UNIVERSIDAD CENTROCCIDENTAL
"LISANDRO ALVARADO"

ANÁLISIS COMPARATIVO ENTRE METODOLOGÍAS ORIENTADAS A AGENTES

JESÚS RAFAEL RODRÌGUEZ ROMERO

Barquisimeto, 2005

UNIVERSIDAD CENTROCCIDENTAL
"LISANDRO AL VARADO"
DECANATO DE CIENCIAS Y TECNOLOGÍA
MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN
MENCIÓN INTELIGENCIA ARTIFICIAL

ANÁLISIS COMPARATIVO ENTRE METODOLOGÍAS ORIENTADAS A AGENTES

Trabajo presentado para optar al grado de
Magíster Scientiarum

Por: JESÚS RAFAEL RODRÌGUEZ ROMERO
jesus_rafael@hotmail.com

Barquisimeto, 2005

AGRADECIMIENTOS

Agradezco primero que todo a **Dios**, (ya él sabe todas las cosas por las que le agradezco y le seguiré agradeciendo)

A ***mi Familia*** por todo el apoyo brindado, muy especialmente el de mi mamá.

A ***mis Compañeros de Estudios*** a quienes les agradezco tantas cosas: Rafael, Francis, Oscar, Gerardo, Manuel, Gustavo, Luisa, Maria Auxiliadora, Wilmer, Sonia y muy especialmente a Carlos Lameda por ser el guía, ejemplo y amigo de todo el grupo.

A todos ***los Profesores de Postgrado*** que de una u otra manera nos brindaron sus conocimientos y ayuda a lo largo de toda la escolaridad.

A la Prof. ***Gladis Marante*** y La Prof. ***Maria Luisa Cappodieci***, por haberme orientado a ingresar al postgrado en la Mención de Inteligencia Artificial, quienes me apoyaron y confiaron en mí desde un principio. Gracias a cada una de ustedes.

A La Prof. ***Maritza Bracho***, por haber aceptado ser mi tutora, por orientarme en lo referente a la Tesis y por haberme tenido paciencia para que terminara este trabajo.

A ***Henry Rodríguez***, mi gran amigo, por tantos consejos, favores y pare de contar. Gracias.

A ***Carmela Lozada***, por ayudarme siempre en lo que estuvo a su alcance.

En fin, a ***Todos mis amigos y amigas***, que de una u otra manera me apoyaron y alentaron a mantenerme y culminar mi postgrado. No creo poder nombrarlos a TODOS, así que mi más sincero agradecimiento a cada uno de ustedes.

Gracias a Todos....

UNIVERSIDAD CENTROCCIDENTAL
"LISANDRO AL VARADO"
DECANATO DE CIENCIAS Y TECNOLOGÍA
MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN
MENCIÓN INTELIGENCIA ARTIFICIAL

ANÁLISIS COMPARATIVO ENTRE
METODOLOGÍAS ORIENTADAS A AGENTES

Autor: Jesús Rafael Rodríguez Romero
Tutora: Maritza Bracho

RESUMEN

Existen varias metodologías para el desarrollo de Sistemas Multi-Agentes (SMA), estas metodologías parten de un modelo, informal en la mayoría de casos, de cómo debe ser un SMA, en las primeras metodologías, las guías consistían en una lista breve de pasos a seguir. Las metodologías más recientes como MaSE o GAIA, aún no muestran la madurez que se puede encontrar en metodologías convencionales como el Proceso Unificado Racional (RUP). El motivo principal de no alcanzar la madurez requerida es que siguen faltando herramientas de soporte y un lenguaje para la especificación del SMA que permitan aplicarlas de forma similar a las metodologías Orientadas a Objetos.

Basándose en esta consideración se realiza este trabajo, ubicado en el área de agentes racionales y esta enmarcado en la modalidad de proyecto de investigación monográfica documental, el cual consiste fundamentalmente en explicar conceptualmente y analizar algunas de las metodologías existentes para el desarrollo de agentes racionales, para de esta manera poder construir un cuadro comparativo entre estas metodologías, que permita exponer sus similitudes y diferencias más relevantes, conformándose en un apoyo al momento de crear y modelar agentes, mediante el cual el diseñador podrá elegir con mayor facilidad la que le conviene o se adapta mejor a su objetivo de desarrollo. Dentro de las metodologías que se tomaron en cuenta para el estudio se encuentran: **BDI** (Belief Desire Intentiton), **GAIA**, **MaSE** (Multiagent Systems Engineering), MAS-Common KADS y Cassiopeia.

Palabras Claves: Metodologías de Agentes, Inteligencia Artificial, **BDI**, **GAIA, MaSE**, MAS Common KADS, Cassiopeia.

INDICE

Pág

INDICE DE ILUSTRACIONES

Figuras Pág.

GLOSARIO DE TERMINOS Y ABREVIATURAS

agentTool	Sistema de desarrollo de agentes creado por la Air Force Institute of Technology's (AFIT)
ASM	Abstract State Machines
BDI	Belief Desire Intention (Creencias, Deseos e Intenciones)
ITU	International Telecommunication Union 99 A.D.b
MAS - SMA	Multi-Agent system (Sistemas Multi-Agentes)
MaSE	Multi-agent systems Software Engineering
OMG	Object Management Group
OO	Orientado a Objetos
Paradigm+	Software desarrollado por PLATINUM Technology International, Inc, como herramienta para casos OO
Rational Rose	Herramienta para modelado en UML, desarrollada por SOFTWARE DEVELOPMENT y RATIONAL SOFTWARE CORPORATION
Rol	Papel que desempeña una persona o grupo en cualquier actividad. Conducta que un grupo espera de un miembro en una situación determinada
SBC	Sistemas Basados en el Conocimiento
TogetherJ	Programa para desarrollar UML para aplicaciones en JAVA, desarrollada por TogetherSoft.
UML	Unified Modeling Language (Lenguaje de Modelamiento Unificado)
OMT	Object Modeling Technique (Técnica de Modelado de Objeto)

INTRODUCCION

En un mundo globalizado donde el avance tecnológico cada día juega un papel primordial, principalmente en la automatización de procesos y en la toma de decisiones, la inteligencia artificial es un punto clave en el desarrollo de los sistemas informáticos, en la construcción de estos programas se está aplicando una tecnología íntimamente relacionada con la inteligencia artificial. Se trata de los ***agentes,*** que son programas autónomos e inteligentes. Bajo el punto de vista de esta tecnología, los sistemas distribuidos pasan a ser *Sistemas Multi-Agente* (SMA).

Al igual que para los sistemas de información convencionales, para los agentes también existen metodologías, principalmente para el desarrollo de Sistemas Multi-Agentes (SMA). Metodologías que brindan una guía para los diseñadores, indicando una serie de pasos enmarcados dentro de fases de desarrollo. En este trabajo, fundamentalmente se tratara de explicar conceptualmente y analizar algunas de las metodologías existentes para el desarrollo de agentes racionales, permitiendo exponer sus similitudes y diferencias más relevantes, conformándose en un apoyo al momento de crear y modelar agentes.

CAPITULO I

EL PROBLEMA

Planteamiento del Problema

A finales de la última década del siglo pasado y en los pocos años que van del siglo XXI, ha cobrado fuerza la hipótesis de que si los programas que componen los sistemas de información distribuidos fueran inteligentes, si pudiesen reorganizar sus funciones para corregir o tolerar los fallos generales que se le presenten y si sus ejecución pudiese estructurarse en términos intuitivos, el diseño y el mantenimiento de dichos sistemas sería más fácil de elaborar, más adaptable y más fiable [Gómez, 03].

En la Inteligencia Artificial (IA) ha surgido un paradigma conocido como «paradigma de agentes», el cual está tomando un gran auge entre los investigadores. Dicho paradigma aborda el desarrollo de entidades que puedan actuar de forma autónoma y razonada. Encontrándonos ante el nacimiento de una nueva tecnología, la **de agentes racionales** (llamados también por algunos como agentes inteligentes), que permite abordar de una manera más apropiada la construcción de sistemas de información racionales más complejos aplicados a diversos campos.

Tal como dijo el Dr. Nicholas Jennings [Jennings, 99], Los agentes constituyen el próximo avance más significativo en el desarrollo de sistemas y pueden ser considerados como la nueva revolución en el software. Avance que aun se estudia para determinar su impacto en el desarrollo de sistemas de información.

Es importante resaltar que en el momento en que se disponga de metodologías claras para la construcción de dichos agentes racionales, la aplicabilidad de las técnicas de IA a cualquier tipo de problema podrá abordarse de una manera más clara y unificada. La mayoría de las metodologías para el desarrollo de agentes expresan son sugerencias de cómo desarrollarlo, dejando muchas observaciones al libre albedrío del diseñador.

La IEEE 1074-1995 estándar [IEE96] describe el proceso de desarrollo de software, las actividades a ser realizadas, y las técnicas que pueden ser usadas para desarrollo de software. Las actividades no son presentadas en un orden de tiempo, desde el estándar se recomienda que ellas sean incorporadas en un ciclo de vida de software, que es seleccionado y establecido por el usuario a desarrollar el proyecto. El estándar no define un particular ciclo de vida. Estas actividades forman parte de lo que es llamado procesos del software. Posteriores a este estándar no se hace mención de algún lineamiento para metodologías que puedan emplear agentes.

La falta de estándares de metodologías en la Ingeniería de Software Orientada a Agentes y la existencia de varias metodologías para el desarrollo de agentes, hace necesaria la revisión de las distintas propuestas con el fin de encontrar las ventajas y desventajas de cada una, lo que genera un problema para los desarrolladores que se inician en el tema de agentes o para aquellos que aún no cuentan con una metodología propia. El problema es poder determinar cual de las metodologías existentes, brinda mayor dinamismo y robustez para modelar y crear agentes, o sencillamente cual de las metodologías es la que se adapta al tipo de agente que se desea desarrollar.

De entre las metodologías existentes, tres criterios básicos para su desarrollo son: utilización de diferentes vistas para la especificación del sistema de información, incorporar la idea de proceso de desarrollo, e integrar técnicas de ingeniería de software y teoría de agentes, pero debido a la diversidad de metodologías y a las pocas bibliografías que planteen análisis comparativos entre éstas, se pretende desarrollar un marco comparativo entre algunas de las metodologías, el cual permitirá establecer las similitudes y diferencias que se deben tener en cuenta a la hora de decidir la que se va a implementar.

Objetivos

Generales

Desarrollar un análisis comparativo entre algunas de las metodologías orientadas a Agentes, entre las que se encuentran: **BDI** (Belief Desire Intentiton), **GAIA**, **MaSE** (Multiagent Systems Engineering), MAS Common KADS y Cassiopeia, el cual permita resaltar las características fundamentales de cada metodología, así como también las similitudes y diferencias existente entre las metodologías elegidas para el caso de estudio.

Específicos

- Describir en que consisten las metodologías **BDI**, **GAIA**, **MaSE,** MAS Common KADS y Cassiopeia para el desarrollo de agentes racionales.

- Determinar las similitudes o semejanzas, y las diferencias que existen entre las metodologías de desarrollo de agentes, que se seleccionaron para el estudio, que puedan ser útiles al momento de seleccionar alguna de ellas.

- Crear cuadros sinópticos que resuman los resultados obtenidos en el análisis cualitativo, en función de los estudios de ciclos de vida, debilidades y/o fortaleza, similitudes y/o diferencias.

Justificación e Importancia

En el momento en que surgieron los primeros desarrollos de agentes racionales, sus metodologías consistían en una lista breve de pasos a seguir [Gómez, 00]. En poco tiempo su popularidad y versatilidad para numerosas aplicaciones permitieron ampliar su horizonte de desarrollo y de esta manera incrementar su complejidad, por lo que algunos desarrolladores se plantearon métodos que le permitiesen la creación de agentes de una forma organizada y controlada, lo que generó metodologías mucho mas completas.

Tal como se indico en el planteamiento del problema, la existencia de múltiples metodologías para el desarrollo de agentes, sumado a las pocas referencias bibliográficas en las que se aborden este tipo de comparaciones, produce un proceso de diseño y desarrollo de agentes lento y no estandarizado. Debido a ello se considera necesario y de gran ayuda para los desarrolladores de agentes establecer análisis comparativos, como apoyo bibliográfico que faciliten la labor de selección de la metodología que reúna las características requeridas para el tipo de agente en particular que se desee desarrollar

En particular este trabajo se basa en un análisis comparativo de algunas de las metodologías más comunes, en el que se especifique las bondades y/o carencias que estas presentan. Información que podría ser requerida en algunos casos durante el desarrollo de agentes o mejor aún durante la selección de la metodología a seguir para la construcción de agentes.

La importancia de este estudio radica en aportar información útil que ayude al diseñador, sobre las particularidades presentadas en estas metodologías, reduciendo de esta manera el tiempo que, por lo general, se invierte en la selección de alguna metodología, debido a la diversidad de metodologías y a la carencia de herramientas que sirvan de apoyo para tal fin.

Alcance y Limitaciones

El auge que ha tenido el desarrollo de agentes ha llevado a la generación de varias metodologías para crear y modelar agentes. En consecuencia, esta investigación se limita a analizar cualitativamente cinco de las más conocidas y prominentes metodologías, según [Arsène et al, 02], [Dam et al, 03], [Bonesi, 01], [Botti et al, 00] y otros, quienes resaltan varias metodologías, de las cuales se tomaron las que coinciden entre ellos, estas son: **BDI** (Belief Desire Intentiton), **GAIA**, **MaSE** (Multiagent Systems Engineering), MAS Common KADS y Cassiopeia.

Se abordarán algunos aspectos claves en cada una de las metodologías buscando aquellos más resaltantes que permitan exponer similitudes o diferencias muy marcadas, que son las que determinan su uso o no de la metodología dentro del área de desarrollo de agentes. Dentro de los aspectos a evaluar por cada metodología estarán:

- Conformación de su ciclo de vida.
- Debilidades y/o fortalezcas que presenten para determinados casos.
- Similitudes y/o diferencias con respecto a las demás metodologías presentadas en la investigación.
- Los aspectos que se destaquen por cada metodología durante la investigación.

CAPITULO II

MARCO TEORICO

Antecedentes de la Investigación

Los ***agentes*** nacieron de la investigación en Inteligencia Artificial (IA). Al principio, la IA hablaba de los agentes como programas especiales cuya naturaleza y construcción no se llegaba a detallar. Recientemente se ha escrito mucho acerca de este tema explicando qué se puede entender por un agente y qué no [Wooldridge et al, 98] [Gómez, 00].

En el desarrollo de Sistemas Multi-Agentes (SMA) existen dos filosofías distinguidas por la carga de componente práctica. La primera ve el SMA como el resultado de utilizar un lenguaje de especificación de agentes. Para generar SMA de esta manera, se parte de principios basados en modelos operacionales y modelos formales de SMA [Lespérance et al, 96] [De Giacomo et al, 00]. La segunda estudia el SMA como un sistema software que hay que construir. El desarrollo no parte de cero, sino que utiliza plataformas de desarrollo de agentes que proporcionan servicios básicos de comunicación, gestión de agentes y una arquitectura de agente [Bellifemine et al, 01], [Baümer et al. 00]. En cualquiera de los dos casos, y sobre todo cuando el sistema a desarrollar es grande, se necesitan metodologías que estructuren el desarrollo de acuerdo con las prácticas de ingeniería del software.

Existen numerosas metodologías para el desarrollo de SMA, 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 [Gómez, 00]. 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 la de Proceso Unificado Racional [Gómez, 00]. 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+ utilizados en las metodologías Orientadas a Objetos.

En la mayoría de la bibliografía revisada ([Iglesias et al, 02], [Collinot et al, 96], [Kinny et al, 97], [Bonesi, 01] y otros), se explican muchas de estas metodologías, pero en muy pocas se encuentran detalles sobre las comparaciones o similitudes que éstas puedan tener entre sí, en muchas solo se aporta si las metodologías cumplen con ciertos criterios o no, debido a que no se cuenta con antecedentes que resalten u ofrezcan aportes significativos referentes al tema de comparaciones entre metodologías, y como resaltan [Iglesias et al, 02] que a pesar del vertiginoso progreso de teorías, arquitecturas y lenguajes de agentes, se ha realizado muy poco trabajo en la especificación de técnicas para desarrollar aplicaciones empleando tecnología de agentes. La introducción de la tecnología de agentes en la industria requiere de metodologías que asistan en todas las fases del ciclo de vida del sistema de agentes. La necesidad de estas técnicas será especialmente necesaria a medida que el número de agentes de los sistemas aumente.

Se ha avanzado mucho en el desarrollo de sistemas basados en agentes pero aún no se ha unificado todo el conocimiento adquirido en este continuo avance, es cierto que quizás nunca se llegue a unificar todo, pero si se pueden realizar trabajos para interconectar y analizar parte de esa información recolectada y aportarle a los desarrolladores las herramientas que faciliten la ardua labor de modelar agentes.

Bases Teóricas

1 - Definición de Metodologías

Para definir que es una metodología, hay que partir de la definición de dos términos íntimamente relacionados a la palabra metodología, estos son: el Método y la Técnica.

El método, según [Zorrilla et al, 99], proviene de las voces griegas *meta* que significa con, y *odos*, camino; esto es, manera de proceder para descubrir algo o alcanzar un fin. El método representa la manera de conducir el pensamiento o las acciones para alcanzar un fin. Los métodos, de un modo general y según la naturaleza de los fines que procuran alcanzar, pueden ser agrupados en tres tipos, a saber: (a) métodos de transmisión, son los destinados a transmitir conocimientos, actitudes o ideales, se denominan métodos de enseñanza; (b) métodos de organización, son los que trabajan sobre hechos conocidos y procuran ordenar y disciplinar esfuerzos para que haya eficiencia en lo que se desea realizar; (c) métodos de investigación, destinados a descubrir nuevas verdades, a esclarecer hechos desconocidos o a enriquecer el patrimonio de conocimientos, pueden ser de investigación religiosa, filosófica o científica, de acuerdo con el mundo de valores o hechos que se pretenda esclarecer. Por otro lado la Técnica, según [Bisquerra, 89], son solo medios auxiliares que concurren a la misma finalidad.

Los conceptos de método y técnica no están totalmente esclarecidos, habiendo definiciones diferentes al respecto. Por lo general, el método es más amplio que técnica en sentido de que método indica aspectos generales de acción no específica, mientras que técnica conviene al modo de actuar, objetivamente, para alcanzar una

meta. Para alcanzar sus objetivos, un método necesita echar mano de una serie de técnicas. Se puede decir que el método se hace efectivo a través de las técnicas.

Una definición más explícita sobre Técnica, la encontramos en [Zorrilla et al, 99] donde se dice que es el sistema de supuestos y reglas que permite realizar algo. La diferencia entre el método y la técnica radica en que el primero es el proceso fundamental mediante el cual avanza toda ciencia, y técnica es la manera particular en que se emplea el método.

