Metodologías de desarrollo de sistemas multi-agente: un análisis comparativo

Tulio José Marchetti
tjm@cs.uns.edu.ar

Alejandro Javier García
ajg@cs.uns.edu.ar

Laboratorio de Investigación y Desarrollo en Inteligencia Artificial (LIDIA)*
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Avda. Alem 1253 - (B8000CPB) Bahía Blanca
Tel: ++54 291 4595135 - Fax: ++54 291 4595136

Resumen

Este trabajo presenta y compara algunas metodologías de desarrollo de sistemas multi-agente, y tiene como objetivo extraer cuales son los principios básicos que deberían contemplarse al desarrollar una plataforma o una metodología para este tipo de sistemas. Para esto, se consideraron tres de las metodologías de desarrollo mas prometedoras de la literatura: GAIA, MaSE y PASSI. Como las publicaciones que describen a las metodologías consideradas no utilizan un ejemplo común para mostrar sus características, se definió un ejemplo propio de un sistema multi-agente, a fin de realizar el análisis y la comparación de las mismas. De esta manera, el mismo ejemplo es utilizado a fin de mostrar como se realizaría el diseño y la implementación de un sistema para cada una de las metodologías.

Palabras clave: Inteligencia Artificial, Sistema Multi-agente, Diseño Orientado a Agentes

1. Introducción

El paradigma de agentes y sistemas multi-agente constituye actualmente un área de creciente interés dentro de la Inteligencia Artificial, entre otras razones, por ser aplicable a la resolución de problemas complejos no resueltos de manera satisfactoria mediante técnicas clásicas. Numerosas aplicaciones basadas en este nuevo paradigma vienen ya siendo empleadas en infinidad de áreas, tales como control de procesos, procesos de producción, control de tráfico aéreo, aplicaciones comerciales, gestión de información, comercio electrónico, aplicaciones médicas, juegos, etc..

Debido a la utilización de los sistemas multi-agente para el desarrollo de aplicaciones complejas, cada vez son más necesarios métodos, técnicas y herramientas que faciliten el desarrollo de aplicaciones basadas en dicho paradigma. El progreso en la ingeniería del software en los

Financiado parcialmente por SeCyT Universidad Nacional del Sur (Subsidio: 24/ZN09) y por la Agencia Nacional de Promoción Científica y Tecnológica (PICT 2002 Nro 13096)

*Miembro del Instituto de Investigación en Ciencia y Tecnología Informática (IICyTI)

últimos años se ha realizado gracias al desarrollo de abstracciones más poderosas y naturales para modelar sistemas más complejos. Se podría pensar en el concepto de agente como un avance similar en cuanto a abstracción. Si consideramos a los agentes con el suficiente potencial como un paradigma de la ingeniería del software, entonces es necesario desarrollar técnicas de ingeniería del software que sean específicamente aplicables a este paradigma. Por otra parte, la aceptación de métodos en la industria y/o empresa depende de la existencia de las herramientas necesarias que soporten el análisis, diseño e implementación de agentes software.

En los últimos años han aparecido diferentes aproximaciones que tratan de presentar una metodología apropiada para el desarrollo de sistemas multi-agente. En este artículo se presenta, en primer instancia, una visión generalizada de tres de estas propuestas: GAIA [WK00], MaSE [SADS01, Woo00] y PASSI [CP00]. Las propuestas son analizadas desde diversos puntos de vista, y avanzando un paso mas de la presentación de las mismas realizadas en [JB03, San03]. Posteriormente, se presenta el ejemplo con el cuál se realizará el testeo de estas plataformas y luego utilizando este ejemplo, se presentan en forma detallada los pasos que se deben seguir en cada una de ellas para llegar desde la especificación del problema a una solución implementable.

Consideramos el análisis aquí realizado como un nivel de abstracción mas alto dentro de nuestra investigación reflejada en [MG03a, MG03b] y como la base de la propuesta realizada en [MG04].

