde esta web también se tiene acceso a ejemplos de especificación siguiendo la metodología.

8.1 Comentarios

El proceso de desarrollo que propone INGENIAS es excesivo cuando se trata de desarrollos reducidos. A diferencia de MAS-CommonKADS que dota de procesos de desarrollo adaptados al tamaño del problema, INGENIAS da la impresión de dedicarse exclusivamente a desarrollos de gran tamaño.

Esta metodología es bastante reciente. Aunque se apoya en MESSAGE y dispone de varios desarrollos donde se experimenta con ella, queda por demostrar su viabilidad real.

Como MAS-CommonKADS, INGENIAS dispone de una cantidad ingente de entidades y relaciones. Su uso mediante la herramienta de soporte INGENIAS IDE, se facilita pero se sigue requiriendo que el desarrollador revise la documentación de la tesis para entender qué hace cada entidad y cuál es el propósito de cada relación.

El proceso de generación de código es más flexible que el ofrecido en ZEUS y MaSE. La principal diferencia consiste en que los desarrolladores pueden configurarlo a voluntad y adaptarlo a sus necesidades sin tener que modificar la herramienta de análisis/diseño.

9 Eligiendo una metodología

Si hubiera que elegir una metodología, ¿cuál sería la ganadora? Esta pregunta no tiene una respuesta simple. Cada metodología, por el bagaje de sus creadores, se especializa en áreas concretas.

Si se está acostumbrado a trabajar con sistemas basados en conocimiento con la metodología CommonKADS, lo lógico es que se elija MAS-CommonKADS. Si la experiencia del usuario está en el área de los objetos, la recomendación sería de MaSE. Si por el contrario está interesado en un enfoque más orientado a agentes, puede seleccionar ZEUS, INGENIAS, BDI o GAIA. Y si lo que se quiere es tener un soporte de herramientas, la lista de metodologías válidas se reduce considerablemente: ZEUS, MaSE, INGENIAS.

En los casos en que se requiera un proceso de desarrollo robusto, detallado y ensayado en desarrollos reales, la recomendación sería MAS-CommonKADS o INGENIAS.

Si solo interesa una visión superficial del sistema, sin entrar en detalles de diseño e implementación, se recomendaría GAIA.

Si se busca una metodología desarrollada a partir de modelos formales, entonces la elección sería BDI.

Otra opción que tienen los desarrolladores es quedarse con lo que más les interese de cada metodología. Una recomendación sería utilizar la técnica de modelado de INGENIAS junto con su herramienta de especificación e implementar con ZEUS.

10 Evaluando metodologías

Existe pocos trabajos relacionados con la evaluación de metodologías de agentes software. Mientras que para metodologías orientadas a objetos, se pueden encontrar bastantes referencias. En el área de los agentes, debido a lo relativamente nuevo de la aplicación de técnicas de ingeniería, hay poco hecho.

Los trabajos conocidos por el autor se limitan a los cuatro siguientes [Ricordel et al. 00], [Shehory et. al. 01], [Cernuzzi et al. 02] y [Omalley et al. 01]. El primero da relevancia a la presencia de plataformas de desarrollo y herramientas de soporte. El segundo se centra en aspectos de modelado, dejando aparte el proceso de desarrollo. El tercero continua esta línea, pero de forma cuantitativa, dando estimaciones numéricas de en qué medida recoge la metodología un aspecto concreto. En el cuarto, se identifican un conjunto de criterios para la toma de decisiones acerca de qué metodología elegir en función de qué necesidades se tengan. Así, se considera cómo es la gestión que se necesita y qué necesidades impone el tipo de proyecto al que se enfrenta el desarrollador. Aunque su aplicación se limita a MaSE, los criterios identificados son reutilizables para otras metodologías.

11 Otras fuentes de información acerca de metodologías

Existen otros surveys, como [Iglesias et al. 98b] [Iglesias et al. 99] o [Ricordel et al. 00]. En el primero se pueden encontrar referencias a las primeras metodologías que se crearon en este área y una comparativa con MAS-CommonKADS. En el segundo, se evalúa la adecuación de cuatro metodologías a las etapas de análisis, diseño, implementación y despliegue.

Los estados del arte de las tesis sobre metodologías de agentes también constituyen una buena fuente de material. Concretamente se recomienda [Inglada 03], [Gómez 02], [Iglesias 98].

De forma indirecta, también se revisan metodologías en trabajos como el de [Paniagua 00] que estudia metodologías desde las que se puedan modelar organizaciones.

Para seguir el desarrollo de metodologías a nivel internacional, existen dos foros principales. El primero es el *Agent Oriented Software Engineering workshop* (AOSE), que generalmente aparece asociado a la conferencia *Automous Agents and Multi-Agent Systems* (AAMAS). El segundo es el grupo de interés de metodologías de *AgentLink*, el *Methodologies and Software Engineering for Agent Systems* (MSEAS), accesible desde http://polaris.ing.unimo.it/MSEAS.

Otro foro internacional es el grupo de interés de inteligencia artificial en la ACM (http://www.acm.org/sigart/). En este grupo se publican también artículos sobre metodologías, aunque no es su tema principal.