Santiago [Zorrilla et al, 99] también define Metodología, y explica que se compone de los términos *método* y *logos*, que significa explicación, juicio, tratado, estudio de los métodos, es decir, representa la manera de organizar el proceso de la investigación, de controlar sus resultados y de presentar posibles soluciones a un problema que conlleva la toma de decisiones.

Se ha definido metodología en términos generales, pero dentro de la Ingeniería de Software esta puede definirse, en un sentido amplio, como un conjunto de métodos o técnicas que ayudan en el desarrollo de un producto software, y tal como señala [Rumbaugh et al, 91], es un proceso para el desarrollo y producción organizada del software, empleando un conjunto de técnicas predefinidas y convenciones en las notaciones. Una metodología se presenta normalmente como una serie de pasos, con técnicas y notaciones asociadas a cada paso.

Las principales actividades de una metodología según [Pressman, 97] son:

- La definición y descripción del problema que se desea resolver.

- El diseño y descripción de una solución que se ajuste a las necesidades del usuario.

- La construcción de la solución.

- La prueba de la solución implementada.

Entre los requisitos que debe cumplir una metodología se pueden citar los siguientes:

- La metodología está *documentada*: el procedimiento de uso de la metodología está contenido en un documento o manual de usuario.

- La metodología es *repetible*: cada aplicación de la metodología es la misma.

- La metodología es *enseñable*: los procedimientos descritos tienen un nivel suficientemente detallado y existen ejemplos para que personal cualificado pueda ser instruido en la metodología.

- La metodología está basada en *técnicas probadas*: la metodología implementa procedimientos fundamentales probados u otras metodologías más simples.

- La metodología ha sido *validada*: la metodología ha funcionado correctamente en un gran número de aplicaciones.

- La metodología es *apropiada* al problema que quiere resolverse.

- La metodología ha de ser *Adaptativa*: lo que le permite ser moldeable y configurarse a un problema en particular

Las metodologías están formadas por Modelos, Según [Álvarez, 98], Los modelos son un instrumento muy común en el estudio de sistemas de toda índole. Los modelos son especialmente importantes porque ellos nos ayudan a comprender el funcionamiento de los sistemas. El empleo de modelos facilita el estudio de los sistemas, aún cuando éstos puedan contener muchos componentes y mostrar numerosas interacciones como puede ocurrir si se trata de conjuntos bastante complejos y de gran tamaño.

Un modelo es un bosquejo que representa un conjunto real con cierto grado de precisión y en la forma más completa posible, pero sin pretender aportar una réplica de lo que existe en la realidad. Las metodologías proponen modelos de sistemas que representan los procesos, los flujos y las estructuras de datos. Las metodologías por lo general se basan en el modelo básico **entrada/proceso/salida** [Alvarez, 98]**.**

2 - La Ingeniería de Software y Metodologías

2.1- La Ingeniería de Software

Según la definición del IEEE, citada por [Lewis, 94] software es la suma total de los programas de computadora, procedimientos, reglas, la documentación asociada y los datos que pertenecen a un sistema de cómputo. Según el mismo autor, un producto de software es un producto diseñado para un usuario. En este contexto, la Ingeniería de Software es un enfoque sistemático del desarrollo, operación, mantenimiento y retiro del software, y en palabras más sencillas, se considera que la **Ingeniería de Software** es la rama de la ingeniería que aplica los principios de la ciencia de la computación y las matemáticas para lograr soluciones efectivas a los

problemas de desarrollo de software, es decir, permite elaborar consistentemente productos correctos, utilizables y costo-efectivos [Cota, 94].

Según [Pressman, 93], la Ingeniería de Software es el establecimiento y uso de principios de ingeniería robustos, orientados a obtención económica de software que sea confiable y funcione eficientemente sobre máquinas reales.

El **Proceso de Ingeniería de Software** se define como un conjunto de fases parcialmente establecidas con la intención de lograr un objetivo, en este caso, la obtención de un producto de software de calidad [Jacobson, 98]. El P**roceso de Desarrollo de Software** es aquel en que las necesidades del usuario son traducidas en requerimientos de software, estos requerimientos transformados en diseño y el diseño implementado en código, el código es probado, documentado y certificado para su uso operativo. Concretamente define quién está haciendo qué, cuándo hacerlo y cómo alcanzar un cierto objetivo [Jacobson, 98].

2.2 - Metodologías desarrolladas para la Ingeniería de Software

El proceso de desarrollo de software requiere por un lado un conjunto de conceptos, una metodología y un lenguaje propio. A este proceso también se le llama el **ciclo de vida del software** que comprende cuatro grandes fases a nivel muy general: concepción, elaboración, construcción y transición. La **concepción** define el alcance del proyecto, la **elaboración** define un plan del proyecto, especifica las características y fundamenta la arquitectura, la **construcción** crea el producto y la **transición** transfiere el producto a los usuarios [Zavala, 00], obviamente cada una de estas fases reúne una serie de pasos según la metodología que se aplique y el tipo de software que se desee desarrollar, en muchos casos cambia el nombre de la fase e incluso puede variar el numero de fases dependiendo de la metodología.

A continuación se presentan algunas metodologías diseñadas para la ingeniería de software:

2.2.1 - Estructurada (Análisis Estructurado)

En primera instancia debemos decir que el análisis estructurado según Senn "permite al analista conocer un sistema o proceso (actividad) en una forma lógica y manejable al mismo tiempo que proporciona la base para asegurar que no se omite ningún detalle pertinente". El objetivo que persigue el análisis estructurado es organizar las tareas asociadas con la determinación de requerimientos para obtener la comprensión completa y exacta de una situación dada. Se puede decir adicionalmente que los componentes del análisis estructurado son los siguientes: símbolos gráficos, diccionarios de datos, descripciones de procesos y procedimientos, reglas [Yourdon, 91].

A finales de los años 60´s e inicios de los 70´s el análisis estructurado surge de la necesidad de buscar una forma interpretativa más rápida y eficiente, de tal manera que se pudiesen definir los requerimientos del usuario y las especificaciones funcionales del sistema. Pero esto no se daba porque lo que existía eran grandes volúmenes de información que había que leer por completo y que acarreaban una serie de problemas de: monolitismo, redundancia, ambigüedad e imposibilidad de mantener [Morales, 95]**.**

Antes de explicar en que consiste el Análisis Estructurado, veamos algunas definiciones simples de análisis: la primera nos explica que es el estudio realizado para separar las distintas partes de un todo [Larousse, 98]. Y la segunda nos plantea que es el análisis es el estudio de un problema, antes de realizar alguna acción. [DeMarco, 79]

Aunque la primera es una definición completamente válida, es más tradicional, mientras que la segunda se aproxima más a nuestros intereses, es decir, al análisis con respecto al proceso de desarrollo de software. Más propiamente, el análisis aplicado al desarrollo de software, se puede considerar como el estudio de una aplicación o área de negocios que se dirige a la especificación de un nuevo sistema [DeMarco, 79].

Se han ideado distintas maneras de realizar este análisis, pues aunque en teoría es un proceso similar al del análisis en cualquier otro campo, ya sea abstracto o concreto, en la computación se deben tener muchas consideraciones con respecto al trabajo que se desea realizar. Como respuesta a esta necesidad han aparecido diferentes métodos de análisis en el desarrollo de software, de los cuales uno de los más aceptados ha sido el Análisis Estructurado, creado por Tom DeMarco.

Según [DeMarco, 79], El principal objetivo del Análisis Estructurado es el proveer a los constructores del sistema el Documento de Especificación Estructurada (DEE), el cual es el resultado del uso de distintas herramientas, principalmente los Diagramas de Flujo de Datos (DFD) y los Diccionarios de Datos (DD). El DEE se complementa mediante el uso de Lenguaje Estructurado, Tablas de Decisión y Árboles de Decisión.

La secuencia metodológica planteada por Tom DeMarco [DeMarco, 79]. Es la siguiente:

- **Definir la lista de eventos**

  Desarrollar una lista de requerimientos en lenguaje natural.
- **Producir un diagrama de contexto**

  Modelar la relación del sistema con el contexto, determinando cuales son las áreas de la empresa que participarán del sistema como fuentes de información.

- **Definir el modelo de Comportamiento**

  Utilizamos el DFD como herramienta modeladora de la transformación de las entradas en salidas.

- **Definir el modelo de datos**

  Modelar la relación de los repositorios de datos con la técnica del Modelo Relacional de Datos. -MRD.

- **Crear el modelo de implementación del usuario**

  Definir los módulos del sistema. En esta etapa son decididos los procesos a ser automatizados; se somete a la evaluación del usuario cada proceso del modelo de Comportamiento.

- **Definir los requisitos de implementación**

  Mientras son definidos los procesos a ser informatizados, se debe discutir y documentar los requisitos de implementación de esos procesos y del sistema de software como un todo: Desempeño, restricciones de costos, restricciones operacionales, consideraciones sobre seguridad y auditoria, tecnología a ser empleada, modificaciones en procedimientos manuales y en otros sistemas informatizadas ya existentes.

- **Elaborar diagramas de estructura.**

  Para cada proceso a ser automatizado, será creado un diagrama de estructura. Las funciones de los diagramas son derivadas de los flujos de datos que entran y que salen de los procesos, y de las transformaciones que generan los datos de salida a partir de los datos de entrada.

- **Integrar los diagramas de Estructura.**

  Los diagramas de estructura deben ser integrados en programas. La estructura del software es completada, incorporándose a él módulos de apoyo operacional, como: módulos de implementación de backups, módulos de control, módulos para alteración de parámetros de operaciones, etc. estos módulos serán incorporados al Diagrama de estructura, donde el acceso a ellos fuese más conveniente.

- **Proyectar la interfaz con el usuario**

  La parte más importante y más compleja de la interfaz con el usuario será desarrollada a partir de los flujos de datos de entrada y de salida de los procesos a ser automatizados. Una única interfaz puede ser generada para atender varios flujos simultáneamente. Las interfaces necesarias a los módulos que implementan menús de selección y a los módulos de apoyo operacional complementaran el proyecto de la interfaz con el usuario.

- **Proyectar la base de datos física**

  Definir las características físicas de cada dato, como el tipo el dominio; la organización de cada archivo, como la definición de las llaves principales, índices, etc.

- **Especificar los módulos.**

  La especificación de los módulos, a través de pseudo código flujogramas u otros.

Aunque esta metodología fue, si se quiere decir de alguna manera, la más importante por un largo periodo de tiempo, es fuerte solo al momento del análisis del funcionamiento del software, en la que se detalla hasta el más mínimo proceso, pero es muy débil al momento del desarrollo y en las etapas siguientes, ya que solo plantean los lineamientos funcionales de la concepción del software.

A mediados de los años 70´s estando el análisis estructurado clásico en su apogeo aparecen una serie de dificultades que limitan al analista hacer un buen desempeño de sus actividades. Entre estos problemas Según [Yourdon, 91] podemos mencionar:

- Distinción difusa y poca, definida entre los modelos lógicos y los modelos físicos.
- Limitación para modelar sistemas en tiempo real.
- El modelo de datos se hacía de una manera primitiva.

Estas y otras razones dieron nacimiento a ciertas mejoras en el análisis estructurado clásico tales como:

Diagramas de entidad-relación, diagramas de transición de estados, división de eventos, modelos esenciales y modelos de implantación. Pero a pesar de esto según Yourdon se siguieron dando más problemas como los siguientes:

- Tras la segunda y tercera correcciones de un diagrama, el analista se volvía cada vez más apuesto y renuente a hacer más cambios.
- Debido a la cantidad de trabajo requerido, el analista dejaba a veces de dividir el modelo del sistema en modelos de menor nivel, quedando por ende, funciones primitivas.

- A menudo no se incorporaban en el modelo del sistema los cambios en los requerimientos del usuario sino hasta después de la fase de análisis del proyecto.
- Inclusive las correcciones de los diagramas había que hacerlas en forma manual, para asegurar que fuesen consistentes y estuviesen completas; lo cual era bastante tedioso y dejaba por fuera muchos errores que debían de encontrarse.

2.2.2 - Orientada a Objetos

La Metodología Orientada a Objetos (MOO), consiste en construir un "**modelo"** de un dominio de aplicación añadiéndole detalles de implementación durante el diseño de un sistema. Según [Farías, 98]**,** Esta metodología consta de cuatro etapas: Análisis, Diseño del Sistema, Diseño de Objetos e Implementación, las cuales se explican a continuación:

Análisis: Es la descripción del problema. El analista construye un modelo de la situación del mundo real que muestra sus propiedades importantes. El modelo del análisis es una "**abstracción"** resumida y precisa de lo *que* debe hacer el sistema deseado y no de la *forma* en que se hará. El modelo de análisis no deberá contener ninguna decisión de implementación.

Diseño del Sistema: En esta fase se toman decisiones de alto nivel acerca de la arquitectura global. El diseño del sistema destino se organiza en subsistemas basados tanto en la estructura del análisis como en la arquitectura propuesta. Se decide qué características de rendimiento hay que optimizar.

Diseño de Objetos: El diseñador de objetos construye un modelo de diseño basándose en el modelo de análisis que lleven incorporados detalles de implementación. El diseñador añade detalles al modelo de acuerdo con la estrategia establecida durante el diseño del sistema. El punto central de esta fase son las estructuras de datos y los algoritmos necesarios para implementar cada una de las clases.

Implementación: Las clases de objetos y las relaciones desarrolladas durante su diseño se traducen finalmente a un lenguaje de programación concreto, a una base de datos o a una implementación en hardware. La programación debería ser una parte relativamente pequeña del ciclo de desarrollo y fundamentalmente mecánica porque todas las decisiones importantes deberán hacerse durante el diseño. El lenguaje destino influye en cierta medida sobre las decisiones de diseño pero éste no debería depender de la estructura final de un lenguaje de programación.

[Farías, 98] también afirma que dentro de la metodología orientada a Objetos se emplean tres clases de Modelos para describir el sistema. Estos modelos son aplicables en la totalidad de las fases del desarrollo y van adquiriendo detalles de implementación a medida que progresa el desarrollo. Estos tres modelos son:

- Modelo de Objetos: describe la estructura estática de los Objetos del Sistema y también sus relaciones.

- Modelo Dinámico: Describe las interacciones que existen entre los Objetos, es decir, los aspectos de un sistema que cambian con el tiempo.

- Modelo funcional: Describe las transformaciones de los datos del sistema, es decir, las transformaciones de Valores de datos que ocurren dentro del sistema.

Los tres modelos, según [Farías, 98]**,** son parte ortogonales de la descripción del sistema completo y estan enlazados entre si. Sin embargo, el más importante es el modelo de Objeto porque es necesario para describir qué esta cambiando o transformandose entes de describir cuando o como cambia.

La base principal de la Metodología Orientada a Objetos, es la de **"MODELAR"** el sistema en base a Objetos, según [Romero, 98] uno de los métodos más resaltantes es el propuesto por [Rumbaugh et al, 91] conocido como la Técnica de Modelado de Objetos (OMT) este es considerado ampliamente como uno de los sistemas de análisis orientados a objetos más completos que se han publicado.

OMT consta de tres fases o actividades principales: Análisis, Diseño de Sistemas y Diseño de Objetos.

El Análisis presupone que existe una especificación de los requisitos y se desarrolla construyendo tres modelos distintos mediante el uso de tres notaciones diferentes.

El diseño de Sistemas se realiza organizando los objetos en subsistemas identificando la concurrencia a partir del modelo dinámico (DM), asignando subsistemas a procesadores o tareas, diciendo si los datos deben o no estar almacenados en archivos, en memoria o en un sistema de administración de base de datos, diciendo el uso de periféricos, y recursos globales.

El Diseño de Objetos implica transformar la información del DM y del modelo funcional (FM) en operaciones de modelo objeto (OM), los pasos restantes consisten en:

1. Diseñar algoritmos.
2. Optimizar vías de acceso.
3. Realizar el control.
4. Ajustar estructuras.
5. Indicar los detalles de los atributos.
6. Empaquetar las estructuras en módulos.
7. Escribir el informe de diseño, incluyendo un OM, DM, y FM detallados.

El OMT tiene la intención de ser un método tanto para el análisis como para el diseño, pero, aún cuando contiene un método bastante completo para el análisis, solamente tiende a dar indicaciones prácticas para el diseño. El OMT abarca más temas que la mayoría de los demás métodos, pero sigue siendo incompleto en algunos aspectos y resulta muy complejo aprender y utilizar sus notaciones [Romero, 98].

Entre otros, existe también el Método de Booch [Romero, 98], el cual comienza por un análisis de flujo de datos, que se utiliza entonces como ayuda para identificar objetos, buscando tanto objetos concretos como objetos abstractos en el espacio del problema, que se encontraran a partir de las burbujas y almacenes de datos en el diagrama de flujo de datos (DFD).

La notación y método de diseño revisados de Booch consta de 4 actividades principales y 6 notaciones. Los primeros pasos tratan los aspectos estáticos del sistema, tanto en su aspecto lógico como en su aspecto físico.

Estructura lógica

- Diagramas de clases
- Diagramas de objetos

Estructura física

- Diagramas de módulos
- Diagramas de procesos

Dinámica de clases

- Diagramas de transición de estados

Dinámica de instancias

- Diagramas temporales

2.2.3 - Basada en Componentes

La Metodología basada en componentes (MBC), según [Santaolaya et al, 98], es aquella que se basa en la manera de implementar sistemas utilizando componentes previamente programados y probados. La construcción de esos componentes se realiza mediante la programación orientada a componentes.

La Programación Orientada a Componentes (POC) aparece como una variante natural de la programación orientada a objetos (POO) para los sistemas abiertos, en donde la POO presenta algunas limitaciones; por ejemplo, la POO no permite expresar claramente la distinción entre los aspectos computacionales y meramente composicionales de la aplicación, no define una unidad concreta de composición independiente de las aplicaciones (los objetos no lo son, claramente), y define interfaces de bajo nivel como para que sirvan de contratos entre las distintas partes que deseen reutilizar objetos [Garrido, 04].

La MBC, según [Santaolaya et al, 98], nace con el objetivo de construir un mercado global de componentes software, cuyos usuarios son los propios desarrolladores de aplicaciones que necesitan reutilizar componentes ya hechos y probados para construir sus aplicaciones de forma más rápida y robusta. Las

entidades básicas de la POC son los componentes, desarrollados en el sentido que son como procesos terminados listos para su utilización, es decir cajas negras que encapsulan cierta funcionalidad y que son diseñadas para formar parte de ese mercado global de componentes, sin saber ni quién los utilizará, ni cómo, ni cuándo. Los usuarios conocen acerca de los servicios que ofrecen los componentes a través de sus interfaces y requisitos, pero normalmente ni quieren ni pueden modificar su implementación [Garrido, 04].