2. Metodologías

En esta sección se muestran algunas de las metodologías que se encuentran en la actualidad en la literatura. Las consideradas en este trabajo fueron las que a nuestro parecer son las mas populares dentro del área de investigación.

GAIA

GAIA [WK00] es una metodología para el diseño de sistemas basados en agentes cuyo objetivo es obtener un sistema que maximice alguna medida de calidad global. 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 [FIP02]. 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 modelo 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.

MaSE

MaSE (Multi-agent systems Software Engineering) [SADS01, Woo00] 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 ***"procesos de software que interactúan entre si con el fin de alcanzar una meta global. 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 [AGE03].

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.

PASSI

Para comprender esta metodología, es importante tener en cuenta la definición de agente propuesta en [CP00]. Un agente es ***"una instancia de una clase agente que es la implementación de software de una entidad autónoma capaz de lograr sus objetivos a través de sus decisiones autónomas, sus acciones y sus relaciones sociales"***. Un agente puede tener varios roles para alcanzar sus metas, siendo un rol una función temporal asumida por el agente en la sociedad mientras busca alcanzar una sub-meta.

Dentro de esta metodología podemos identificar varias fases con las cuales se asegura llegar de la especificación del sistema a una versión implementable. Dentro de la etapa de análisis, tenemos las siguientes fases: descripción del dominio: utilizando los diagramas de casos de uso

de UML se describe el dominio de la aplicación, identificación de agentes: a partir del diagrama anterior y utilizando Rational Rose identificamos los agentes que componen el sistema, identificación de roles: los roles de los agentes los representamos mediante diagramas de secuencia e identificación de tareas: se dibuja un diagrama de actividad para cada agente y se deciden que tareas son necesarias para realizar las funcionalidades descritas anteriormente.

En la segunda etapa, la etapa de diseño, podemos identificar otras fases entras cuales se encuentran: descripción de ontologías: se describe la sociedad desde el punto de vista ontológico, descripción de roles: se modela la vida de los agentes observando sus roles, descripción de protocolos: se define cuales protocolos se utilizarán y si es necesario, se definirán los nuevos, definición de la estructura y del comportamiento de los agentes: se diseña la estructura de la sociedad y de los agentes en particular.

Finalmente tenemos las fases de implementación y configuración. La fase de producción de código no está completamente implementada, pero está en desarrollo la búsqueda de una solución en un metalenguaje basado en XML. En cuanto a la configuración, se acentúa su importancia si se trabaja con agentes móviles y con problemas significativos en la diseminación de los agentes en el sistema.

3. Evaluación de las metodologías

Las publicaciones que describen a las metodologías consideradas en este trabajo no utilizan un ejemplo común para mostrar sus características y tampoco existe en la literatura un ejemplo clásico que sirva para este propósito. Todas ellas eligen un ejemplo particular que muestra sus características mas destacables. Sin embargo resulta muy difícil poder comparar sin un ejemplo común. Dentro del marco de este trabajo, se mostrará con un ejemplo propio las características y la forma de trabajo de las metodologías descritas anteriormente. Para esto se consideró un sistema multi-agente que implementa un administrador de impresión para un entorno distribuido el cual fue desarrollado en [Mar02]. El objetivo en la elección de este ejemplo fue lograr el sistema multi-agente más simple posible que muestre las características de las metodologías evaluadas y pueda hacerse una comparación entre ellas.

En este administrador de impresión podemos distinguir tres tipos de agentes (ver Figura 1): los agentes que representan a las impresoras y cuya tarea es imprimir los trabajos que le son enviados por el administrador. Un segundo tipo de agentes que representa a los usuarios, estos envían un pedido de impresión de un archivo con ciertas características al administrador y esperan la respuesta de donde fue impreso dicho trabajo. Finalmente tenemos al administrador que posee una funcionalidad doble. Por un lado mantiene un registro de las impresoras que están conectadas junto con las particularidades de cada una y su estado (busy/free). La segunda función es la de administrar los pedidos de impresión cumpliendo la tarea de un spooler de impresión.

A continuación veremos como son las fases o etapas de cada una de las metodologías antes mencionadas para el ejemplo del administrador de impresión.