Dentro de los foros de habla hispana, se tiene la revista Iberoamericana de Inteligencia Artificial y las conferencias organizadas por la Asociación Española de Inteligencia Artificial (http://www.aepia.org), responsable de la edición de esta revista. También se tiene la conferencia IBERAMIA (http://www.iberamia.org) y especialmente las series de workshop sobre Sistemas Multi-Agente que tienen lugar en cada celebración de esta conferencia.

12 Conclusiones

Las metodologías estudiadas son representantes de diferentes líneas de investigación en agentes: agentes como sistemas basados en concimiento (MAS-CommonKADS), procesos concurrentes (MaSE), agentes BDI (BDI, ZEUS), agentes según la definición de Newell (INGENIAS) o agentes como entidades computacionales que integran diferentes tecnologías (Vowel Engineering).

La aplicación de estas metodologías depende mucho de la formación que tenga su usuario final. Esto quiere decir que son interpretadas en función de los intereses de quien las aplica. De momento, no hay soluciones que fuercen un uso concreto.

Es de destacar que la mayoría de las metodologías expuestas carecen de ejemplos detallados y suficientemente extensos que expliquen cómo se aplica. La presentación de una metodología debería de acompañarse con desarrollos que la avalen y muestren su utilidad.

Finalmente, solo mencionar que la experiencia en otros paradigmas, como el estructurado y el de objetos, dice que los que verdaderamente perduran son la notación y las herramientas. Así pues, estos dos elementos deberían cuidarse en cualquier propuesta. La realidad, en la mayoría de los casos, es que no se cuidan. El tiempo dirá, de las metodologías existentes, qué persiste y qué no. La opinión del autor es que debido al incremento de investigadores con fuerte formación en ingeniería del software, los próximos años van a marcar cambios importantes en este área. Muchas de las notaciones y herramientas que hoy en día se están aceptando como válidas, sufrirán importante cambios y algunas incluso desaparecerán. Es el precio a pagar en el camino hacia la madurez de la tecnología de agentes.

13 Agradecimientos

A Juan Pavón y Francisco Garijo, los directores de mi tesis doctoral. Fueron ellos los que me orientaron hacia el área de las metodologías e hicieron posible que participara en el desarrollo de la metodología MESSAGE.

14 Referencias

[angentTool 03] MultiAgent and Cooperative Robotics Lab:*AgentTool 1.8.3 User's manual*. http://www.cis.ksu.edu/~sdeloach/ai/mase.htm

[Bauer et al. 01] Bauer, B., Müller, J. P., and Odell, J., *Agent UML:A Formalism for Specifying Multiagent Interaction,* International Journal of Software Engineering and Knowledge Engineering (IJSEKE), vol. 11, no. 3, 2001.

[Baümer et al. 00] Baümer, G., Breugst, M., Choy, S., and Magedanz, T.:*Grasshopper: a universal agent platform based on OMG MASIF and FIPA standards.* Informe. IKV ++ Technologies. 2000

[Bellifemi et al. 01] Bellifemine, F., Poggi, A. y Rimassa, G.: *JADE: a FIPA2000 compliant agent development environment*. Actas de conferencia. Proceedings of the fifth international conference on Autonomous agents, ACM. 2001.

[Bratman 87] Bratman, M. E.: *Intentions, Plans, and Practical Reason*. Libro completo. Harvard University Press. 1987.

[Brazier et al. 97] Brazier, F. M. T., Dunin-Keplicz, B. M., Jennings, N. R., and Treur, J., *DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework*, International Journal of Cooperative Information Systems, special issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems, 1997.

[Caire et al 01] Caire, G., Leal, F., Chainho, P., Evans, R., Garijo, F., Gomez-Sanz, J. J., Pavon, J., Kerney, P., Stark, J. y Massonet, P.: *Agent Oriented Analysis using MESSAGE/UML*. Actas de conferencia. Springer Verlag. LNCS 2222. 2001. pp. 119-135

[Carnegie 03] Carnegie Mellon:*Unicon*. http://www-2.cs.cmu.edu/People/UniCon/

[Cernuzzi et al. 02] Cernuzzi,L.,Rossi,G.:*On the Evaluation of Agent Oriented Methodologies*. Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies.

[Collis et al. 99] Collis, J. C. and Ndumu, D. T.:*The Role Modelling Guide*. Informe. Applied Research and Technology, BT Labs. 1999

[DeLoach 01] DeLoach, S.: *Analysis and Design using MaSE and agentTool*. Actas de conferencia. Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conferece (MAICS). 2001.

[Demazeau 95] Demazeau, Y.: *From cognitive interactions to collective behaviour in agent-based systems*. Actas de conferencia. European Conference on Cognitive Science. 1995.

[FIPA 03] FIPA:*ACL Message Structure Specification*. http://www.fipa.org

[Gomez 02] Gomez Sanz, J.: *Modelado de Sistemas Multi-Agente*. Tesis, Facultad de Informática, Universidad Complutense, 2002

[Gomez et al. 02] Gomez-Sanz, J., Fuentes, R.: *The INGENIAS Methodology*. Fourth Iberoamerican Workshop on Multi-Agent Systems Iberagents 2002.