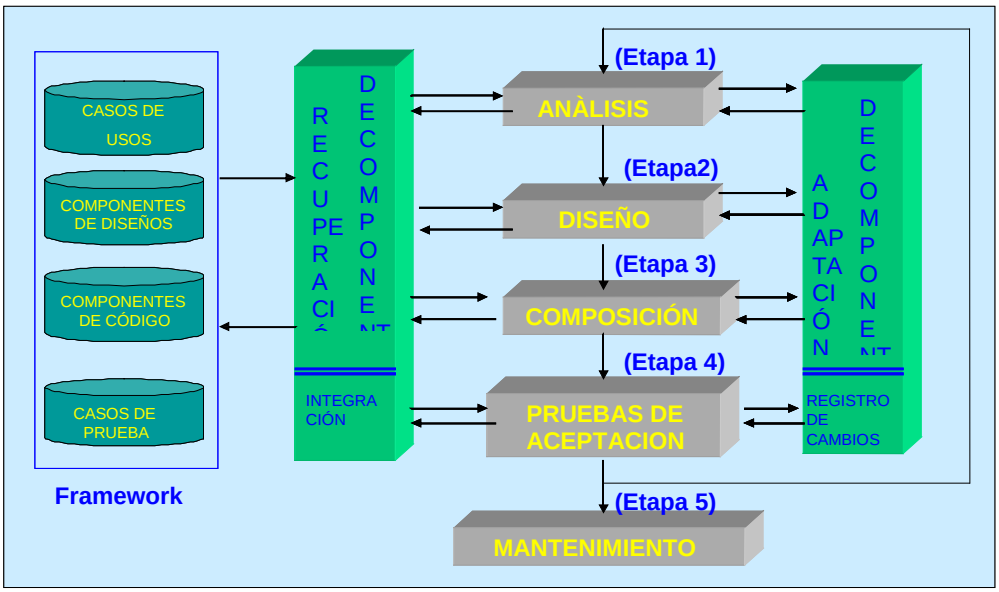

FIGURA 1: Ciclo de Desarrollo de Sistemas con Componentes [Santaolaya et al, 98]

En la figura 1 se muestra la metodología de desarrollo de sistemas basados en componentes propuesta por [Santaolaya et al, 98] (también llamado ciclo de desarrollo de sistemas basado en componentes). Este ciclo es muy similar a otros tales como el de cascada [Amo et al, 95] [Somerville et al, 98], evolutivo [Amo et al, 95], de prototipos[Amescua et al, 95], [Somerville et al, 98], etc., ya que todos contemplan actividades tales como la identificación de requerimientos, la construcción de un diseño, la implementación, y las pruebas de un sistema, sin embargo a diferencia de esos modelos, este introduce actividades de búsqueda, recuperación, selección e integración de componentes. Como se puede observar en la figura 1, la etapa de análisis de requerimientos se puede llevar a cabo apoyada en consultas dirigidas a una base en la que los componentes pueden ser casos de uso ó escenarios, el producto de esta etapa es una especificación de requerimientos y la definición de componentes de diseño que cumplen con esos requerimientos.

El diseño se refiere a la especificación de las entidades, atributos, comportamiento y a las relaciones que existen entre estos, los cuales en conjunto formarán la arquitectura del nuevo sistema. En esta etapa se realiza, mediante consultas a la base de componentes de diseño, el cual puede estar poblado de los patrones de diseño del dominio, en contraste con los tradicionales modelos de referencia en que la implementación se realiza por la escritura textual de código en un lenguaje de programación.

En la etapa 3, la generación de código del nuevo sistema se realiza por composición, es decir, mediante la consulta, recuperación e integración del código asociado a cada uno de los componentes de la arquitectura definida en la etapa de diseño.

Después de implementado un sistema bajo el enfoque de [Santaolaya et al, 98], se realiza las pruebas de aceptación, las cuales permiten certificar la eficiencia del sistema. Las pruebas también pueden llevarse a cabo apoyándose en casos de prueba documentados, predefinidos, y existentes en la base de componentes, asociados a especificaciones de requerimientos de sistemas similares.

Una parte medular del modelo de referencia, es el contar con el marco de trabajo que define un dominio de aplicación, su especificación y arquitectura, y que administre los artefactos de reutilización, incluyendo componentes de especificación, diseño, código, documentación y planes de prueba; para la generación automática de nuevas aplicaciones de software a partir de los depósitos de estos componentes.

3 – Agentes y Metodologías Basadas en Agentes

3.1 – Agentes

Al igual que ocurre con la propia definición de la IA, desde el punto de vista conceptual se pueden encontrar diversas propuestas en la literatura sobre definiciones del concepto de *agente,* entre las que se pueden destacar:

El Diccionario de la Lengua Española [Real Academia Española, 92] indica que la palabra agente proviene del latín. agens, -entis, p. a. de agere, hacer. Y realiza la siguiente definición de agente: 1. adj. Que obra o tiene virtud de obrar. 2. m. Persona o cosa que produce un efecto. 3. Persona que obra con poder de otro.

El Diccionario [Webster, 96] define la palabra *agent (agente)* con las siguientes acepciones traducidas al español: (a). Una persona autorizada por otra a actuar en sus asuntos. (b). Lo que o quien actúa o tiene el poder de actuar. (c). Una fuerza natural u Objeto producido o usado para obtener resultados específicos. (d). Una causa activa; una causa eficiente.

El Glosario de Tecnología de Agentes del Grupo de Administración de Objetos (OMG) [Thompson et al, 99] destaca la siguiente definición: Un agente o agente software es un programa software que hace algo, generalmente en nombre de una persona o de otro agente,

- Posiblemente automatizando alguna tarea (por ejemplo indexando),
- Posiblemente usando cierta inteligencia (por ejemplo planificando o negociando).
- Posiblemente activado por algún evento que es ejecutado como tarea de fondo (por ejemplo seleccionando y filtrando información de interés).
- Posiblemente moviéndose de un sitio a otro (para recopilar información),
- Posiblemente comunicando o interactuando con el usuario (a través de charla o diálogo),
- Posiblemente comunicando o interactuando con otros agentes de una forma coordinada y cooperativa.
- Posiblemente aprendiendo y cambiando su comportamiento a lo largo del tiempo (en respuesta a cambios en el entorno).

- Posiblemente operando con iniciativa propia y generando informes periódicos.

Por otro lado, [Franklin et al, 96] indican que un agente autónomo es un sistema dentro de un entorno, que siente el entorno y actúa dentro de él siguiendo su propia agenda para conocer el efecto de lo que sentirá en el futuro.

Además, [Franklin et al, 96] señalan una apreciación tomada de Hayes-Roth en la destaca que los agentes inteligentes (o racionales) realizan continuamente tres funciones: Percepción de las condiciones dinámicas de un entorno; acción que afecta a las condiciones del entorno; y razonamiento para interpretar percepciones, resolver problemas, realizar inferencias y determinar acciones.

Desde un punto de vista más pragmático [Franklin et al, 96] también señalan la definición dada por el Grupo de Agentes Inteligentes de la IBM, citado también en [Giret et al, 00], en la que se dice que los agentes racionales son entidades software que desempeñan una serie de operaciones en nombre de un usuario o de otro programa, con algún grado de independencia o autonomía, empleando algún conocimiento o representación de los objetivos o deseos del usuario.

Sin que ninguna de ellas haya sido plenamente aceptada por la comunidad científica, quizás la más simple sea la propuesta por [Russell, 96], quien considera a un agente como una entidad que percibe y actúa sobre un entorno (**figura 2, tomada de** [Russell, 96]). Basándose en esta definición, se pueden caracterizar distintos agentes de acuerdo a los atributos que posean y que van a definir su comportamiento [Botti et al, 99] para resolver un determinado problema.

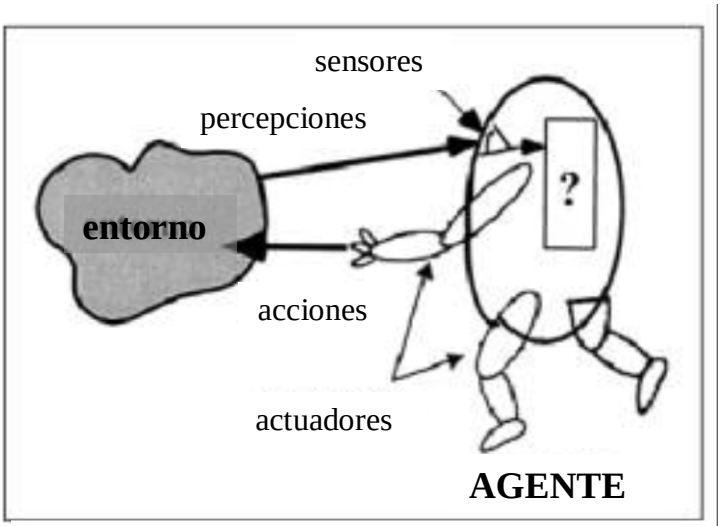

Figura 2: Visión esquemática de un agente racional

FIGURA 2: Visión Esquemática de un Agente Racional [Russell, 96]

Según [Botti et al, 00] un agente va a estar caracterizado por una serie de calificativos, los cuales vienen a denotar ciertas propiedades a cumplir por el agente. Esto nos lleva a plantear otra definición bastante aceptada de agente donde se emplean tres calificativos que se consideran básicos. Esta definición ve a un agente como un sistema de computación capaz de actuar de forma autónoma y flexible en un entorno [Wooldridge et al, 95], entendiendo por flexible que sea:

- **Reactivo:** el agente es capaz de responder a cambios en el entorno en que se encuentra situado.

- **Pro-activo:** a su vez el agente debe ser capaz de intentar cumplir sus propios planes u objetivos.

- **Social:** debe de poder comunicarse con otros agentes mediante algún tipo de lenguaje de comunicación de agentes.

Como se observa en este caso, se han identificado una serie de características o apellidos, que de por sí debe tener un agente. Atendiendo a esta idea, para poder asociar a una herramienta el término agente, esta debe ser capaz de cumplir los requerimientos anteriormente expuestos. Para el año 2000, un pequeño porcentaje del software de propósito general existente se adaptaba a dicha definición, esto según [Botti et al, 00].

Algunas de las características que en la literatura se suelen atribuir a los agentes en mayor o menor grado para resolver problemas particulares y que han sido descritos por autores tales como Franklin [Franklin et al, 96], y Nwana [Nwana, 96], estas son:

- **Continuidad Temporal**: Se considera un agente un proceso sin fin, ejecutándose continuamente y desarrollando su función.

- **Autonomía**: Un agente es completamente autónomo si es capaz de actuar basándose en su experiencia. El agente es capaz de adaptarse aunque el entorno cambie severamente. Por otra parte, una definición menos estricta de autonomía sería cuando el agente percibe el entorno.

- **Sociabilidad**: Este atributo permite a un agente comunicar con otros agentes o incluso con otras entidades.

- **Racionalidad**: El agente siempre realiza lo correcto a partir de los datos que percibe del entorno.

- **Reactividad**: Un agente actúa como resultado de cambios en su entorno. En este caso, un agente percibe el entorno y esos cambios dirigen el comportamiento del agente.

- **Pro-actividad**: Un agente es pro-activo cuando es capaz de controlar sus propios objetivos a pesar de cambios en el entorno.

- **Adaptabilidad**: Está relacionado con el aprendizaje que un agente es capaz de realizar y si puede cambiar su comportamiento basándose en ese aprendizaje.

- **Movilidad**: Capacidad de un agente de trasladarse a través de una red telemática.

- **Veracidad**: Asunción de que un agente no comunica información falsa a propósito.

- **Benevolencia**: Asunción de que un agente está dispuesto a ayudar a otros agentes si esto no entra en conflicto con sus propios objetivos.

No existe un consenso sobre el grado de importancia de cada una de estas propiedades para un agente. Sin embargo, se puede afirmar que estas propiedades son las que distinguen a los agentes de meros programas. Según la definición de Wooldridge, las características de autonomía, reactividad, pro-actividad y sociabilidad son las características básicas. Se pueden encontrar otras definiciones donde varían ligeramente las características a aplicar a un agente básico.

Tal como indica el Dr. H. Van Dyke Parunak en su trabajo [Parunak, 99], un agente es como una navaja del ejército suizo en el que se puede ver la definición básica como sólo la navaja y en el que si se necesita algún accesorio más se le añade y, si no se necesita, no hay necesidad de acarrear con todos los accesorios (**figura 3**).

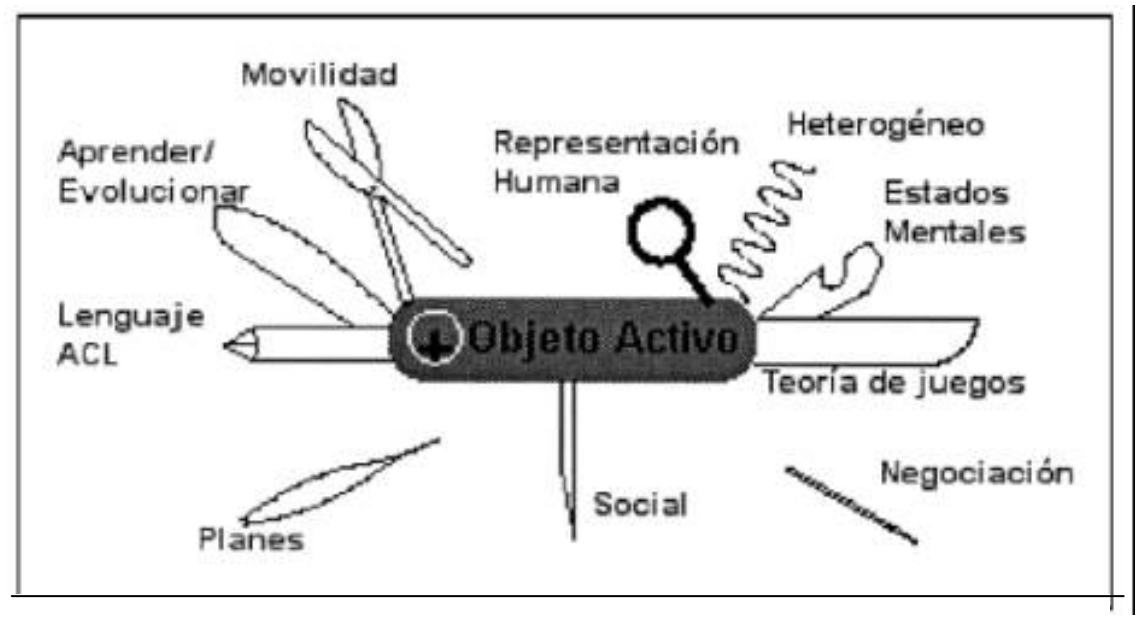

FIGURA 3: Navaja Suiza de Parunak, representando los conceptos que puede integrar un agente [Parunak, 99]

3.2 – Sistemas MultiAgentes (SMA)

La mayor parte de problemas que intenten solucionarse con la tecnología de agentes podrían ser solventados con un solo agente. Sin embargo, lo mismo que el concepto de programación descendente, modular y orientada a objetos intenta dividir los problemas en subproblemas cada vez más sencillos de resolver, en el ámbito de los agentes se huye de la idea del agente omnipotente, por el contrario se tiende al planteamiento en el que múltiples agentes autónomos interactúen, se coordinen (cooperando, compitiendo o ambas cosas) y todo ello de una forma adaptativa para resolver dicho problema. Es por ello que los agentes no son desarrollados de forma independiente, sino como entidades que constituyen un sistema. A este tipo de sistema se le denomina **Multiagentes** [Huhns, 98]. En este caso los agentes deben o pueden interactuar entre ellos.

Formalmente se puede decir que, un conjunto de agentes que interactúan entre sí para lograr un objetivo en común se conoce como un sistema multiagente (SMA) cooperante [Ramos, 95], el cual se encuentra inmerso en un medio ambiente de trabajo. Cada uno de los agentes del entorno tendrá sus propios objetivos, tomará sus propias decisiones y podrá tener la capacidad de interactuar y comunicarse con el resto de agentes [Cugola et al, 96], tener en cuenta lo que realiza cada uno de ellos y razonar acerca del papel jugado por los diferentes agentes que constituyen el sistema.

El [OMG, 98] define un sistema multiagente como una plataforma que puede crear, interpretar, ejecutar, transferir y terminar agentes. Como características principales de los SMA pueden destacarse las siguientes [Jennings et al, 98]:

- Cada agente individual del sistema tiene un conocimiento limitado del mismo, es decir, los agentes no disponen de información del sistema completo.

- No existe un control global de todo el sistema por lo que los datos quedan descentralizados.

- La computación es asíncrona en lo que respecta a la interacción entre los diversos elementos del SMA.

- Deben permitir la interoperación con otros sistemas existentes, aunque ésta es de las tareas más complejas en la actualidad.

Sin embargo, hay que destacar también que para poder soportar SMA es necesario disponer de un entorno adecuado para su desarrollo. Este tipo de entornos debe disponer al menos de las siguientes características:

- Contener agentes con las capacidades de autonomía, adaptabilidad, interacción y coordinación.

- Partir de un planteamiento abierto donde se huya de un diseño centralizado.

- Ofrecer una infraestructura donde se pueda especificar claramente la comunicación y los protocolos de interacción entre agentes y con el propio entorno.

3.3 - Metodologías Orientadas a agente

La tecnología de agentes ha recibido gran atención en los últimos años y, como resultado, la industria está comenzando a interesarse en esta tecnología para desarrollar sus propios productos. Grandes empresas como IBM, Microsoft, Mitshubishi o BT, según [Iglesias, 98]**,** cuentan ya con laboratorios de agentes inteligentes y han sacado los primeros productos al mercado, principalmente de

agentes de usuario e Internet. Sin embargo, para que el paradigma de la programación orientada a agentes se extienda para desarrollar aplicaciones genéricas, es necesario que se desarrollen técnicas de análisis y diseño con este paradigma.

La investigación en metodologías orientadas a agentes es un campo incipiente. Debido a la relación del paradigma de la orientación a agentes con la orientación a objetos y con los sistemas basados en conocimiento, las metodologías orientadas a agentes (OA) no han surgido como metodologías totalmente nuevas, sino que se han planteado como extensiones tanto a metodologías orientadas a objetos (OO) como a metodologías de ingeniería del conocimiento (ICO) [Iglesias, 98].

3.3.1 – Vista como Extensiones de Metodologías OO

Vistas como Extensiones de metodologías OO, se pueden enunciar diversas razones que justifican la apreciación como una extensión de dichas metodologías. En primer lugar, hay notables similitudes entre el paradigma orientado a objetos y el paradigma de agentes [Burmeister, 96]. Desde los primeros tiempos de la IAD, fue establecida la estrecha relación entre la tecnología multiagente y la programación orientada a objetos concurrente. Tal y como enunció Shoham en su propuesta de programación orientada a agentes (AOP) [Shoham, 93], los agentes pueden considerarse como objetos activos, esto es, objetos con un estado mental. Ambos paradigmas comparten el paso de mensajes para comunicarse, y el empleo de herencia y agregación para definir su arquitectura. Las principales diferencias [Shoham, 93] radican en que los mensajes de los agentes tienen un tipo predeterminado (su acto comunicativo) y en que el estado mental del agente se basa en las intenciones para la cual fue programado.