GAIA

Para mostrar con el ejemplo propuesto, esta metodología, utilizará los modelos antes mencionados.

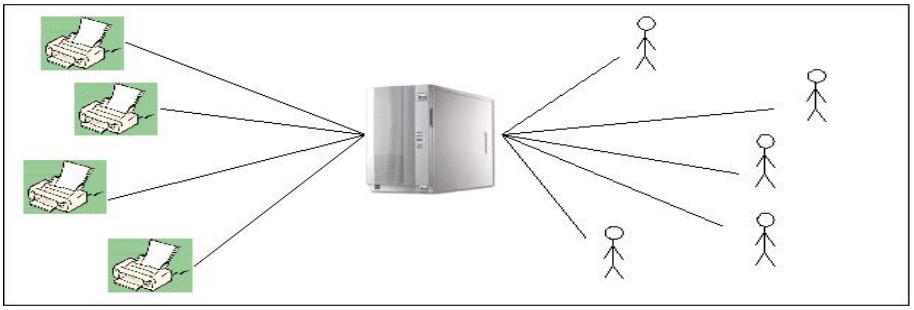

Figura 1: Administrador de Impresión

1. El modelo de roles: identifica los roles claves del sistema. Un rol puede ser visto como una descripción abstracta de una funcionalidad esperada de una entidad. Lo que se puede observar en la Tabla 1.

Rol	**Permisos**	**Responsabilidades**
usuario		Envía documentos imprimir con las características que desea obtener
impresora	**reads/deletes** queue	Recibe un documento y lo imprime
administrador	**write/adds** queues	Determina a que impresora va un documento determinado y lo envía de ser posible sino lo encola

Cuadro 1: Modelo de roles

2. El modelo de interacción: define las relaciones y dependencias entre los roles en una organización multi-agente. Lo que se puede ver en la Tabla 2.
3. Modelo de agente: permite documentar los distintos tipos de agentes que serán utilizados en el sistema. Para el caso del ejemplo tenemos una correspondencia uno a uno entre roles y tipos de agentes. Tenemos los agentes impresoras, los agentes usuarios y el administrador de impresión.
4. Modelo de servicios: permite identificar los servicios asociados con cada rol de agente y especificar las propiedades principales de cada servicio. Por utilizar PROLOG, este paso no es muy claro ya que los agentes no tienen servicios, sino que tienen métodos que son privados y la comunicación se realiza por medio de mensajes. Una posible aproximación sería la Tabla 3.
5. Modelo de comunicación (acquaintance): determina los links de comunicación que existen entre los tipos de agentes. El grafo correspondiente al ejemplo, lo podemos ver en la Figura 2.

MaSE

Los pasos a seguir para esta metodología se vieron anteriormente. De todas maneras realizaremos un breve resumen en esta sección además de mostrar su utilización para el ejemplo

Protocolo	**Imprimir**	**Impreso**
Propósito	Indicar que se quiere imprimir un documento	Indicar que se ha impreso un documento
Iniciador	Usuario	Impresora
Receptor	Administrador	Administrador
Entrada	nombre del documento, tipo de impresora, calidad de impresión	nombre del documento
Salida		
Procesamiento	El administrador encola el documento de acuerdo al tipo de impresora que se seleccionó utilizar	

Cuadro 2: Modelo de interacción

Agente	**Servicio**	**Entrada**	**Salida**	**Precond**	**Postcond**
Usuario	print	documento, tipo de impresión, calidad		el documento tiene que existir	la solicitud se envía a una impresora o a una cola de impresión
Impresora	printed	documento		el documento se imprimió	

Cuadro 3: Modelo de servicios

propuesto.