[IBM 03] IBM:*Agent Building and Learning Environment (ABLE)*. URL:

http://www.alphaworks.ibm.com/tech/able

[Iglesias 98] Iglesias, C.: *Definicion de una metodología para el desarrollo de Sistemas Multi-Agente*. Tesis doctoral. Departamento de ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid. 1998.

[Iglesias et al. 98a] Iglesias, C.A., Garijo, M. AND González, J.C.. *Metodologías orientadas a agentes. Inteligencia Artificial.* Revista Iberoamericana de Inteligencia Artificial. Número 6, Volumen 2, Otoño 1998. pp. 12-23.

[Iglesias et al. 99] Iglesias, C.A., Garijo, M. AND González, J.C.. *A Survey of agent-oriented methodologies*. Proceedings of the 5th International Workshop on Agent Theories, Architectures, and Languages, 1999.

[Iglesias et al 98b] Iglesias, C., Mercedes Garijo, M., Gonzalez, J. C., and Velasco, J. R., *Analysis and design of multiagent systems using MAS-CommonKADS*, en *Intelligent Agents IV* . LNAI Volume 1365 ed. SpringerVerlag: Berlin, 1998.

[ITU 99a] International Telecommunication Union:*ITU-120: Formal Description Techniques (FDT): Message Sequence Chart*. Informe.

[ITU 99b] International Telecommunication Union:*ITU 100:Formal Description Techniques (FDT)- Specification and Description Language (SDL)*. Informe.

[Jacobson et al. 99] Jacobson, I., Rumbaugh, J. y Booch, G.: *The Unified Software Development Process*. Libro completo. Addison-Wesley. 1999.

[Inglada 02] Julián Inglada, V. J.: *RT-MESSAGE: Desarrollo de Sistemas Multiagente de Tiempo Real*. Tesis. Universidad Politécnica de Valencia, 2002.

[Kinny et al. 97] Kinny, D., Georgeff, M., and Rao, A.:*A Methodology and Modelling Technique for Systems of BDI Agents*. Informe. 1997

[Lyytinen et al. 99] Lyytinen, K. S., Rossi, M.: *METAEDIT+ A fully configurable Multi-User and Multi-tool CASE and CAME Environment*. Actas de conferencia. Springer Verlag. LGNS 1080. 1999

[MAGMA 03] MAGMA:*MAGMA Research Group*. http://www-leibniz.imag.fr/MAGMA/

[MIX 99] MIX: *Modular Integration of Connectionist and Symbolic Processing in Knowledge-based systems*. Esprit-9119.

[Nwana et al. 99] Nwana, H. S., Ndumu, D. T., Lee, L. C., and Collis, J. C., *ZEUS: A Toolkit for Building Distributed Multi-Agent Systems*, Applied Artificial Intelligence Journal, vol. 1, no. 13, pp. 129-185, 1999.

[Omalley et al. 01] Proceedings of the Second International Workshop On Agent-Oriented Software Engineering (AOSE-2001), Montreal, Canada, May 29th 2001.

[Paniagua et al. 01] Paniagua E., Palma K.G., Martín, G.: *Los Sistemas Multi-Agente para el Modelado de la Actuación en Organizaciones Humanas.* Revista Iberoamericana de Inteligencia Artificial. Número 14, Volumen 11, Otoño 2001. pp. 78-90.

[Pressman 82] Pressman, R. S.: *Software Engineering: A Practitioner's Approach*. Libro completo. McGraw-Hill Series in Software Engineering and Technology. McGraw-Hill, Inc. 1982.

[Ricordel 01] Ricordel, P. M.: *Programmation Orientée Multi-Agents , Développement et Déploiement de Systèmes Multi-Agents Voyelles*. Tesis doctoral. INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE. 2001.

[Ricordel et al. 00] P.M. Ricordel and Y. Demazeau: "*From Analysis to Deployment: A Multi-Agent Platform Survey*", in Omicini A., Tolksdorf R., and F. Zambonelli (Eds.) Engineering Societies in the Agents World, pp. 93-105, Springer-Verlag, 2000.

[Shehory et al. 01] Shehory, O., Sturm, A.: Evaluation of modeling techniques for agent-based systems , ACM Press New York, NY, USA, 2001

[Tansley et al. 93] Tansley, D. S. W. y Hayball, C. C.: *Knowledge Based systems Analysis and Design a KADS developer's handbook*. Libro completo. Prentice Hall. 1993.

[Wood et al. 00] Wood, M. y DeLoach, S.: *Developing Multiagent Systems with agentTool*. Actas de conferencia. ATAL 2000. LNAI 1986. Castelfranchi, C. and Lespérance, Y.2000.

[Wooldridge et al. 00] Wooldridge, M., Jennings, N. R., and Kinny, D., *The Gaia Methodology for*

Agent-Oriented Analysis and Design, Journal of Autonomous Agents and Multi-Agent Systems, vol. 15 2000.

[Zambonelly et al. 00] Zambonelly, F., Wooldridge M., and Jennings N.R., *Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems*, International Journal of Software Engineering and knowledge Engineering, 2000.