Otra posible razón proviene del empleo habitual de los lenguajes orientados a objetos para implementar sistemas basados en agentes porque se consideran un entorno natural de desarrollo [Bond et al, 88].

También se puede citar como razón la popularidad de las metodologías orientadas a objetos. Muchas metodologías orientadas a objetos han sido empleadas en el desarrollo de software con éxito, como es el caso de RUP, Proceso Unificado Racional y OMT, Técnica de Modelado de Objetos [Rumbaugh et al, 91]. La experiencia del empleo de estas metodologías puede facilitar la adopción de metodologías de agentes que las extiendan, es decir, que la experiencia y difusión de las metodologías orientadas a objetos facilitarían el aprendizaje y comprensión de los diagramas de las metodologías orientadas a agentes que las extiendan.

En cuanto a las tres vistas comúnmente empleadas para analizar un sistema en las metodologías orientadas a objetos, también son interesantes para describir a los agentes: una *vista estática* para describir la estructura de los objetos/agentes y sus relaciones; una *vista dinámica* para describir las interacciones entre objetos/agentes; y una *vista funcional* para describir el flujo de datos entre los métodos/tareas de los objetos/agentes.

Por último, algunas de las técnicas utilizadas para identificar objetos son propicias para ser aplicadas en la identificación de agentes. En concreto, tienen especial interés las técnicas de *casos de uso* [Jacobson et al, 92] y *tarjetas CRC* (Clase-Responsabilidad-Colaboración) [Wirfs-Brock et al, 90].

A pesar de las similitudes entre los paradigmas de la orientación a objetos y a agentes, obviamente, los agentes no son simplemente objetos. Por tanto, las metodologías orientadas a objetos no abordan estos aspectos diferenciadores [Burmeister, 96] [Kendall et al, 96], tales como:

Los objetos tienen una estructura simple (atributos y métodos) mientras que los agentes tienen una estructura compleja. Una arquitectura de agente puede ser comparada a un módulo de las metodologías orientadas a objetos, y puede ser analizada e implementada siguiendo una metodología orientada a objetos.

Tanto los objetos como los agentes emplean paso de mensajes para comunicarse, mientras que en los objetos el paso de mensaje se traduce en invocación de métodos, en los agentes este paso de mensaje se suele modelar como el intercambio de un conjunto predeterminado de actos de "habla", con protocolos asociados para negociar o responder a cada acto comunicativo. Además, los agentes realizan un procesado de los mensajes, y pueden decidir ejecutar, o no, la acción correspondiente al mensaje recibido.

Otro aspecto diferenciador consiste en que las metodologías orientadas a objetos no proporcionan técnicas para modelar cómo los agentes llevan a cabo sus inferencias, su proceso de planificación, etc. Por último, los agentes se caracterizan por su dimensión social. Es necesario definir procedimientos para modelar las relaciones "sociales" de los agentes.

3.3.2 – Vista como Extensiones de Metodologías ICO

Según [Iglesias, 98] Las metodologías de ingeniería del conocimiento pueden proporcionar una buena base para modelar sistemas multiagente, ya que estas metodologías tratan del desarrollo de sistemas basados en conocimiento. Dado que los agentes tienen características cognitivas, estas metodologías pueden proporcionar las técnicas de modelado de la base de conocimiento de los agentes.

La definición del conocimiento de un agente puede considerarse un proceso de adquisición de conocimiento, y dicho proceso sólo es abordado por éstas metodologías.

La extensión de metodologías de conocimiento puede aprovechar la experiencia adquirida en dichas metodologías. Además se pueden reutilizar las bibliotecas de métodos de resolución de problemas y ontologías así como las herramientas desarrolladas en estas metodologías.

Iglesias también plantea que los aspectos no abordados de ambas metodologías son que la mayor parte de los problemas planteados en las metodologías de ingeniería del conocimiento también se dan, obviamente, en el diseño de sistemas multiagente: adquisición del conocimiento, modelado del conocimiento, representación y reutilización. Sin embargo, estas metodologías conciben un sistema basado en conocimiento como un sistema centralizado. Por tanto, no abordan los aspectos distribuidos o sociales de los agentes, ni en general su conducta proactiva, dirigida por objetivos [Iglesias, 98].

CAPITULO III

MARCO METODOLOGICO

Tipo de Investigación

Esta investigación se inserta dentro del criterio y modalidad de investigación monográfica documental [Camacho et al, 02], ya que tiene como objetivo dar apoyo al estudio descriptivo y diagnóstico de una situación inherente al tema sobre metodologías de Agentes y Multiagentes, debido a que su desarrollo esta enfocado en determinar analogías y discrepancias teórico-Practicas de las metodologías empleadas en la construcción de agentes, esta investigación se basara principalmente en fuentes bibliográficas, documentales y estudios comparativos. En tal sentido, permitirá construir un cuadro comparativo para determinar la aplicabilidad de una metodología de agentes en algún caso en particular.

Diseño de la Investigación (Enfoque del Análisis)

La presente investigación se desarrollara en tres fases a seguir: a) Recopilación y revisión bibliográfica de los tópicos relacionados en esta investigación, es decir, análisis bibliográfico sobre las metodologías para la construcción de Agentes y Multi-Agentes, específicamente las que tengan que ver con las metodologías: **BDI** (Belief Desire Intention), **GAIA**, **MaSE** (Multiagent

Systems Engineering), MAS Common KADS y Cassiopeia , b) Desarrollo individual de cada metodología resaltando los aspectos más significativos relacionados a su definición, ciclo, tipo de agentes a las que se relaciona, etc., y la fase c) Construcción del análisis comparativo entre las metodologías seleccionadas, basándose en los resultados obtenidos en la fase (b), resumido en cuadros o tablas de fácil interpretación.

En la parte (a) se consulta en libros, publicaciones e Internet, con el propósito de obtener la mayor información posible sobre las 5 metodologías a estudiar (**BDI** , **GAIA**, **MaSE**, MAS Common KADS y Cassiopeia.), en la mayoría de los casos solo se recolectara información y se anotará su respectiva fuente para efectos de bibliografía y referencias. Posteriormente se procederá a la revisión, para valorar y determinar la información pertinente para la investigación, esto se hace puesto que en Internet se puede conseguir mucha información, en algunos casos repetidas o en otros sin basamento, y en el mejor de los casos demasiada información casi imposible de poderla revisar a cavalidad, lo que amerita una revisión detallada para la selección oportuna de la información a utilizar.

En la fase (b) se procederá con desarrollo teórico individual de cada metodología, explicando en que consiste y resaltando los aspectos más significativos relacionados a su definición, tipo de agentes a las que se diseñaron (agentes o multiagentes), ciclo o etapas que la conforman, procesos involucrados en ella, lo pragmática que resulte la metodología, así como los aspectos o comentarios a favor o en contra de cada metodología hechos por otros autores o que se obtengan en base al estudio que se realizara.

Por último, en la fase (c) se procederá a la construcción del cuadro sinóptico, para ello se analizara cualitativamente en una primera instancia, los estudios realizados que traten sobre las comparaciones que pueden haber hecho otros autores en relación al tema, en una segunda instancia el análisis cualitativo será en base a los resultados obtenidos en la fase (b) lo que permitirá determinar las posibles semejanzas y/o diferencias que existan entre el grupo de las 5 metodologías seleccionadas y agregarlas en el cuadro resultante de la primera parte de la fase (c).

CAPITULO IV

RESULTADOS

Desarrollo de las Metodologías

1 - Metodología de Mas-CommonKADS

La metodología MAS-CommonKADS se maneja como una extensión de la metodología CommonKADS para integrar conceptos de sistemas multiagentes y técnicas de la metodología orientadas a objetos con el fin de generar un modelo alternativo que permita desarrollar aplicaciones empleando tecnología de agentes.

1.1 - Definición de CommonKADS

CommonKADS es una metodología que se aplica para el análisis y la construcción de Sistemas Basados en el Conocimiento (SBC) [Perez, 96], es decir, sistemas computarizados que contienen el conocimiento y el razonamiento de un dominio específico con el objetivo de dar solución a problemas en la misma forma como lo podría hacer un experto humano.

Cuando se habla de sistemas de conocimiento, se tiene el ingrediente del conocimiento. Para hacer un proyecto de este estilo se debe tener en cuenta tanto el conocimiento para el desarrollo de proyectos en general, como también la experiencia que se tiene en los sistemas de conocimiento. CommonKADS proporciona las herramientas para poder llevar a cabo todo lo anterior. Es así como se han definido ciertos niveles para manejar el conocimiento y la información de un sistema en un entorno de la organización. Además, en cada uno de los niveles se cuenta con unos modelos que permiten, por medio de unos formularios, hacer el análisis detallado de los procesos, la información y el conocimiento.

CommonKADS es una metodología que refleja el modelo de espiral, favoreciendo el enfoque de administración de proyectos y presenta un modelo más flexible que el modelo de caída en cascada y más controlado que el desarrollo por prototipos. Además, para su planteamiento también se han tenido en cuenta otras metodologías para el desarrollo de sistemas de información computarizados, tales como el análisis y diseño estructurado, y la orientación a objetos. Adicionalmente, en CommonKADS se han trabajado factores que se manejan en algunas prácticas y teorías administrativas como la teoría de las organizaciones, la reingeniería de procesos, la administración por proyectos y la gerencia de la calidad [Henao, 00].

1.2 - MAS-CommonKADS

La metodología denominada *MAS-CommonKADS* [Iglesias, 98], es una extensión de la metodología *CommonKADS*, añadiendo los aspectos que son relevantes para los sistemas multiagente (MAS) e integrando técnicas de las metodologías orientadas a objeto para facilitar su aplicación.

1.3 - La Metodología de MAS-commonKADS

1.3.1 - Modelos de MAS-CommonKADS

MAS-CommonKADS según [Iglesias, 98], propone los siguientes modelos para el desarrollo de sistemas multiagente (ver Figura 4):

- ***Modelo de Agente (AM):*** específica las características de un agente: sus capacidades de razonamiento, habilidades, servicios, sensores, efectores, grupos de agentes a los que pertenece y clase de agente. Un agente puede ser un agente humano, software, o cualquier entidad capaz de emplear un lenguaje de comunicación de agentes.

- ***Modelo de Organización (OM):*** es una herramienta para analizar la organización humana en que el sistema multiagente va a ser introducido y para describir la organización de los agentes software y su relación con el entorno.

- ***Modelo de Tareas (TM):*** describe las tareas que los agentes pueden realizar: los objetivos de cada tarea, su descomposición, los ingredientes y los métodos de resolución de problemas para resolver cada objetivo.

- ***Modelo de la Experiencia (EM):*** describe el conocimiento necesitado por los agentes para alcanzar sus objetivos. Sigue la descomposición de *CommonKADS* y reutiliza las bibliotecas de tareas genéricas.

- ***Modelo de Comunicación (CM):*** describe las interacciones entre un agente humano y un agente software. Se centra en la consideración de factores humanos para dicha interacción.

- ***Modelo de Coordinación (CoM):*** describe las interacciones entre agentes software.

- ***Modelo de Diseño (DM):*** mientras que los otros cinco modelos tratan del análisis del sistema multiagente, este modelo se utiliza para describir la arquitectura y el diseño del sistema multiagente como paso previo a su implementación.

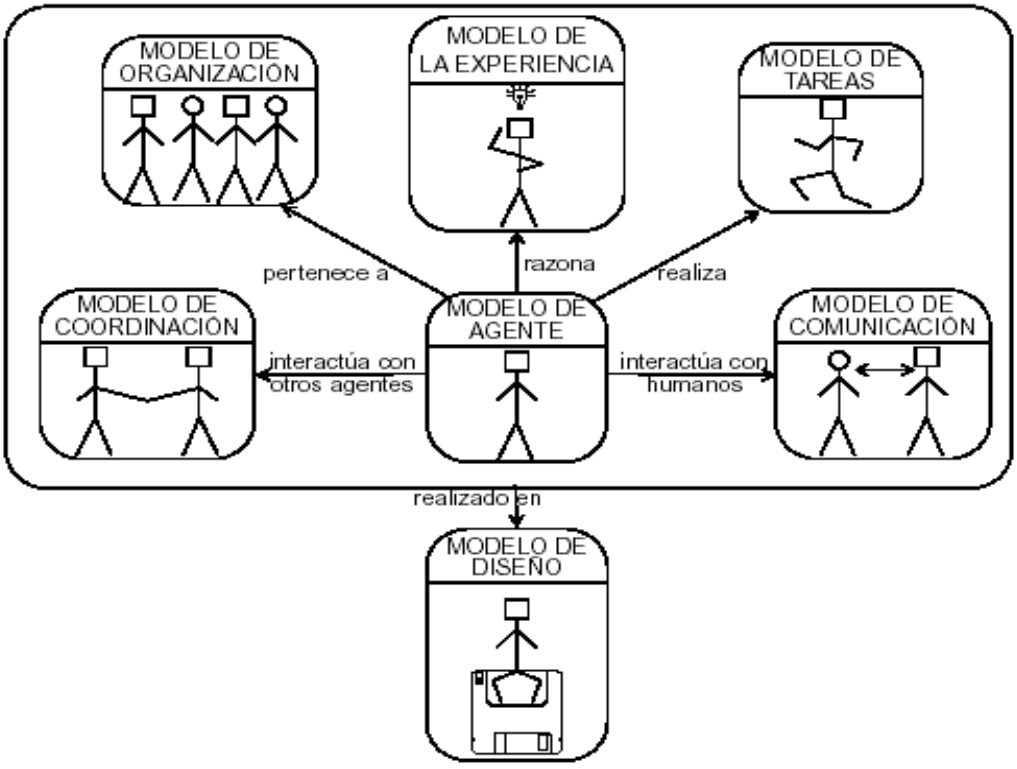

FIGURA 4: Los modelos de MAS-CommonKADS [Iglesias, 98]

1.3.2 - Fases de desarrollo de MAS- CommonKADS

El ciclo de vida propuesto para desarrollar estos modelos es la metodología del modelo en espiral dirigido por riesgos [Boehm, 93], siguiendo la gestión de proyectos de *CommonKADS* [Hoog et al, 94]. El modelo de ciclo de vida para el desarrollo de sistemas multiagente en *CommonKADS,* consta de las siguientes fases:

- ***Conceptuación*:** tarea de elicitación[1] para obtener una primera descripción del problema y la determinación de los casos de uso que pueden ayudar a entender los requisitos informales [Potts et al, 94] y a probar el sistema.

- ***Análisis*:** determinación de los requerimientos de nuestro sistema partiendo del enunciado del problema. Durante esta fase se desarrollan los siguientes modelos: organización, tareas, agente, comunicación, coordinación y experiencia.

- ***Diseño*:** determinación de cómo los requisitos de la fase de análisis pueden ser logrados mediante el desarrollo del modelo de diseño. Se determinan las arquitecturas tanto de la red multiagente como de cada agente.

- ***Codificación* y *prueba* de cada agente.**

- ***Integración*:** el sistema completo es probado.

- ***Operación* y *mantenimiento***.

[1] Traducción de "*elicitación*": extracción o adquisición de conocimiento, según [Iglesias, 98].

1.3.3- Modelo de proceso software para proyectos pequeños

Los proyectos pequeños, desarrollados por una única persona y con un número reducido de agentes, pueden emplear un ciclo de vida convencional en el que se incluye la utilización de componentes y, según [Iglesias, 98], resumir los "pasos" de la metodología de la manera siguiente:

Conceptuación

El objetivo de esta fase es comprender mejor cuál es el sistema que desea el cliente. Los principales resultados de esta fase serán una identificación de los objetivos que debe satisfacer el sistema desde el punto de vista del usuario, junto con la identificación de los actores que interactúan con el sistema.

Análisis

El resultado de esta fase es la especificación del sistema compuesto a través del desarrollo de los modelos descritos anteriormente. Sólo las extensiones a *CommonKADS* serán desarrolladas en esta sección. Los pasos de esta fase son:

- **Estudio de viabilidad:** el modelo de organización permite modelar la organización en que va a ser introducido el sistema multiagente, para estudiar las áreas posibles de aplicación de sistemas inteligentes, su viabilidad y su posible impacto.

- **Delimitación:** delimita el sistema multiagente de los sistemas externos**.** El desarrollo del modelo de organización habrá delimitado la interacción del sistema multiagente con el resto de sistemas de la organización.

Los sistemas externos (predefinidos) deben ser encapsulados en agentes, modelados mediante el desarrollo de ejemplares del modelo de agente, y modelando sus interacciones mediante el desarrollo de modelos de coordinación. En el caso de que haya interacción con agentes humanos, esta interacción se describe en el modelo de comunicación.

- **Descomposición:** el sistema se analiza mediante el desarrollo solapado de varios puntos de vista:

    - Descomposición funcional: se descompone el sistema en funciones (tareas u objetivos) que deben ser realizados, mediante el desarrollo de un modelo de tareas global.

    - Descomposición en ejecutores: el sistema se descompone en agentes que realizan las funciones anteriormente desarrolladas, mediante el desarrollo del modelo de agente.

- **Descripción de las interfaces:** tomando como punto de partida la fase de conceptuación y el modelo de agente, se describen las relaciones estáticas que determinan los canales de comunicación con otros agentes y con el exterior mediante el desarrollo del modelo de la organización de los agentes. Es necesario identificar los flujos de comunicación de la organización, que serán motivados, principalmente, por la necesidad de información y para evitar o resolver conflictos.

- **Identificación y descripción de interacciones:** la identificación y descripción de las relaciones dinámicas entre los agentes se realiza de la siguiente manera:

- Las interacciones dinámicas con otros agentes software se describen en el modelo de coordinación.

- Las interacciones dinámicas con agentes humanos se describen en el modelo de comunicación.

- **Descripción del razonamiento de los agentes:** para cada agente, debemos modelar el conocimiento que necesita para llevar a cabo sus objetivos, mediante el desarrollo del modelo de la experiencia.

- **Validación:** Cada vez que un agente es descompuesto en nuevos agentes, éstos deben ser consistentes lógicamente con la definición previa:

  - Los subagentes son responsables de los objetivos del agente.

  - Los subagentes deben ser consistentes con el modelo de coordinación y mantener las mismas interacciones externas.

  Y cumplir con la validación cruzada con los otros modelos y los conflictos detectados en los escenarios deben tener determinado, al menos, un método de resolución de conflictos.

Diseño

Como resultado de la fase de análisis, un conjunto inicial de agentes ha sido determinado, así como sus objetivos, capacidades e interacciones. Durante esta fase se desarrolla el Modelo de Diseño, que se basa en extender el modelo homónimo de *CommonKADS* para diseñar sistemas multiagente. El Modelo de Diseño recopila los modelos desarrollados en el análisis y se subdivide en tres actividades:

- *Diseño de la aplicación.* El sistema es descompuesto en subsistemas. Para una arquitectura multiagente, se determina la arquitectura más adecuada para cada agente[2].