1. Capturar los objetivos: se toma la especificación inicial del sistema y se transforma en un conjunto estructurado de metas como se puede ver en la Figura 3.
2. Aplicar los casos de uso: este paso es importante en la traducción de las metas en roles y tareas mediante la utilización del diagrama de casos de uso como se puede observar en la Figura 4
3. Refinar los roles: ahora se asegura que se han encontrado todos los roles y que se han desarrollado las tareas que definen el comportamiento de los roles y los patrones de comunicación. Para el caso del ejemplo, no es necesario hacer el diagrama de roles, ya que hay una correspondencia uno a uno entre los roles y los agentes.
4. Crear las clases de agentes: las clases de agentes son identificadas de los roles y documentadas en un diagrama de clases de agentes, como lo muestra la Figura 5.
5. Construcción de conversaciones: una conversación es un protocolo de coordinación entre dos agentes y se representa por dos diagramas de comunicación. Esto esta ejemplificado en las Figuras 6 y 7
6. Ensamblaje de agentes: en esta última etapa del diseño, se crean las clases de los agentes. Este proceso es simplificado con la utilización de un lenguaje de modelado arquitectónico que combina la naturaleza abstracta de los lenguajes de descripción arquitectónica con el

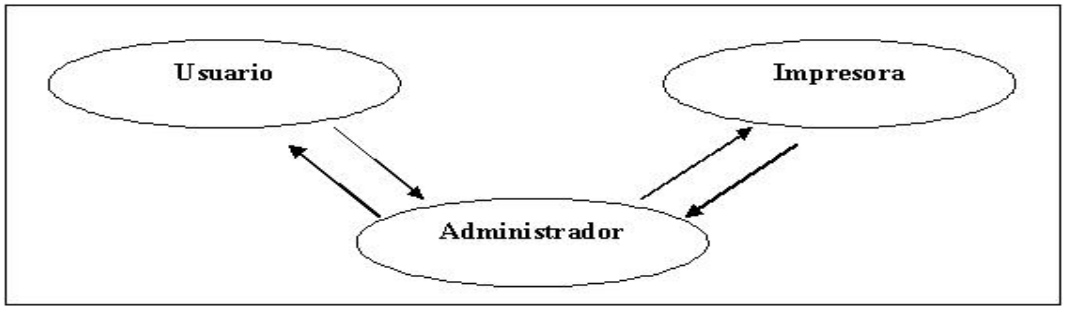

Figura 2: Modelo de comunicación

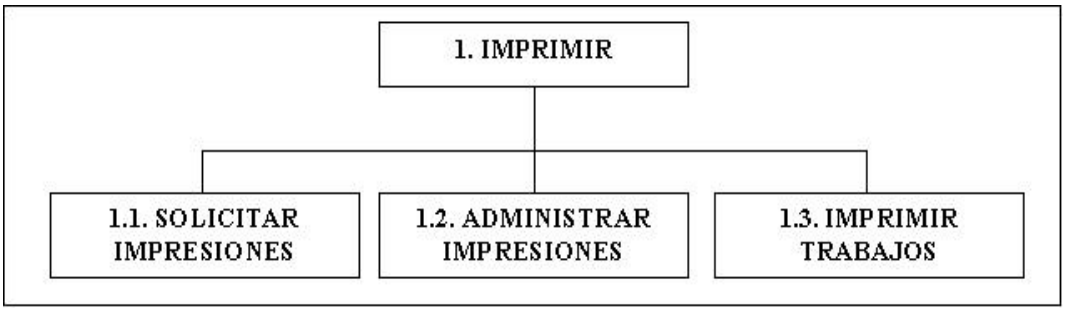

Figura 3: Jerarquía de Metas

lenguaje de restricción de objetos, lo que permite al diseñador especificar detalles de bajo nivel.

7. Configuración del sistema: en esta última etapa de la metodología, se define la configuración del sistema que se implementa. Hasta el momento, solo existe soporte para sistemas estáticos(no móviles).