- *Diseño de la arquitectura.* En nuestro caso, se selecciona una arquitectura multiagente (en vez de, por ejemplo, una pizarra o una descomposición orientada a objetos). Proponemos que en este punto se determine la infraestructura del sistema multiagente (el denominado *modelo de red* [Iglesias et al, 96]). Durante el diseño de la arquitectura se definen los agentes (denominados a*gentes de re*d) que mantienen dicha infraestructura.

- *Diseño de la plataform*a. Se especifica el software y hardware que se necesita (o está disponible) para el sistema.

Codificación y prueba

Las dos tendencias principales son el uso de un lenguaje de propósito general y la utilización de un lenguaje formal de agentes, que puede ser directamente ejecutable o traducible a una forma ejecutable.

[2] El término arquitectura de un agente significa describir una construcción hardware/software concreta y modelo de agente describe el conjunto de requisitos (habilidades, papel en la organización, etc.) del modelo de agente de CommonKADS.

Integración y prueba global

Se puede comprobar parcialmente la corrección de la conducta global del sistema utilizando los escenarios típicos que tratan los posibles conflictos y los métodos de resolución de los mismos. Pero, dado que la conducta global del sistema no puede ser determinada durante la fase de análisis o diseño, porque depende de los acuerdos y compromisos concretos realizados entre los agentes [Huang et al, 95], normalmente se necesitará recurrir a la simulación.

Operación y mantenimiento

Una vez probado el sistema, puede ponerse en operación. La filosofía de los agentes facilita el mantenimiento del sistema dada su naturaleza modular.

1.3.4 - Modelo de proceso software para proyectos grandes

La gestión del proyecto de *MAS-CommonKADS* adopta la definida en *CommonKADS* [Hoog et al, 94], que sigue un ciclo de vida en espiral, dirigido por riesgos. En la primera fase de cada ciclo se identifican los objetivos de dicho ciclo y se establece qué productos deben desarrollarse para satisfacer dichos objetivos; en la segunda se identifican los riesgos asociados tanto a los objetivos como a la realización de productos, y se ordenan dichos riesgos por orden de prioridad; el tercer paso es construir un plan de trabajo, definiendo actividades para realizar los productos y asignando recursos a dichas actividades; la última fase de cada ciclo consiste en realizar los planes y supervisar los criterios de calidad establecidos.

En *MAS-CommonKADS* los productos se corresponden con estados alcanzados de un modelo. Estos estados se asocian a objetivos y riesgos. Es necesario, por tanto, establecer los estados del modelo de coordinación para que pueda ser gestionado.

Definición de estados de MAS- CommonKADS

La definición de los estados de los modelos de *CommonKADS* fue establecida en [Hoog et al, 94], posteriormente al desarrollo de la mayoría de los modelos, que no han sido revisados y por tanto no se ajustan a dicha definición. En [Hoog et al, 94], se define el "estado" a partir de los siguientes atributos:

- **Modelo:** nombre del modelo de *MAS-CommonKADS* para el que se define el estado (modelo de agente, modelo de tarea, modelo de la experiencia, modelo de organización, modelo de comunicación, modelo de coordinación, modelo de diseño).

- **Variable del estado:** nombre del aspecto del modelo cuyo estado es de interés. Dicho aspecto depende de la granularidad escogida para gestionar un modelo. Normalmente será el nombre de un constituyente del modelo o de una relación del modelo.

- **Valor:** Valor de la variable del estado. Este valor puede definirse de una forma cuantitativa (p. ej. 33% del modelo realizado), declarativa (empleando un lenguaje de representación formal, semiformal o informal) o cualitativa. Se ha preferido la descripción cualitativa, definiendo el siguiente conjunto de etiquetas [Aben, 93], [Arsène et al, 02]:

    - *Vacío:* ningún trabajo ha sido hecho.

- *Requisitos identificados:* los requisitos que parten de otras partes del modelo se han listado.

- *Restricciones identificadas:* las restricciones de la estructura interna se han clarificado.

- *Entradas identificadas:* las fuentes de información necesarias para construir los componentes del modelo han sido documentadas.

- *Identificado:* los rasgos de los componentes del modelo han sido listados.

- *Descrito:* los componentes del modelo han sido descritos en detalle.

- *Verificado:* la consistencia interna y corrección de los componentes del modelo ha sido establecida.

- *Validado:* los componentes del modelo han sido comprobados con criterios (externos) de validación.

- *Complet*o: el modelo ha sido descrito y validado con respecto a su completitud.

- *Descartado:* los trabajos adicionales han sido cancelados.

- *Aprobado:* la especificación de los componentes del modelo ha sido aceptada y aprobada por el cliente.

- **Roles:** los estados pueden desempeñar los siguientes roles:

    - *Intermedio:* empleado sólo para supervisión interna.

    - *Hito interno (landmark):* empleado para revisión en la conclusión de un ciclo de gestión del proyecto.

    - *Hito externo (milestone):* estado hito interno en que se entrega un producto al cliente.

    - *Obligatorio:* estado que debe realizarse en todo proyecto para asegurar la calidad del proyecto.

    - *Definido por el usuario:* cualquier estado que el usuario de la metodología desee definir.

- **Dependencias entre estados:** representa cómo un estado está relacionado con otros estados. Pueden darse dependencias internas (entre elementos de un modelo) o externas (entre elementos de modelos diferentes).

- **Métricas de calidad del estado:** En la práctica, se emplean simplificaciones en la definición de un estado [Avouris et al, 92]:

    - Se omite la descripción cuantitativa y declarativa de los estados.

    - La designación de estado hito (interno o externo) o definido por el usuario se deja al gestor del proyecto. Salvo que se indique que el estado es obligatorio, se considera que un estado es intermedio.

- Los modelos sólo emplean cuatro estados posibles: vacío, identificado, descrito y validado, con el siguiente significado:

  - *Vacío:* no hay información disponible sobre el componente del modelo.

  - *Identificado:* los "nombres" de los principales componentes se han definido, por ejemplo, los nombres de los agentes del sistema, pero aún no se ha concretado su significado.

  - *Descrito:* se han descrito los "nombres" previamente identificados.

  - *Validado:* hay una fuerte evidencia de que la información es correcta.

- Se asume una relación de orden entre los valores posibles de un estado: "descrito" requiere "identificado", y "validado" requiere "descrito".

Aproximaciones al desarrollo de los modelos

Se distinguen los siguientes aspectos para caracterizar el desarrollo de un modelo:

- *Número de ejemplares del modelo desarrollado*s. Normalmente en *MAS-CommonKADS* se suelen contemplar dos situaciones:

  - Desarrollar dos ejemplares de un modelo: Uno describiendo la situación antes de la introducción del SBC y después de su introducción.

- Desarrollar un ejemplar del modelo, describiendo una situación que incluye al SBC.

- *Criterios empleados para comenzar el desarrollo del modelo.* Se distinguen tres posibilidades [Wærn et al, 93]:

  - El modelo es un "motor del proyecto". Esto ocurre si la actividad de modelado es iniciada como el medio principal para abordar un riesgo concreto.

  - El modelo es desarrollado "bajo demanda". Esto ocurre si el modelo es desarrollado para abordar riesgos secundarios de otros modelos que son motores del proyecto o como una actividad lateral en el desarrollo de otros modelos.

  - Ninguna. El modelo no se desarrolla ya que los riesgos que puede abordar no son significativos y pueden ignorarse en el desarrollo del proyecto.

- *Nivel de interacción entre el modelo y la construcción de otros modelo*s. La dependencia puede ser:

  - *Ningun*a. El modelo puede ser construido como una actividad independiente y separada.

  - *Algun*a. El desarrollo del modelo puede descomponerse en subactividades identificables que se pueden construir como actividades separadas e independientes con relaciones identificables con otras subactividades de modelado.

- *Complet*a. El desarrollo del modelo no se puede separar de otras actividades de modelado.

1.4 – Comentarios

Esta metodología ha sido la primera en incorporar la idea de *proceso de ingeniería* en el sentido de [Pressman 93]. 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-CommonKADS 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 conque 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.

2 - Metodología Gaia

Esta metodología, según [Zambonelli et al, 01]**,** enfoca el análisis y diseño de los SMA como la construcción de organizaciones computacionales. Para Gaia los SMA son sistemas de agentes compuestos de múltiples entidades interactuando, sociedades organizadas de individuos, en las cuales cada agente desempeña uno o más roles específicos. La estructura del SMA es definida en términos de un modelo de roles, en el cual se identifican los roles que puede ejecutar cada agente y los protocolos de interacción.

Sin embargo cuando se tratan de agentes interesados en sí mismos deben introducirse las reglas de la organización, la estructura organizativa y los patrones organizacionales.

Las reglas organizacionales, expresan los requerimientos globales para la creación y ejecución del SMA. Una estructura organizativa define las clases específicas de organización y el régimen de control a los cuales los agentes y roles tienen que adaptarse en orden de que todo el SMA trabaje eficientemente y de acuerdo a sus requerimientos específicos. Los patrones organizacionales expresan las estructuras que pueden ser rehusadas de sistema en sistema.

En [Zambonelli et al, 01], los autores distinguen entre dos clases de SMA: (a) sistemas distribuidos para la resolución de problemas, en los cuales los agentes componentes son explícitamente diseñados para cooperar en el logro de una meta global; (b) sistemas abiertos, en los cuales los agentes, no necesariamente codiseñados para compartir conocimiento común, pueden entrar y salir del sistema en forma dinámica. En el primer caso, todos los agentes son conocidos a priori y todos los agentes se suponen que son benevolentes unos con otros, por lo tanto, pueden fiarse unos de otros durante sus interacciones. En el segundo caso, la llegada dinámica de agentes desconocidos requiere ser tomada en cuenta, así como la posibilidad de comportamientos egoístas, - interesados en sí mismos -, en el curso de las interacciones.

El diseño de aplicaciones paralelas y distribuidas, usualmente depende en una arquitectura que se deriva de la descomposición de datos y funcionalidades requeridas por el sistema para lograr la meta u objetivo y en la definición de sus interdependencias. En SMA, sin embargo, el comportamiento proactivo y autónomo de los agentes integrantes, sugiere que la aplicación puede ser diseñada emulando el comportamiento y estructura de organizaciones humanas. Por lo tanto, a cada agente se le asigna un rol en el sistema, esto es, una tarea - responsabilidad bien definida en el contexto del sistema completo, que el agente debe lograr de una manera autónoma, sin un control centralizado. En este modelo, las interacciones no son meramente expresión de interdependencia, más bien son vistas como medios para los agentes de lograr su rol en la organización. Por lo tanto, las interacciones están bien definidas y localizadas en el rol como tal, y entonces ayudan a caracterizar la posición del agente en la organización.

La perspectiva organizacional también puede hacer menos compleja y más fácil la administración que muchas de las metáforas tradicionales de los sistemas concurrentes. (a) Cada agente se convierte en un foco separado de control, a cargo de lograr su rol y siendo completamente responsable por él. (b) Como los agentes tienen

empotrada la mayor parte de su funcionalidad, ellos necesitan lograr sus roles, la interdependencia entre los sistemas es reducible. Estos aspectos facilitan el proceso de diseño porque conducen a una separación entre el nivel de componentes y el nivel del sistema.

En muchos casos SMA son para soportar y controlar organizaciones del mundo real. En estos casos el diseño basado en organización reduce la distancia conceptual entre el software y el sistema en el mundo real que tiene que soportar, simplificando consecuentemente el desarrollo del sistema.

En la medida que aumenta la complejidad los principios de modularidad y encapsulamiento sugieren la división en diferentes suborganizaciones. Un SMA complejo puede ser visto como varias organizaciones interactuando.

Rol definido en términos de tareas a ser ejecutadas en el contexto de la organización, le otorga al agente una posición bien definida, con un conjunto de comportamientos asociados.

Para lograr su rol en la organización el agente debe interactuar con otros agentes para intercambiar conocimiento y coordinar sus actividades.

2.1 - Modelos de la Metodología Gaia

La metodología Gaia, según [Wooldridge et al, 00], contiene los modelos siguientes.: a) en la fase de Análisis: Modelo de Roles, Modelo de Interacciones; b) en la fase de diseño: Modelo del Agente, Modelo de Servicios, Modelo de trato, correspondencia, entre los agentes

2.1.1 – Fases de Análisis

En [Wooldridge et al, 00] se explica que para entender el sistema y su estructura, se captura la organización del sistema como una colección de roles, que se muestran en ciertas relaciones y que toman en sistemáticos patrones de interacción entre los roles.

La entidad más abstracta en nuestro concepto de jerarquía es el sistema, usada en la forma clásica, aunque tiene un significado relacionado en este tipo de sistema basado en agentes para indicar a la sociedad o la organización. Pensamos que un sistema basado en agentes es una sociedad artificial o una organización artificial.

En una organización con realización concreta los roles están instanciados en los individuos. Las instancias no son estáticas, tampoco hay relaciones uno a uno entre roles e individuos. Pueden existir varios individuos que toman un solo rol.

Los roles están definidos por los atributos: responsabilidades, permisos, actividades, protocolos. La Responsabilidades determinan la funcionalidad y son quizás el atributo clave asociado con el rol. Las responsabilidades son de dos tipos de propiedades de progreso y propiedades de seguridad.

Para ejecutar sus propiedades un rol tiene un conjunto de permisos, los permisos son derechos asociados con los roles en orden de realizar sus responsabilidades.

Las actividades de un rol son los cómputos asociados con el rol que puede ser ejecutado por el agente sin interactuar con otros agentes. Las actividades son acciones privadas de los agentes.

Un rol está identificado por un número de protocolos, los cuales definen la forma como interactúa con otros roles.

El modelo de la organización comprende el modelo de los roles y el modelo de interacción.

Modelo de roles:

Atributos: permisos, derechos asociados con el rol. Responsabilidades del rol. La responsabilidad define la funcionalidad del rol. Nombre del rol.
Actividades: métodos.
Protocolos: actividades que requieren interacción con otro agente.

Modelo de Interacción:

Existen dependencias y relaciones inevitables entre los diferentes roles en una organización de múltiples agentes. Dado este hecho las interacciones requieren obviamente ser capturadas y representadas en la fase de análisis. Consiste de definiciones de protocolo, una por cada tipo de interacción entre roles. Cada protocolo puede ser visto como un patrón institucionalizado de interacción. Esto es un patrón de interacción formalmente definido y abstraído de cualquier secuencia de pasos de ejecución, De esta forma el énfasis puede ser colocado en la esencia propia y naturaleza de la interacción, en lugar de orden en que se intercambias mensajes particulares. Esto significa que un protocolo indicará los mensajes de intercambio en tiempo de ejecución.

Un protocolo de contiene los atributos siguientes:

- **Propósito:** descripción de la naturaleza de la comunicación.

- **Iniciador:** rol responsable de iniciar la interacción.

- **Receptor:** rol con el cual el iniciador interactúa.

- **Entradas:** información usada por el rol iniciador cuando activa el protocolo.

- **Salidas:** información suministrada por él al receptor durante el curso de la interacción.

- **Procesamiento:** descripción del cualquier protocolo de procesamiento que el iniciador procesa durante el curso de la interacción.

2.1.2 – Fases de Diseño

Consisten en transformar los modelos del análisis, en niveles suficientes de abstracción para que técnicas tradicionales de diseño puedan ser aplicadas en la implementación de los agentes [Wooldridge et al, 00].

Gaia se aplica a la sociedad de los agentes, como cooperan para lograr las metas del sistema y lo que se requiere de cada individuo para lograrlo. Como el agente realiza el servicio dependerá del dominio de aplicación y no se contempla en Gaia.

Encontramos en [Wooldridge et al, 00], que en el diseño, el modelo del agente identifica los tipos de agente que se ejecutarán en el sistema y las instancias que habrá de estos tipos. El modelo de servicios identifica los servicios que son requeridos para la realización del rol del agente. Finalmente el modelo de trato documenta las líneas de comunicación entre los diferentes tipos de agente.

Modelo de Agentes:

Un tipo de agente es un conjunto de roles. Puede existir una correspondencia uno a uno entre roles y tipos de agente. Sin embargo este no es necesariamente el caso. Por ejemplo se puede empaquetar un número de roles relacionados en el mismo tipo de agente. Esto mejorará la eficiencia.

El modelo del agente es un árbol en el cual los nodos hojas corresponden a roles y otros nodos corresponden a tipos de agentes. La herencia aplica en el modelo de agentes.

Modelo de Servicios:

Identifica los servicios asociados con cada agente en cada rol y especifica las propiedades principales de los servicios. Un servicio es una función de un agente. Servicio es un bloque coherente de actividad en la cual un agente debe engranarse. Se identifican las entradas, salidas, precondiciones, postcondiciones de cada servicio.

Modelo de trato:

Define las formas de comunicación entre los tipos de agentes. No definen los mensajes a ser enviados, ni cuando, simplemente indican cuales son las rutas de comunicaciones. Identifica cuellos de botella en la comunicación que pueden causar problemas en tiempo de ejecución.

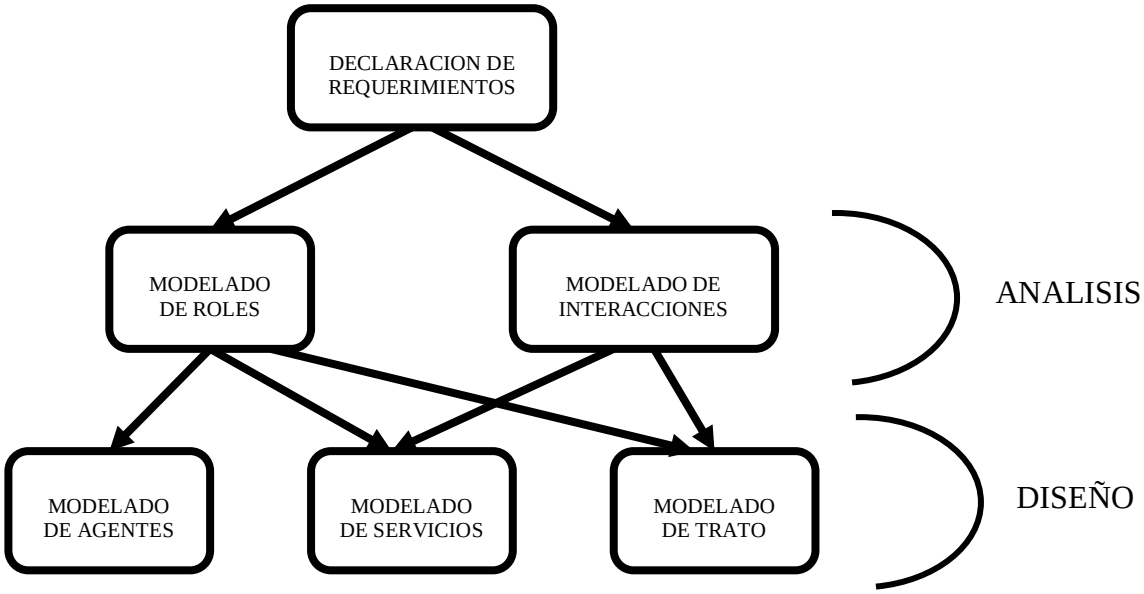

FIGURA 5: Relación entre los Modelos de Gaia [Wooldridge et al, 00]

2.2 - Comentarios

GAIA Utiliza el concepto de Rol para analizar el sistema. Define los roles en forma muy detallada; cada Rol se expresa en términos de responsabilidades y permisos, dando una definición más completa que otras metodologías.