PASSI

Para esta metodología, al igual que para las anteriores, describimos los pasos necesarios para su utilización. Basados en el ejemplo propuesto, las fases son las siguientes:

1. Modelo de requerimientos de sistema: un modelo de requerimientos de sistema en términos de agencia y propósito esta compuesto de cuatro fases: (a) Descripción del dominio: descripción funcional utilizando diagrama de casos de uso, Figura 4. (b) Identificación de agentes: se atribuyen las responsabilidades a los agentes (c) Descripción de roles: se utilizan diagramas de secuencia para determinar las responsabilidades de cada agente, Figura 8. (d) Especificación de tareas: se especifican las capacidades de cada agente con diagramas de actividades.
2. Modelo de sociedad de agentes: es un modelo de las interacciones sociales y las dependencias entre los agentes involucrados en la solución. Partiendo del modelo anterior (a) Identificación de roles: ver del modelo de requerimientos del sistema, se deben realizar tres pasos: (b) Descripción de ontologías: utiliza diagrama de clases, Figura 9, para describir el conocimiento de los agentes. (c) Descripción de roles: con el diagrama de clases de agentes, Figura 5, se muestran los roles que juega cada agente. (d) Descripción de

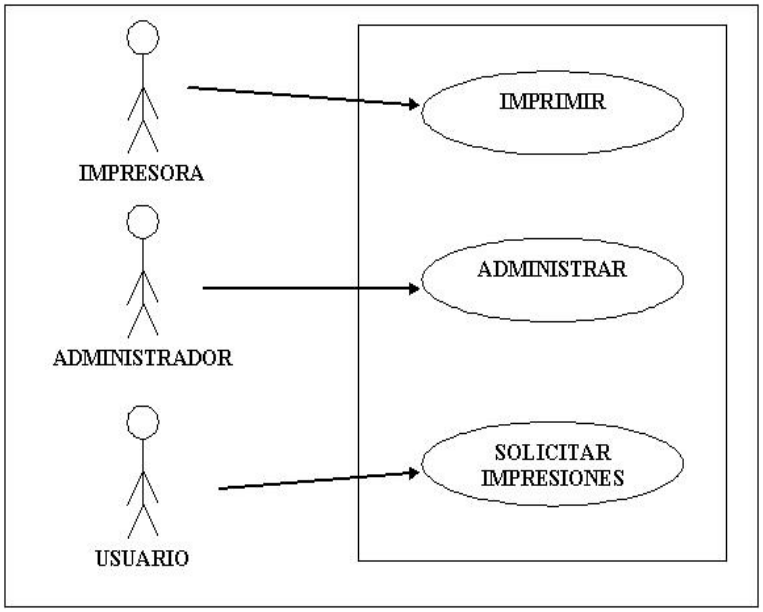

Figura 4: Casos de uso

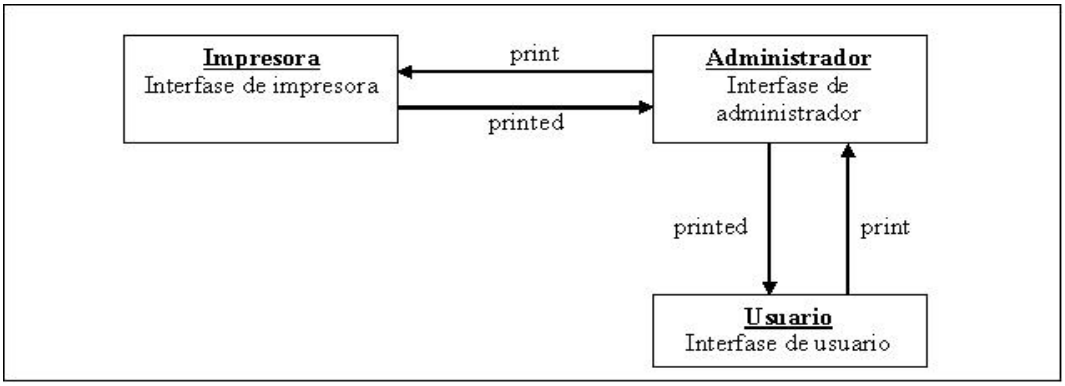

Figura 5: Clases de agentes

protocolos: con los diagramas de secuencia se determina la gramática de cada protocolo de comunicación, Figura 8.