A pesar de presenta pasos metodológicos bien definidos y acotados, no hay una noción clara de cómo pasar de una etapa a otra, es decir, existen ciertos pasos que requieren más ayuda para su resolución. Supone definir roles e interacciones (en forma de protocolos) sin ayuda de algún modelo intermedio, como pueden ser Casos de Uso y Diagramas de Secuencias.

En general, los modelos no son muy adecuados y cómodos para trabajar, se tienen diagramas que no agregan demasiada información y diagramas muy literales, es decir, presentan mucha información, pero pierden el sentido gráfico.

El sistema total debe contener un número comparativamente pequeño de diversos tipos del agente (menos de 100). [Wooldridge et al, 00]

3 - Metodología BDI

Las arquitecturas que ve el sistema como un agente racional, que tiene ciertas actitudes mentales de Creencias (Expresan las expectativas acerca del estado actual del mundo y acerca del curso de acción para lograr ciertos efectos), Deseos (noción abstracta que representa las preferencias específicas sobre estados del mundo futuro o cursos de acción) e Intenciones (conjunto de metas seleccionadas conjuntamente con sus estados de procesamiento). Como la definición de deseo es bastante débil, el agente debe seleccionar el subconjunto de deseos que puede conseguir, por lo tanto las metas denotan las opciones que el agente actualmente tiene. Los planes no forman parte de la teoría de creencias-deseos-intenciones, pero son fundamentales para la implementación pragmática de las intenciones. Estas actitudes mentales determinan el comportamiento del sistema.

Esta es comúnmente conocida como arquitectura 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.

3.1 - La Metodología

La metodología de diseño para sistemas multiagentes que emplean la arquitectura BDI es planteada en [Kinny et al, 96] y [Kinny et al, 97], y es la que se presenta a continuación.

Esta técnica de modelado emplea clases de objeto e instancias para describir diferentes clases de entidades dentro de un sistema multiagente en diferentes niveles de abstracción. Incluye dos puntos de vista del sistema, uno **externo** que captura los agentes que los conforman, sus responsabilidades, servicios, información que requiere y sus interacciones con otros elementos del sistema, y un punto de vista **interno** que recoge los elementos o estructuras que conforma cada agente.

3.1.1 - Vista Externa del Sistema

Requiere que identifiquemos los roles principales del sistema y sus relaciones. Y con esto poder identificar las clases de agente que existen y su jerarquía, los agentes entonces son instancias particulares de estas clases. Un análisis de las responsabilidades de cada agente permitirá identificar sus servicios proporcionados y

usados y de esto ultimo sus interacciones externas. La creación y la duración de los roles en conjunto con sus interacciones nos permitirán determinar las relaciones de control existentes entre las distintas clases de agente.

Para que el diseñador logre esta perspectiva del sistema debe cumplir una serie de etapas o pasos que se resumen en:

- Identificar los roles del dominio de la aplicación, estos sirven como punto de partida para el análisis de los posibles agentes que integran el sistema.

- Por cada rol detectado identificar sus responsabilidades y los servicios que proporciona y usa, para cumplirlas.

- Por cada servicio, identificar las interacciones asociadas y las primitivas que se necesitan para lograrlas, igualmente identificar la información que ellos contienen.

- Refinar la jerarquía de los agentes.

Los roles, las responsabilidades y los servicios describen los comportamientos del sistema en distintos niveles de abstracción, los roles se pueden ver como un conjunto de responsabilidades, y las responsabilidades como un conjunto de servicios.

La Vista externa del sistema multiagente se expresa por dos modelos:

> ***Modelo de Agentes:*** Describe las relaciones jerárquicas entre las distintas clases de agentes identificadas, ya sean clases abstractas o concretas, Señala las instancias de agentes que pueden existir dentro del sistema, la multiplicidad y el momento en que ellos dejan de existir.

Modelo de Interacción: Describe las responsabilidades de una clase de agente, los servicios que proporciona, las interacciones asociadas y además las relaciones de control entre las clases de agentes. Especifica la semántica y la sintaxis de los mensajes que usan los agentes para comunicarse unos con otros y para comunicarse con otros componentes del sistema, tales como las interfaces de usuario.

El Modelo de Agentes tiene 2 componentes:

Modelo de Clases de Agentes:

Conjunto de diagramas de clases, que define los agentes abstractos y concretos, captura las relaciones de herencia y agregación entre ellos.

Es un diagrama acíclico y dirigido que contiene nodos que denotan clases abstractas y concretas. Las clases se representan por un rectángulo y las clases abstractas son distinguidas por un símbolo A en la sección superior de este rectángulo.

Se permite la herencia múltiple. Cuando un agente es una subclase de otro agente este refina las creencias, objetivos y planes que hereda de su superclase. La agregación en el diagrama denota la incorporación dentro de un agente de subagentes. La instanciación dentro del diagrama puede expresar conceptos como, cuando un agente puede crearse y cuantas veces este puede ser instanciado.

Modelo de Instancias de Agentes:

Conjunto de diagramas de instancias que identifican las instancias de los agentes. Los sistemas que contienen un pequeño número de agentes, las clases y las instancias pueden ser representados en un único diagrama de clases de agentes.

Define el conjunto de agentes estáticos – instanciados en tiempo de compilación - y el conjunto de agentes dinámicos - instanciados en tiempo de ejecución.

Cada Instancia de un agente es especificada por un rectángulo con los vértices ovalados encadenado a una clase de agente por un arco punteado dirigido de la instancia del agente a la clase del agente. Los agentes dinámicos son distinguidos por el símbolo S en la parte superior rectángulo ovalado. Las instancias estáticas pueden ser identificadas por un nombre pero las dinámicas tienen que ser diferidas hasta el momento de su instanciación. Se puede usar la notación de multiplicidad en el enlace de instanciación para indicar si una clase dinámica puede ser instanciada varias veces.

El estado mental inicial del agente puede ser especificado con el uso de los atributos initial-belief-state y initial-goal-state, cuyos valores son elementos particulares de las creencias y objetivos de la clase agente.

3.1.2 – Vista Interna Del Sistema

En esta vista del sistema se exponen los componentes internos de los agentes desarrollados bajo la arquitectura BDI, esta cuenta con tres modelos, que en general describen el estado motivacional e informacional del agente y su posible comportamiento. Los modelos en referencia son:

Modelo de Creencias: Describe la información del ambiente en el que se desenvuelve un agente, su estado interno, las clases que lo componen, las acciones que puede ejecutar, las creencias y sus posibles propiedades. Adicionalmente se pueden representar uno o más estados de las creencias – que en si son instancias del conjunto de creencias –, generalmente esto ultimo es usado para definir la condición mental inicial de un agente.

Modelo de Objetivos: Describe los objetivos que pueden ser logrados por un agente y el evento que los dispara. En general consiste de un conjunto de objetivos, donde se establece el dominio de eventos y uno o más estados de estos.

Modelo de Plan: Describe los posibles planes que un agente puede emplear para lograr sus objetivos o responder a los eventos que este percibe.

En estos modelos está tácito las propiedades de ejecución de la arquitectura BDI, que determina exactamente como los eventos y objetivos determinan una intención y como la intención da pasó a las acciones definidas en un plan y posterior a esto se revisan de las creencias y los objetivos. De esta manera se garantiza que las creencias, objetivos e intenciones se desarrollen racionalmente.

Para el desarrollo de estos modelos se comienza con la determinación de los servicios proporcionados por un agente y los eventos e interacciones asociados a estos. Estos definen su propósito y determina los objetivos de más alto nivel asignados. El análisis de los objetivos y su refinación en sub-objetivos conducen naturalmente a la definición de diferentes agentes.

En lo que respecta a los planes, su efectividad y la forma de ejecutarse, depende en general de las creencias que una agente tiene sobre el estado de su ambiente y otro tipo de información disponible.

Esta fase de la metodología puede ser expresada en dos pasos:

- Analizar la manera de lograr un objetivo.
- Construir las Creencias del Sistema.

El diseño de un sistema requiere varias iteraciones, donde el refinamiento de un modelo puede originar retroalimentación en otros, para nuestros efectos la representación interna del sistema retroalimenta a la representación externa, observaremos como la construcción de planes y creencias aclara los requerimientos de información de los servicios, particularmente los que corresponden al monitoreo y notificación. Para mejorar nuestro diseño se pueden emplear escenarios de interacción, que servirán para la redefinición de los servicios.

La metodología estudiada tiene un fin central y es determinar los principales roles, responsabilidades, servicios y objetivos del sistema, estos se convertirán en los elementos de abstracción claves, que facilitaran el manejo de la complejidad. Analizaremos el dominio de la aplicación en términos de lo que se necesita lograr y en que contexto.

El Modelo de Creencias

El modelo de creencias consiste en uno o más estados de creencias. Existe un conjunto de creencias que es especificado por un conjunto de diagramas de objetos, este define el dominio de las creencias de una clase de agente. Un estado de una creencia es un conjunto de diagramas de instancias, que define un caso particular de un conjunto de creencias.

Conjunto de creencias

Conjunto de predicados, funciones y dominios que son derivados directamente de las definiciones y asociaciones de las clases que componen los diagramas de creencia.

Un diagrama de un conjunto de creencias es un gráfico acíclico y dirigido que contiene nodos que representan clases de creencias abstractas y concretas. Las clases de creencias se representan por un icono de clase y las clases abstractas son distinguidas por un símbolo A en la sección superior. La herencia entre clases de creencias son validas y se representan por un arco que va de la clase a la superclase terminando con un triángulo en el extremo adyacente a la superclase. En estos diagramas también se permite la agregación de clases y otro tipo de asociaciones.

Las clases e instancias presentadas en estos diagramas corresponden, en muchos casos, a entidades reales del dominio de la aplicación, pero, a diferencia de un modelo objeto OO, estas definiciones no definen el comportamiento de estas entidades por el contrario, ellos representan las creencias del agente acerca de esas entidades.

Propiedades de una Creencia

Los predicados y funciones que constituyen las creencias de un agente pueden clasificarse como extensionales, que son los predicados que son almacenados como relaciones en la base de datos de creencias y las funciones que son generadas automáticamente de los predicados extensionales que son funcionalmente restringidos, los predicados binarios derivados de la definición de los atributos y asociaciones uno a uno, son evaluables.

Estados de Creencias

Especifica un estado particular de las creencias del agente y puede ser usado para inicializar el estado mental inicial de este. Consiste en un conjunto de instancias de predicados extensionales, estos son especificados a través de un conjunto de diagramas de creencia que contiene instancias de objetos y asociaciones entre ellos.

Las instancias de predicados son derivadas del diagrama de instancias de creencias, este proceso es muy similar al de recoger los predicados de un diagrama de clases de creencias.

Modelo de Objetivos

Cada clase de agente es asociada con un modelo de objetivos, que consiste en un conjunto de objetivos y uno o más estados de estos objetivos. Un conjunto de Objetivos especifica el dominio de los objetivos de una instancia de una clase de agente. Un estado de Objetivo es un conjunto de instancias de elementos del conjunto de objetivos El conjunto de objetivos es definido formalmente por un conjunto de formulas. Cada una de la cual tiene un operador modal que se aplica a un predicado del conjunto de creencias.

Modelo de Planes

Consiste en uno o más planes y representan cómo el agente debe actuar para lograr un objetivo o responder a un evento. Los elementos presentes en los diagramas de plan son tres, un estado de Inicio, un estado final y los estados internos. Existen líneas que denotan las transiciones entre estados. En algunos un estado puede tener otro diagrama, estos tienen el nombre de sub-diagramas.

En el diagrama se pueden expresar eventos, condiciones y acciones que siempre son asociadas a las transiciones de estado. Esto significa que las transiciones entre estados ocurren cuando se disparan los eventos asociados, y la condición es verdadera. Cuando la transición ocurre la acción asociada es ejecutada. Las condiciones por lo general son predicados del conjunto de creencias del agente.

Fallas

Estos diagramas de planes incluyen una notación para las fallas. Estas pueden ocurrir cuando una acción asociada a una transición falla, o cuando existe una transición explícita a un estado de falla o cuando la actividad de un estado activo termina en falla.

Ejecución de un Plan

Siempre la transición inicial en un diagrama de plan debe ser etiquetada con un evento de activación y una condición de activación, que expresa cuando y en que contexto debe activarse el Plan. Este evento de activación puede ser un evento de creencia, o un evento de obtención de objetivo. Si múltiples planes son aplicables a un evento determinado en un contexto especifico, pueden pasar varias cosas, si la activación es producida por un evento de creencia todos son activados en paralelos, pero si la activación ha sido producida por un evento de objetivo entonces su activación es secuencial.

3.2 – 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.

4 - Metodología 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. Dado el enfoque inicial, los agentes se ven como especializaciones de objetos. De hecho, el sistema se construye sobre tecnología orientada a objetos y su aplicación a la especificación y diseño de sistemas multiagente.

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), estos elementos son característicos del UML, esto también sirve para indicar cuántos agentes habrá en el sistema y de qué tipo.

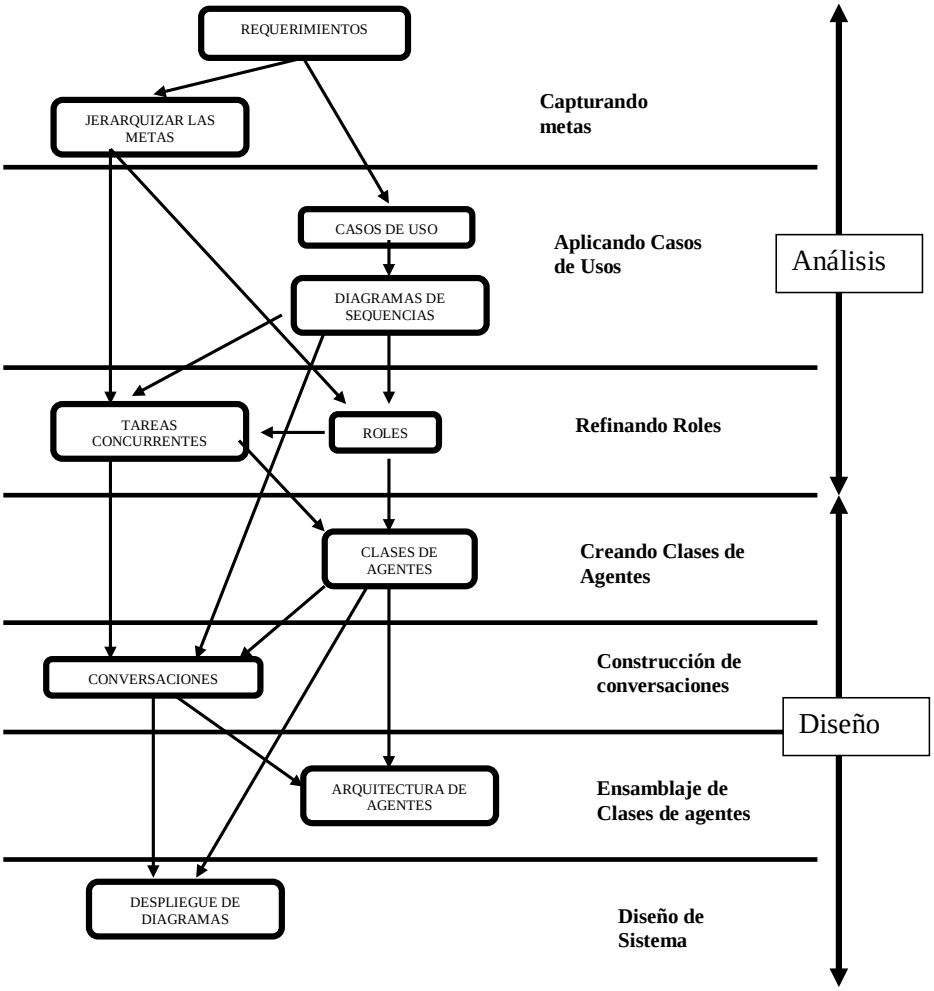

FIGURA 6: DIAGRAMA DE LA METODOLOGIA MaSE [Wood et al, 01]

4.1 - Pasos de la metodología

Según [Scott et al, 02], estos siete pasos podrían explicarse de la siguiente manera:

Capturar las Metas

El primer paso en MaSE es Captura las Metas, que toma la especificación inicial del sistema y lo se transforma en un conjunto estructurado de metas de sistema representado como en una Meta. En MaSE, una meta es un objetivo del nivel de sistema; agentes pueden ser asignados a metas a lograr, pero las metas tienen un contexto dentro del sistema.

Hay dos pasos en Capturar las Metas: identificación de metas y estructuración de las metas. El analista identifica las metas por analizar, cualquier requisito está disponible (por ejemplo, los documentos técnicos detallados, los cuentos de usuario, o especificaciones formales). Una vez que las metas se han capturadas y explícitamente indicadas, son analizadas y estructuradas en un esquema de jerarquía de meta. En un Esquema de la Jerarquía de Meta, Las metas son organizadas por la importancia. Cada nivel de la jerarquía contiene las metas que son aproximadamente iguales en el alcance Y las sub-metas son necesarias para satisfacer las metas del padre. Eventualmente, cada meta se asociará con roles y clases de agentes que son responsables de satisfacer esa meta.

Aplicando Casos de Uso

El paso de Aplicar Casos de Uso es un paso importante en traducir las metas en roles y tareas asociadas. Use Case son dibujados de los requisitos del sistema y son narraciones descripciones de una sucesión de acontecimientos que definen la conducta de sistema deseado. Hay ejemplos de cómo el sistema debe comportarse en un caso dado.

Para ayudar a determinar las comunicaciones actuales requeridas dentro de un sistema, los casos del uso son reestructurados como diagramas de secuencias. Un Diagrama de Secuencias representa una sucesión de acontecimientos entre múltiples roles y, como un resultado, define la comunicación mínima que debe suceder entre roles. Los roles identificados en este paso forman el conjunto inicial de roles utilizados para definir completamente el sistema de roles en el próximo paso y los acontecimientos identificados se utilizan luego para ayudar a definir las tareas y las conversaciones.

Refinar los Roles

El tercer paso en MaSE es asegurarse de haber identificado todos los roles necesarios y para desarrollar las tareas que definen la conducta del rol y modelar la comunicación. Los roles se identifican del Caso de Uso así como las metas del sistema. Aseguremos que todas metas del sistema son justificadas asociando cada meta con un rol específico que es actuado eventualmente por lo menos por un agente en diseño definitivo. Cada meta es trazada generalmente a un solo rol. Sin embargo, hay muchas situaciones donde es útil combinar múltiples metas en un solo rol por conveniencia o eficiencia. Las definiciones de roles se capturan en un Modelo uniforme del Rol.

Creando las Clases de Agente

Después que cada tarea sé definida, se Diseña la Fase. En el primer paso, Creando las Clases de Agente, las clases de agentes son identificadas de roles y documentadas en un Diagrama de Clase de Agente. El Diagrama de Clase de agente representa a clases de agente como cajas y las conversaciones entre ellos como líneas conectando las clases de agentes. Al igual que con las metas y los roles, se pueden definir una a una, mapeando entre los roles y las clases de agente; sin embargo, se puede combinar múltiples roles en una sola clase de agente o mapear un solo rol a

múltiples clases de agentes. Desde que los agentes heredan las rutas de comunicación entre roles, cualquier ruta entre dos roles llega a ser conversaciones entre sus respectivas clases. Así, como los roles son asignados a clases de agente, el sistema organizacional es definido.

Construir Conversaciones

Una vez que se ha determinado cómo asignar los roles a agentes, comienza la construcción de Conversaciones. Una conversación define un protocolo de coordinación entre exactamente dos agentes y es modelado utilizando dos Diagramas de Clase de Comunicación, Uno para el iniciador y para uno para el contestador. Un Diagrama de Clase de comunicación es un par de maquinas de estado finito que definen una conversación entre dos clases de agente participantes.

Ensamblar las Clases de Agentes

Después de definir los detalles de cada conversación, el paso al diseño final es definir la implementación en su destinado ambiente usando Diagramas de Despliegue. En las aplicaciones robóticas, los despliegues definen cuales agentes son asignados a cuales robots. A veces, sólo un agente es permitido por robot; sin embargo, si el poder de procesamiento disponible es suficiente, no hay razón de limitar el número de agentes por robot.

Implementación o Diseño de Sistemas

En esta fase se procede a implementar o diseñar el sistema que hemos modelado en las fases anteriores, esta ultima etapa estará enmarcada dentro del lenguaje de desarrollo seleccionado y el hardware necesario para construcción de los agentes robot.

Continuando con [DeLoach 01], el explica que 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 integración de estos diagramas en el proceso de desarrollo parece demasiado simple. Al final, la metodología podría traducirse como *tome la herramienta de soporte y rellene los diferentes apartados*. Esto supone ignorar 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.

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.

4.2 - Comentarios

MaSE usa intensivamente máquinas de estados para describir 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 99] 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 MASCommonKADS.

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 recopilación necesaria de la herramienta completa.

Presenta como desventaja que al contar con una herramienta de desarrollo específica hace que la etapa de implementación de MaSE esté dentro de dicha herramienta y no en forma explícita en la metodología. Por lo que no parece factible elegir cuál herramienta de desarrollo se desea utilizar, siendo obligatorio el uso de AgentTool, y aceptar solo lo que la herramienta brinda.

5 - Metodología Cassiopeia

El método Cassiopeia cuyo propósito es suministrar una referencia metodológica al diseño de SMA. En lo referente al fenómeno colectivo de organización en sociedades de agentes, fue presentado por [Collinot et al, 98] y [Drogoul et al, 98].

El problema está en diseñar agentes de software que puedan desempeñarse con espíritu de equipo, incluso ellos deben tener la capacidad de organizarse a sí mismos para poder lograr un objetivo común. La mayor dificultad es expresar localmente a nivel del agente los comportamientos que permiten obtener el logro de la tarea colectiva del equipo.

El método es primariamente una forma de dirigir un tipo de solución de un problema donde el comportamiento colectivo es puesto en operación a través de un conjunto de agentes. Aunque Cassiopeia no ofrece aún todos los componentes que uno podría esperar para encontrar un método completo, suministra un marco metodológico para entender mejor y planificar el diseño de una organización computacional.

De acuerdo a Cassiopeia un SMA debe ser diseñado en términos de agentes que suministran con tres niveles de comportamientos: elementales, relacionales, organizacionales. Esto permite identificar varios tipos de selecciones hechas en puntos precisos en el proceso de diseño. La modificación del SMA está entonces sistemáticamente relacionada con la revisión de estas selecciones.

El principio subyacente de Cassiopeia es que el problema de organizar individuos para lograr una solución colectiva a un problema debe ser abordado como tal por el diseñador y/o por los agentes. Este tipo de aspecto organizativo se presenta porque las dependencias funcionales son inherentes al logro colectivo de la tarea considerada: La activación de un comportamiento individual, para solucionar un problema, afecta y es afectada por los comportamientos activados por los otros agentes.

Existen dos tipos de acoplamiento:

1. Cuando los comportamientos individuales no compiten, el acoplamiento es estático, esto es, la organización de los agentes permanece incambiable y el diseñador puede definirlos por adelantado.

2. Cuando algún tipo de competencia debe ser manejada, - existen comportamientos individuales que son equivalentes para una situación dada, o el mismo comportamiento puede ser operado por diferentes agentes -, el acoplamiento es dinámico, eso es, la organización no puede ser definida por adelantado puesto que depende del contexto. Por lo tanto el diseñador puede considerar solamente las estructuras organizacionales entre los agentes, las cuales pueden ser instanciadas dentro del contexto de la solución del problema.

5.1 - Pasos de la metodología Cassiopeia

Cassiopeia procede de la definición de la tarea colectiva al diseño del MAS a lo largo cinco pasos, que reconcilian las vistas locales y globales de un MAS: a) La capa de Roles individuales, que contiene la definición de los roles individual requeridos para definir los tipos de agentes. b) Las capas de dependencias, que contiene la definición de las dependencias entre estos roles (funcionales, el recurso de dependencias base o meta-base). c) La capa de roles emparentados, que contiene la definición de la manera que los agentes pueden manejar estas dependencias, desempeñando los roles emparentados dados, que les permiten influenciar otros agentes o elegir cómo ser influenciado por ellos. d) La capa de grupos, que contiene la definición de los grupos potenciales que pueden aparecer, con respecto al

influenciado. e) La capa de organización de los roles, que contiene la descripción de la dinámica de estos grupos, que son los roles de la organización que los agentes tienen que jugar para hacer el mapeo, para desarrollarse o para desaparecer.

Estos cinco pasos pueden ser resumidos en tres etapas:

- Definición de los comportamientos elementales en orden de definir los tipos de agentes.(a)

- Descripción estructural de la organización de los diferentes tipos de agentes. (b y c)

- Descripción de la dinámica organizacional. (d y e)

La primera etapa consiste en listar los comportamientos elementales que son requeridos para lograr la tarea colectiva considerada. La identificación de estos comportamientos resulta de un análisis funcional. El diseñador define diferentes tipos de agentes basados en comportamientos elementales.

La segunda etapa consiste en analizar la estructura de la organización basada en dependencias entre comportamientos elementales. Estas dependencias definen un grafo de acoplamiento subyacente a la tarea colectiva considerada. Las dependencias entre los agentes son elaboradas basadas en este grafo de acoplamiento como sigue: primero el grafo es proyectado entre los diferentes agentes; luego las dependencias inconsistentes son removidas si es necesario, algunas dependencias son ignoradas de acuerdo a las heurísticas disponibles para el dominio de aplicación. Este grafo proyectado refinado contiene las únicas dependencias que son supuestamente relevantes al logro de la tarea. Tales dependencias entre los diferentes tipos de agentes es denominado el grafo de influencias. Las rutas y los circuitos elementales

de este grafo de influencias definen las potencialidades de agrupamiento de los diferentes tipos de agentes, por lo tanto definen una representación de la estructura organizativa.

Basado en este análisis, el diseñador específica los comportamientos relaciones que permiten a los agentes identificar y manejar las influencias. De esta forma, identifica las señales de influencia, las cuales son producidas por cada tipo de agente influyente y específica como un agente influyente toma en cuenta una señal de influencia, esto es, cual comportamiento elemental el agente activa y en que forma.

La tercera etapa aborda la dinámica de la organización. Consiste en especificar el comportamiento que posibilitará al agente manejar la formación, la durabilidad y la disolución del grupo.

Los comportamientos relacionales pueden permitir la formación de varios grupos que son redundantes a su propio fin. Cuando tal redundancia es inútil o costosa, es necesario determinar el criterio que afecta la selección de formar un grupo en lugar de otro. La ocurrencia de grupos redundantes indica medios redundantes para lograr las necesidades de una agente particular, denominado el agente disparado. Tales agentes disparados pertenecen a todos los grupos potenciales logrando esta necesidad y deben evaluarse para decidir cual es el más apropiado en el contexto actual. El diseñador debe por lo tanto: Identificar los agentes disparados, de acuerdo a las potencialidades del grupo que han sido identificados en el grafo de influencia y debe identificar por cada uno de ellos, los métodos de selección permitiendo el control de la formación de grupos. Existe una variedad de técnicas para hacer tal selección, basados en: anuncio declaración de tareas tales como todas las técnicas derivadas de la red de contrato; la noción de consenso y negociación entre agentes pertenecientes a grupos concurrentes; prioridades permitiendo ordenar grupos potenciales; o el uso de un supervisor o jerarquía. Estas técnicas definen el primer tipo de comportamiento organizacional: comportamientos de formación de grupos.

Entonces el diseñador especifica las señales de compromiso, las cuales son producidas por agentes disparados para indicar a otros agentes que un grupo ha sido formado para logra sus necesidades. Estas señales permiten a los agentes que no son miembros del grupo formado, controlar, por ejemplo inhibir, sus comportamientos organizacionales. El diseñador entonces define para cada tipo de agente un segundo tipo de comportamiento organizacional, el cual toma la señal de compromiso en cuenta para organizar el comportamiento organizacional: comportamientos de unión.

Finalmente, las selecciones resultantes de los comportamientos de formación de grupos pueden necesitar ser revisadas. Un grupo deja de existir cuando ha ejecutado sus compromisos; o un grupo puede ser reemplazado por uno más eficiente. El diseñador entonces define para cada tipo de agente un tercer tipo de comportamiento organizacional para disolver el grupo: comportamientos de disolución de grupos.

Análisis Comparativo

1 – Comparaciones Generales

Conceptualmente la mayoría de propuestas introducen, a lo largo de su desarrollo, términos muy similares. De esta forma, conceptos como *objetivo*, *tarea*, *interacción*, *rol*, *responsabilidad*, *organización*, *componente* y evidentemente *agente* aparecen en la mayoría de las metodologías de una forma u otra, y su significado aunque con matices suele ser el mismo. La mayoría de las definiciones son bastante parecidas y representan en el fondo las mismas ideas.

El uso de UML para especificar tanto los aspectos dinámicos como estáticos es frecuente. Se suelen también emplear esquemas (a modo de tablas) para especificar los atributos o características de ciertas entidades (agentes, roles, interacciones).

La existencia de herramientas de desarrollo asociadas no es un aspecto generalizado. MaSE incorpora una herramienta de desarrollo (agentTool) para facilitar el proceso de desarrollo, aunque por el momento no incorpora la totalidad de etapas. En MAS-CommonKADS existe una herramienta conocida como AgentEditor,

En cuanto a posibles limitaciones impuestas por los propios métodos, comentar las restricciones en cuanto el número de clases de agente a desarrollar de MaSE (máximo 10 tipos de Clases diferentes) o GAIA (máximo 100 tipos de

agentes), el resto de métodos no comenta nada al respecto pero es cierto que dada la dificultad del proceso de depuración de un sistema multiagente, un elevado número de agentes complicaría notablemente su realización.

Con respecto al alcance, generalmente las metodologías tienden a cubrir análisis y diseño, excluyendo así a la implementación y mantenimiento. Se puede mencionar el caso de Gaia que no llega a cubrir totalmente al diseño y el de Cassiopeia que no lo plantea. Otra parte que se deja implícita es la captura de requerimientos. Generalmente no queda definido exactamente cómo se debe pasar de la especificación de requerimientos inicial hacia los primeros modelos del análisis. El problema de esto es que no hay una conexión explícita entre los requerimientos y el sistema, con lo cual es difícil darse cuenta cómo se resuelve cada necesidad del usuario.

Las etapas de implementación y mantenimiento no suelen cubrirse por el simple hecho de no existir herramientas productivas y maduras capaces de desarrollar software basado en agentes. Con lo cual, si se quiere llegar más allá del diseño, como MaSE, deben construirse sus propias herramientas (MaSE tiene una herramienta de desarrollo conocida como AgentTool).

MaSE es una metodología académica que cuenta con el soporte de la Fuerza Aérea de los Estados Unidos, las técnicas que utiliza MaSE provienen del paradigma orientado a objetos. También Gaia es una metodología académica, que se compone de una etapa de análisis y otra de diseño. Esta última, sin embargo, no llega a ser lo suficientemente específica como para llegar a la implementación.

CUADRO 1:

ANÁLISIS SINÓPTICO DE LOS ASPECTOS MÁS RELEVANTES DE LAS COMPARACIONES GENERALES

	MAS-CommonKADS	GAIA	BDI	MaSE	Cassiopeia
FASES	- Análisis - Diseño	- Análisis - Diseño	- Análisis - Diseño	- Análisis - Diseño - Implementación	- Análisis
NOTACIÓN	OO	Puede emplear UML	OMT	UML	No propone
ORIENTACIÓN	Ingeniería del Conocimiento	Modelo cognitivo del ser humano	Extensiones OO	Orientada a Objetos	Fenómeno colectivo de organización
LIMITACIONES	Complejidad en el desarrollo	Máximo 100 tipos de agentes		Máximo 10 clases de agente	
ARQUITECTURA AGENTE	Independiente	Independiente	BDI	Independiente	Independiente
HERRAMIENTA DESARROLLO	AgentEditor			AgentTool	
MANEJO DE CONFLICTOS	No se diseñaron para los sistemas que admiten la posibilidad de conflictos				
PLATAFORMA DE DESARROLLO	No se especifica en ninguna metodología la plataforma adecuada para su desarrollo				

2 - Fases de las Metodologías (Análisis y Diseño)

Analizando el ciclo de vida, haciendo énfasis en cada etapa de desarrollo que presentan las diferentes metodologías explicadas con anterioridad, resaltan aspectos significativos en los que coinciden varios autores como [Bonesi, 01], [Botti et al, 99], [Gómez, 03] y otros, y que conviene mencionar, aspectos tales como:

Es de observar que las metodologías expuestas intentan cubrir fundamentalmente las etapas de análisis y diseño de sistemas multiagente. El resto de etapas no son consideradas o bien se deja total libertad para su desarrollo. Salvo MaSE la propone pero no la desarrolla, de este grupo se excluye a Cassiopeia ya que no define una fase de Diseño, sino solo una fase de Análisis centrada en el modelado de la organización.

Un aspecto interesante es el de proveer sugerencias de desarrollo para las fases no consideradas. En este punto, MAS-CommonKADS da pequeñas sugerencias sobre los procesos de implementación e implantación, pero da total libertad al desarrollador de cómo hacerlo. En MaSE por su parte se habla de la generación automática de código como último paso de su propuesta, pero éste no está concretado, ni se refleja en la herramienta de desarrollo que se propone para la metodología.

Analizando por separado las dos etapas que abordan en común las metodologías presentadas, a saber Análisis y Diseño, encontramos los puntos siguientes:

2.1- En el Análisis

Por lo que respecta a la fase de análisis, se pueden destacar cuatro elementos o aspectos que tratan la mayoría de propuestas, éstos son, la identificación de objetivos, modelado de la organización, detección de los agentes necesarios y establecimiento de las interacciones necesarias.

La identificación de objetivos a conseguir en el sistema se repite en todas las propuestas aunque en diferentes vistas o modelos. En algunos casos se desarrolla un modelo específico de objetivos, en MaSE las tareas se construyen entre las fases de captura de objetivos y transformación de roles. En otros casos los objetivos quedan enmascarados dentro de la especificación de otras entidades, por ejemplo como roles en GAIA o tareas MAS-CommonKADS.

El modelado de la organización existente en el sistema dispone de vistas específicas en distintas aproximaciones. En todas ellas se trata de dotar de una estructura organizacional a las entidades que componen el sistema. En ocasiones dicha organización es representada a través de la especificación de los roles en un modelo o vista de roles. En MaSE, no existe un modelado de la organización como tal, aunque la orientación hacia el objetivo del análisis en esta propuesta puede interpretarse como tal, hay que destacar el caso de Cassiopeia puesto que dos de los tres pasos generales de la metodología, son orientados a la dinámica organizacional.

Este análisis de la organización en las metodologías propuestas, está condicionado completamente por una visión funcional del sistema, dejando de lado una visión estructural o arquitectónica del mismo, excepto Cassiopeia que dedica una fase exclusivamente al estudio estructural. El sistema se construye sobre la base de la

funcionalidad que debe tener, especificada por medio de conceptos como: objetivos, roles, responsabilidades, tareas o servicios, dejando de lado la forma o estructura del sistema. La forma de un sistema depende de su propia estructura, condicionada por aspectos como por ejemplo, la plataforma en la que debe funcionar el sistema. Estas dos visiones (función y forma) deben ir construyéndose y evolucionando de forma paralela y deben ser compatibles una con respecto a la otra.

El inicio en el estudio de las interacciones que derivan en posibles cooperaciones o negociaciones entre agentes también suele ser una fase del proceso de análisis en la mayoría de propuestas. El modelado de las interacciones tiene una continuación en todos los casos en el diseño, en la fase de análisis se trata únicamente de identificar y dar forma a las posibles interacciones del sistema. Esto conlleva la existencia de vistas o modelos de interacción que dotan de herramientas para gestionar este aspecto, en GAIA, DBI y MAS-CommonKADS existe un modelo específico. En el caso de MaSE esto se realiza en la fase de aplicación de casos de uso.

Por último, la detección de los agentes existentes en el sistema es lenta entre el proceso de análisis y diseño. La especificación de un agente consiste fundamentalmente en situarlo dentro de la organización del sistema e indicar sus responsabilidades, capacidades y recursos, por medio de la asignación de roles, objetivos y tareas identificados con anterioridad. Un modelo específico de agente existe en cada propuesta, como GAIA (en su fase de diseño), MAS-CommonKADS, MaSE (en el Análisis) y en Cassiopeia que trata de definirlos en la fase de Definición de los comportamientos elementales. En estos modelos se suele emplear una notación gráfica tipo UML junto con esquemas que recojan los aspectos que modelan al agente en cada caso.

2.2- En el Diseño

Por lo que respecta a la fase de diseño, según [Jacobson99] en esta fase se trata de modelar el sistema y encontrar su forma, de tal manera que de soporte a todos los requisitos que se le suponen y que fundamentalmente son resultado de la fase previa de análisis.