3. Modelo de implementación de agentes: es un modelo clásico en términos de clases y métodos, pero a diferencia de los enfoques orientados a objetos, existen dos niveles de abstracción: el social y el individual. Se utilizan diagramas de clases y diagramas de transición de estados para modelar los agente.
4. Modelo de código: es un modelo de la solución a nivel de código. Involucra (a) generación de código a partir del modelos y (b) completar manualmente el código fuente.
5. Modelo de configuración: un modelo de la distribución del sistema sobre las unidades de procesamiento de hardware utilizando diagramas de "*deployment*"[1].

[1]Se mantiene el término en su lenguaje original por no encontrarse una traducción adecuada.

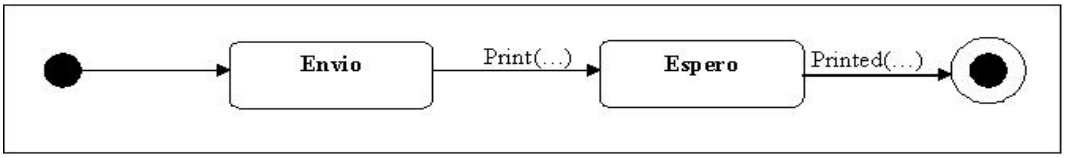

Figura 6: Agente usuario

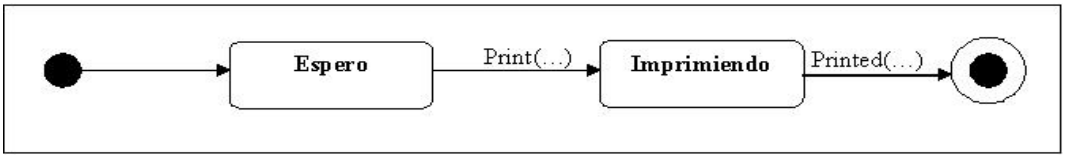

Figura 7: Agente impresora

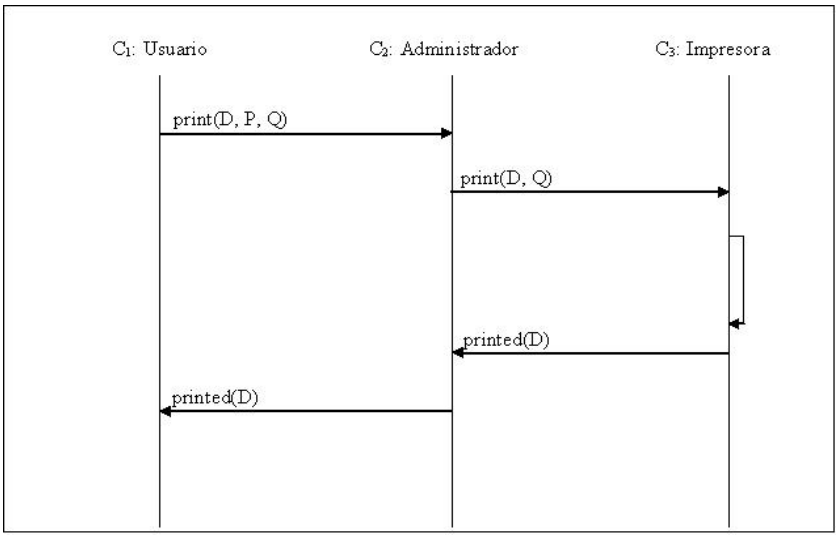

Figura 8: Diagrama de secuencia

4. Comparación de las metodologías

Una metodología constituye un conjunto de etapas o fases que van guiando a los diseñadores partiendo de una definición de alto nivel hacia una aproximación a una implementación. Si estas etapas son pocas entonces es demasiado general y el salto que existe entre una y otra es muy grande. En el otro extremo están las metodologías con muchas etapas y una granularidad muy fina que puede ser tediosa de usar por aquellas personas con alguna experiencia.

En lo que respecta a las etapas de diseño, todas las metodologías comienzan de una manera distinta. Mientras que GAIA comienza con la identificación de los roles de los agentes, MaSE introduce una etapa anterior que tomando la especificación del sistema la transforma en metas. Por otro lado PASSI posee dos etapas anteriores a la etapa de identificación de los roles que son la descripción del dominio y la identificación de agentes, todo dentro del modelo de requerimientos del sistema.

En la etapa siguiente, GAIA define las relaciones y dependencias entre los agentes. MaSE en cambio, primero identifica las clases de agentes y posteriormente construye las conversaciones.

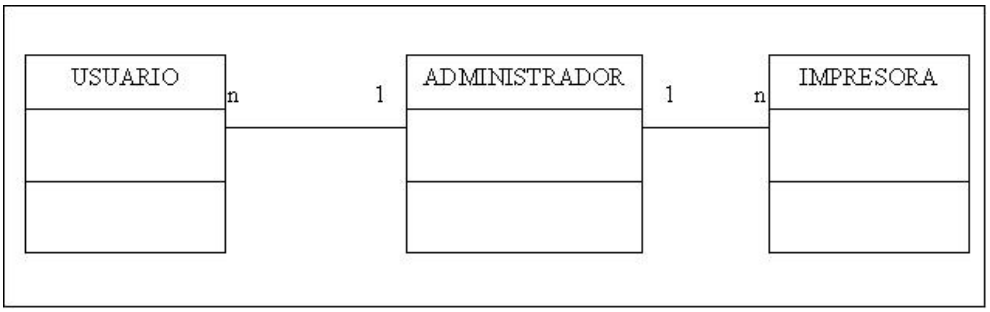

Figura 9: Diagrama de clases de objetos

PASSI identifica los agentes, describe las ontologías y finalmente describe los protocolos de comunicación.

Posteriormente en GAIA se identifican los distintos tipos de agentes, los servicios asociados con cada rol de agente y finalmente se determinan los links de comunicación que existen entre los tipos de agentes. En MaSE se realiza la última etapa de diseño en la cual se crean las clases de agentes. Para la metodología PASSI, esta última etapa involucra el modelado individual y social de los agentes.

En cuanto a la etapa de implementación y generación de código tenemos los siguientes resultados:

- GAIA es sólo de análisis y diseño por lo que no está diseñada para proveer una forma de implementar lo que en ella se obtiene.
- MaSE posee una herramienta, AgentTool [AGE03], que permite la implementación de esta metodología.
- PASSI no posee en la actualidad la posibilidad de generar código, aunque en [CP00] indica que esta característica está en desarrollo.

Desde nuestro punto de vista, una metodología de desarrollo de sistemas multi-agente debería contemplar una variedad de etapas, las cuales no necesariamente siguen en forma exacta a ninguna de las metodología incluidas en este trabajo. Las etapas que a nuestro parecer debería incluir una metodología son las siguientes:

1. **Descripción del dominio**: esta etapa podría realizarse como lo indica la metodología PASSI utilizando diagrama de casos de uso.
2. **Identificar las metas**: Al igual que MaSE o PASSI, de la etapa (1) es posible identificar las metas que tendría el sistema.
3. **Identificar los agentes**: al igual que la etapa (2), partiendo del diagrama de la etapa (1) se pueden identificar los agentes al igual que en las metodologías estudiadas.
4. **Asociar cada meta con un rol de un agente**: Partiendo de las etapas (2) y (3), de la misma manera que las metodologías vistas anteriormente, asociamos cada meta con un rol de agente.
5. **Identificar las interacciones entre agentes**: Al igual que las metodologías evaluadas, se deben identificar todas las formas en interactuarán los agentes.

6. **Seleccionar un protocolo para cada interacción identificada**: Partiendo de la etapa (5), para cada interacción se debe seleccionar un protocolo específico.
7. **Seleccionar el lenguaje de comunicación entre agentes**: una vez definidos los protocolos de interacción en la etapa (6), es necesario decidir si se utilizará un lenguaje de comunicación entre agentes como KQML o FIPA ACL o simplemente se definirá un conjunto reducido de actos comunicativos propios para el sistema.
8. **Clasificar los agentes según su tipo** : Es necesario, partiendo de las etapas (3) y (5), definir que tipo de agente es cada uno dentro del modelo (por ejemplo reactivos, deliberativos, de interfase, etc.).
9. **Seleccionar una arquitectura para los agentes**: Basado en la etapa (8) seleccionamos una arquitectura para cada tipo de agente encontrado.
10. **Producir el código**: Para cada una de las arquitecturas seleccionadas en (9) se debe decidir como implementar el agente. Para esto hay varias opciones:

    a) Utilizar una plataforma de desarrollo de sistemas multi-agente.
    b) Utilizar el mismo lenguaje de programación para todas las arquitecturas.
    c) Utilizar diferentes lenguajes de programación dependiendo de la arquitectura.

5. Conclusiones y trabajo futuro

En este artículo se ha realizado un análisis de distintas aproximaciones al problema de la construcción de sistemas multi-agente. Como se pudo observar en las secciones anteriores, todas estas metodologías poseen una forma muy distinta de trabajo. Cada una de ellas utiliza distintas técnicas y modelos en su búsqueda de modelar e implementar un sistema multi-agente. También se vio que todas estas metodologías de diseño y desarrollo de sistemas multi-agente están fuertemente basadas en el paradigma de orientación a objetos. Basadas en este paradigma hacen uso de algunos de sus modelos y de UML para las etapas de análisis y diseño del sistema.

Como trabajo futuro queda pendiente analizar el funcionamiento de otras metodologías y evaluarlas en búsqueda de una metodología orientada a agentes. Esta metodología deberá permitir, partiendo de una descripción de alto nivel llegar a una implementación de la forma más natural posible.

Referencias

[AGE03] AGENTTOOL. *The agentTool Proyect.* Kansas State University, Department of Computing and Information Sciences, http://www.cis.ksu.edu/ sdeloach/ai/agentool.htm, 2003.

[CP00] Cossentino, and Potts. A case tool supported methodology for the design of multi-agent systems.

[FIP02] FIPA. *Foundation for Intelligent Physical Agents.* http://www.fipa.org/, 2002.

[JB03] Julián, V., and Botti, V. Estudio de métodos de desarrollo de sistemas multiagente. *Revista Iberoamericana de Inteligencia Artificial.*, 18 (2003), 65–80.

[Mar02] Marchetti, T. J. *Software de Soporte para Comunicación entre Agentes. Una implementación para KQML y FIPA ACL*, August 2002.

Tesis de Licenciatura en Ciencias de la Computación, Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur, Bahía Blanca, Argentina.

[MG03a] Marchetti, T. J., and García, A. J. Evaluación de plataformas para el desarrollo de sistemas multiagente. *IX Congreso Argentino de Ciencias de la Computación. La Plata, Argentina* (2003), 625–636.

[MG03b] Marchetti, T. J., and García, A. J. Plataformas para desarrollo de sistemas multiagente. un análisis comparativo. *V Workshop de Investigadores en Ciencias de la Computación. Tandil, Argentina* (2003), 389–393.

[MG04] Marchetti, T. J., and García, A. J. Plataformas para desarrollo de sistemas multiagente. un análisis comparativo. *VI Workshop de Investigadores en Ciencias de la Computación. Neuquén, Argentina* (2004), 384–388.

[SADS01] Scott A. DeLoach, M. F. W., and Sparkman, C. H. Multiagent systems engineering. *The International Journal of Software Engineering and Knowledge Engineering 11*, 3 (2001).

[San03] Sanz, J. G. Metodologías para el desarrollo de sistemas multi-agente. *Revista Iberoamericana de Inteligencia Artificial.*, 18 (2003), 51–63.

[WK00] Wooldridge, J., and Kinny. The gaia methodology for agent-oriented analysis and design.

[Woo00] Wood, M. F. Multiagent systems engineering: A methodology for analysis and design of multiagent systems. Master's thesis, Air Force Institute of Technology, 2000.