En el caso de los sistemas multiagente el diseño se centra en el modelado de la arquitectura del sistema mediante refinamiento de los modelos del análisis, el modelado de los componentes internos de los diferentes agentes y de sus interacciones. Al igual que en la fase de análisis, respecto a estos puntos se puede resaltar lo siguiente:

- El modelado de la arquitectura del sistema viene dado en la mayoría de los casos por una visión general del sistema. Esto se realiza en casi todas propuestas estudiadas (salvo Cassiopeia que no hace mención de la etapa de Diseño) mediante un refinamiento de la organización y de las interacciones identificadas en la fase de análisis (explícitamente como en DBI). Este refinamiento, en general, permite el perfeccionamiento de los tipos de agentes necesarios, permitiéndose en muchos casos la adición de nuevos tipos de agentes que no hubiesen sido aún detectados o la fusión de agentes muy relacionados en uno sólo. En MAS-CommonKADS, el diseño de red permite una visión uniforme del sistema, en GAIA una visión similar se da en el modelo de conocimiento.

- El modelado de los componentes internos de los diferentes agentes presenta diferentes visiones en los trabajos analizados. En algunos casos se presentan diversas alternativas según las características de un agente concreto. Para ello se presentan diversos patrones de arquitecturas, como en MaSE. Finalmente, algunas propuestas optan por el desarrollo de arquitecturas específicas como es el caso de Kinny con la arquitectura BDI.

- El diseño de las interacciones presenta también la opción de un diseño independiente de los modelos de interacción existentes, como el caso de MaSE donde a la hora de especificar las interacciones entre roles (o agentes), en el diseño se utilizan dos diagramas (autómatas de estados finitos) uno para el emisor y otro para el receptor. Mientras en otros casos se emplean en el diseño protocolos de interacción, especificados mediante UML. Esto, en cierto modo, facilita la comprensión de los modelos obtenidos al emplear una notación bastante extendida. Finalmente, en otros trabajos el detalle en el diseño de las interacciones es muy bajo, impidiendo una comprensión total de dicho proceso.

CUADRO 2:

ANÁLISIS SINÓPTICO DE LAS FASES DE ANÁLISIS Y DISEÑO CONCERNIENTES A LAS METODOLOGÍAS

METODOLOGIAS	*VISTAS O FASES EN ANÁLISIS*	*VISTAS O FASES EN DISEÑO*
MAS-CommonKADS	**Fase de Conceptuación** - Modelo de Organización - Modelo de Agente - Modelo de Tareas - Modelo de la Experiencia - Modelo de Comunicación y - Modelo de Coordinación	- Diseño de Red o Arquitectura - Diseño de Agentes o Aplicación - Diseño de Plataforma
GAIA	- Modelo de Roles - Modelo de Interacción	- Modelo de Agente - Modelo de Servicios - Modelo de trato o de Conocimiento
BDI	**Vista Externa:** - Modelo de Agente - Modelo de Interacción **Vista Interna:** - Modelos de Creencias - Modelos de Objetivos y - Modelos de Planes	Refinamiento de los modelos del Análisis
MaSE	- Capturar Metas u Objetivos - Aplicación casos de uso - Refinar los Roles - Creación clases de agente	- Construcción de conversaciones - Ensamblado clases de agente - Diseño del sistema
Cassiopeia	- Definición de los comportamientos elementales - Descripción estructural - Descripción de la dinámica organizacional	

CAPITULO V

CONCLUSIONES

En el desarrollo de los agentes, la necesidad de lograr que el proceso de creación de los sistemas multiagentes, mantuviera ciertos estándares y reglas que le permitieran al desarrollador tener un lineamiento que canalizara sus esfuerzos en pro de lograr su objetivo final (desarrollar un sistema funcional multiagente), surgieron propuestas de metodologías que en la actualidad han sido probadas y discutidas por numerosos investigadores.

Este trabajo por esta ubicado en el área de agentes racionales y esta enmarcado en la modalidad de proyecto de investigación monográfica documental, estuvo enfocado en el análisis comparativo de las metodologías: **BDI** (Belief Desire Intentiton), **GAIA**, **MaSE** (Multiagent Systems Engineering), MAS-Common KADS y Cassiopeia, metodologías basadas en sistemas multiagente. La importancia del tema radica en que no existe aun un estándar de método para desarrollar software orientado a agentes, esto lleva a una confusión dado que no se sabe a priori cuál de todas las alternativas disponibles es mejor. Además de esto, es importante conocer a fondo qué características posee una metodología determinada, debido a que su elección será clave para la construcción del software requerido, es por eso que es útil verificar las características de las metodologías.

De todas las metodologías existentes, las cinco (5) que se plasmaron en este trabajo, a pesar de ser un número muy pequeño comparado al total de metodologías que se pueden encontrar, permiten abarcar una gran gama de posibilidades de desarrollo, aplicables a cualquier sistema multiagente, partiendo de los sistemas prioritarios para la Fuerza Aérea de los Estados Unidos utilizando MaSE, hasta los ya populares partidos de fútbol robot (RoboCup) en los que se emplea Cassiopeia.

El objetivo que se planteó estaba referido a tomar un conjunto limitado de cinco metodologías existentes orientadas a agentes y desarrollar un análisis comparativo entre estas, el cual permita resaltar las características fundamentales de cada metodología, así como también las similitudes y diferencias existente entre las metodologías elegidas para el caso de estudio.

Se puede concluir que el objetivo principal ha sido alcanzado, cumpliéndose a cabalidad, obteniendo como resultado dos cuadros sinópticos, el primero referente a comparaciones generales de las metodologías y el segundo a las fases de análisis y diseño de cada una de ellas.

La interpretación que pueda surgir de este estudio, no es la de determinar cual es mejor que otra, sino por el contrario dar soporte bibliográfico a la toma de decisión sobre cual se adapta mejor al sistema que se desea desarrollar.

Cada metodología aporta un enfoque distinto de cómo desarrollar un sistema multiagente, pero en el fondo casi todas manejan los mismos términos con pequeñas diferencias, dependiendo del enfoque y moldeables a las necesidades del tipo de sistema para el cual fueron diseñadas y aceptadas como metodologías.

La mayoría de las metodologías que se seleccionaron comparten la idea de Rol o comportamiento del agente, considerado la importancia de éste junto a la relación y comunicación con otros agentes que conformen el sistema, así como con su entorno de desenvolvimiento.

El punto más notable del trabajo es que las metodologías expuestas intentan cubrir fundamentalmente las etapas de análisis y diseño de sistemas multiagente. El resto de etapas no son consideradas o bien se deja total libertad para su desarrollo. Salvo MaSE la propone pero no la desarrolla, de este grupo se excluye a Cassiopeia ya que no define una fase de Diseño, sino solo una fase de Análisis centrada en el modelado de la organización, lo que genera la pregunta ¿Cómo desarrollo las otras etapas del ciclo de vida de los sistemas, una vez finalizada las etapas que cubren las metodologías?.

Como se ha dicho en varias ocasiones, no son las únicas metodologías existentes, por lo que queda libertad de poder buscar otra que se pueda adaptar mejor al sistema propuesto para ser desarrollado.

CAPITULO VI

PROSPECCIÓN DE LA INVESTIGACIÓN

Como continuación de este trabajo se proponen a futuros, anexar a este documento una parte practica, donde se expongan casos de estudios muy particulares, que reflejen como se lleva a la realidad cada fase de las metodologías planteadas, lo que representa un valor agregado de mucha utilidad.

En segundo lugar, un progreso significativo, estaría dado por anexar otras metodologías y contrastar sus características más relevantes contra los resultados obtenidos en este trabajo, lo que aumentaría las alternativas presentadas a los desarrolladores para la selección de la metodología que mejor se adapte a sus requerimientos.

REFERENCIAS BIBLIOGRAFICAS

[Aben, 93] M. Aben. CommonKADS inferences. ESPRIT Project P5248 KADS-II/M2/TR/UvA/041/1.0, University of Amsterdam, June 1993.

[Álvarez, 98] J. Álvarez, CONCEPTO DE METODOLOGÍA. Departamento de Informática Universidad de Valladolid Campus de Segovia, 1998

[Amo et al, 95] Alonso Amo, F y Segovia Pérez, F. "Entornos y metodologías de programación". Paraninfo, Madrid 1995

[Amescua et al, 95] Amescua Seco, Antonio de y otros "Ingeniería del software de gestión. Análisis y diseño de aplicaciones" Paraninfo, Madrid 1995

[Arsène et al, 02] Arsène Sabas, Mourad Badri & Sylvain Delisle, Applying A New Multidimentional Framework To The Evaluation Of Multiagent System Methodologies, Universidad Laval y Universidad de Québec à Trois-Rivières, Canada 2002
http://www.damas.ift.ulaval.ca/~sabas/Documents/ISE2002.pdf

[Avouris, 92] N. M. Avouris. User interface design for DAI applications: an overview. In N. M. Avouris and L. Gasser, editors, Distributed Artificial Intelligence: Theory and Praxis, pages 141–162. Kluwer Academic Publishers: Boston, MA, 1992.

[Avouris et al, 92] N. M. Avouris and L. Gasser, editors. Distributed Artificial Intelligence: Theory and Praxis, volume 5 of Computer and Information Science. Kluwer Academic Publishers, Boston, MA, 1992.

[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

[Bellifemine 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.

[Bisquerra, 89] Rafael Bisquerra, METODOS DE LA INVESTIGACIÓN EDUCATIVA. Ediciones CEAC Peru, 1989

[Boehm, 93] B. W. Boehm. A spiral model of software development and enhancement. In R. Donald, editor, Software Management, pages 120–131. IEEE Computer Society Press, 1993. Reprinted from em Computer, Vol. 21, No. 5, May 1988, pp. 61-72.

[Bond et al, 88] A. H. Bond and L. Gasser. An analysis of problems and research in DAI., . Morgan Kaufmann Publishers: San Mateo, CA, 1988.

[Bonesi, 01] Pablo Ezequiel Bonesi, Mejoramiento de Metodologías de Análisis & Diseño de Sistemas Multiagente Ingeniería de Software, Universidad Argentina de la Empresa, 2001.

[Botti et al, 99] V. Botti, C. Carrascosa, V. Julian, J. Soler. The ARTIS Agent Architecture: Modelling Agents in Hard Real-Time Environments. Proceedings of the MAAMAW'99. Lecture Notes In Computer Science, vol. 1647. Springer - Verlag , Valencia 1999.

[Botti et al, 00] V. Botti, V. Julián, "Agentes Inteligentes: el siguiente paso en la Inteligencia Artificial", Dpto. sistemas Informáticos y Computación Universidad, Politécnica de Valencia 2000

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

[Burmeister, 96] B. Burmeister. 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, 1996.

[Camacho et al, 02] Camacho S., Teresa de Niño, Pire R., Rodriguez A. Manual para la Presentación del Trabajo Conducente al Grado Academico: Especialización, Maestría y Doctorado, UCLA 2002

[Collinot et al, 96] Collinot, A., Drogoul, A., and Benhamou, P., Agent Oriented Design of a Soccer Robot Team. Proceedings of the 2nd International Conference on Multi-Agent Systems, ICMAS, 1996: 41-47, Kyoto, Japan.
http://www-poleia.lip6.fr/~drogoul/papers/index.html

[Collinot et al, 98] Collinot, A., and Drogoul, A. 1998. Using the Cassiopeia method to design a Soccer Robot Team. Applied Artificial Intelligence (to appear).

[Cota, 94] Cota A. "Ingeniería de Software". Soluciones Avanzadas. Julio de 1994.

[Cugola et al, 96] G. Cugola, C. Ghezzi, G. P. Picco, and G. Vigna, "A Characterization of Mobility and State Distribution in Mobile Code Languajes," Proceedings of the 2nd Workshop on Mobile Objects Systems, (1996).

[Dam et al, 03] Khanh Hoa Dam y Michael Winikoff, Comparing AgentOriented Methodologies, School of Computer Science and Information Technology RMIT University, Melbourne, Australia 2003 http://goanna.cs.rmit.edu.au/~winikoff/Papers/aois2003.pdf

[De Giacomo et al, 00] De Giacomo, G., Lespérance, Y. y Levesque, H. J.: ConGolog, a concurrent programming language based on the situation calculus. Artificial Intelligence. Vol. 121 pp. 109-169.2000.

[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.

[DeMarco, 79] Tom DeMarco, "Structured Analysis and System Specification" Yourdon Press Computing Series, 1979.

[Drogoul et al, 98] Drogoul, A., and Collinot, A. 1998. Applying an Agent-Oriented Methodology to the Design of Artificial Organisations : a Case Study in Robotic Soccer. Autonomous Agents and Multi-Agent Systems, 1(1).

[Farías, 98] Mario Farías-Elinos**,** Diseños **de Sistemas reinformación, Universidad de la Salle, Mexico, 1998**

[Franklin et al, 96] Franklin, S., Graesser, A.: Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages. Springer-Verlag (1996).

[Garrido, 04] Samuel Garrido Daniel, "Programación Orientada a Componentes" 2004-02-13 , http://computacion.cs.cinvestav.mx/~sgarrido/cursos/ing_soft/Componentes/node28.html

[Giret et al, 00] Adriana Giret, Emilio Insfrán, Oscar Pastor, Luca Cernuzzi, OO-METHOD para el desarrollo de Sistemas de Agentes, Junio 2000.

[Gómez, 00] Jorge J. Gómez Sanz, Termostatos y Agentes. Actas de conferencia. Simposio Español de Informática Distribuida. 2000.

[Gómez, 03] Jorge J. Gómez Sanz, Metodologías para el desarrollo de sistemas multi-agente, Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial. No.18 (2003),

[Huhns, 98] Huhns, M., Singh, M. P.: Readings in Agents. Readings in Agents. Chapter 1, 1-24 (1998).

[Henao, 00] Henao, M. CommonKADS, una buena herramienta para la Gerencia del Conocimiento. Revista Universidad EAFIT, España. 9-13. (2000).

[Hoog et al, 94] R. de Hoog, W. Menezes, C. Toussaint, B. Benus, and D. Gottlieb. CommonKADS development method guidelines volume 2: Project management in CommonKADS. Deliverable DM10.a ESPRIT Project 5248 KADS-II/M10/DM10.a/UvA/068/2.0, University of Amsterdam, Cap Programmator and Cap Gemini Innovation. (1994).

[Huang et al, 95] J. Huang, N. R. Jennings, and J. Fox. Agent-based approach to health care management. Applied Artificial Ingelligence, 9(4):401–420. (1995).

[Iglesias et al, 02] Carlos A. Iglesias, Mercedes Garijo, José C. González, Metodologías Orientadas a Agentes.
URL: http://www.isys.dia.fi.upm.es/doctorado0203/iglesias.pdf

[Igresias et al, 96] C. A. Iglesias, J. C. González, and J. R. Velasco. MIX: A general purpose multiagent architecture. In M. Wooldridge, K. Fischer, P. Gmytrasiewicz, N. Jennings, J. M:uller, and M. Tambe, editors, INTELLIGENT AGENTS II: Agent Theories, Architectures, and Languages, volume 1037 of Lecture Notes in Artificial Intelligence, pages 251–266. Springer-Verlag. (1996).

[Iglesias, 98] Carlos Á. Iglesias F. Tesis Doctoral "Definición De Una Metodología Para El Desarrollo De Sistemas Multiagente", UNIVERSIDAD POLITÉCNICA DE MADRID, 1998.

[**ITU 99]** International Telecommunication Union: ITU 100:Formal Description Techniques (FDT)- Specification and Description Language (SDL). Informe 1999.

[Jacobson et al, 92] I. Jacobson, M. Christerson, P. Jonsson, and Övergaard. Object-Oriented Software Engineering. A Use Case Driven Approach. ACM Press, 1992.

[Jacobson, 98] Jacobson, I. 1998. "Applying UML in The Unified Process" Presentación. Rational Software. Presentación disponible en http://www.rational.com/uml como UMLconf.zip

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

[Jennings, 99] Dr. Nicholas Jennings, discurso al recoger el premio al mejor investigador novel del congreso internacional de Inteligencia Artificial, Estocolmo ,1999.

[Jennings et al, 98] N. R. Jennings, K. Sycara, M. Wooldridge. "A Roadmap of Agent Research and Development," Autonomous Agents and Multi-Agent Systems, I, (1998).

[Julián et al, 03] Vicente J. Julián, Vicente J. Botti, Estudio de métodos de desarrollo de sistemas multiagente , Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial. No.18 (2003),

[Kendall et al, 96] E. A. Kendall, M. T. Malkoun, and C. Jiang. A methodology for developing agent based systems for enterprise integration. In D. Luckose and Z. C., editors, Proceedings of the First Australian Workshop on DAI, Springer-Verlag: Heidelberg, Germany, 1996. URL: http://www.cse.rmit.edu.au/rdsek/papers/AgentBasedMeth.ps.

[Kinny et al, 97] David Kinny, Michael Georgeff y Anand Rao. A methodology and modelling techniques for systems of BDI Agents, Technical Note 58, Australian Artificial Intelligence Institute, 1996.

[Kinny et al, 96] David Kinny y Michael Georgeff. Modelling and Design of Multi-Agent System, Australian Artificial Intelligence Institute, 1996.

[Larousse, 98] Diccionario Larousse en Español, 1998.

[Lespérance et al, 96] Lespérance, Y., Levesque, H. J., Lin, F. y Marcu, D.: Foundations of a Logical Approach to Agent Programmaing. Actas de conferencia. Springer-Verlag. Lecture Notes in Artificial Intelligence. 1996.

[Lewis, 94] Lewis G. "What is Software Engineering?" DataPro (4015). Feb 1994.

[Morales, 95] Adrian Morales Gomez, ANÁLISIS ESTRUCTURADO, Universidad la SAlle de Mexico Ingenieris Ciernetica y en sistemas computacionales, 1995

[Nwana, 96] Nwana, H. S.: Software Agents: An Overview. Intelligent Systems Research. AA&T, BT Laboratories, Ipswich, United Kingdom 1996.

[OMG, 98] MASIF-RTF Results. Object Management Group, 1998.

[Parunak, 99] Parunak, H. Van Dyke and Odell, James: Engineering Artifacts for Multi-Agent Systems, ERIM CEC. 1999.

[Pérez, 96] Alfonso Pérez, DE LA INTELIGENCIA ARTIFICIAL A LA INGENIERÍA DE CONOCIMIENTOS IEEE Senior Member, Departamento de Ingeniería de Sistemas. UNIVERSIDAD NACIONAL - COLCIENCIAS; 1996

[Pressman 93] Roger S. Pressman. "Software Engineering, a proctitioner's approach", Mc Graw Hill, tercera edicion, 1993.

[Pressman, 97] R. S. Pressman. Software engineering. In M. Dorfman and R. H. Thayer, editors, Software engineering, pages 57–74. IEEE Computer Society, 1997.

[Potts et al, 94] C. Potts, K. Takahashi, and A. Anton. Inquiry-based scenario analysis of system requirements. Technical Report GIT-CC-94/14, College of Computing, Georgia Institute of Technology, Atlanta, GA, USA, Atlanta, GA, USA. (1994).

[Ramos, 95] F. Ramos, G. Sanchez, E. Espinoza, M.J. Elliot. "A Fuzzy Temporal Reasoning Mechanism for Planning in Multi-Agent Domains", Third IASTED International Conference in Robotics and Manufacturing, June 1995, Cancún, México.

[Real Academia Española, 92] Diccionario de la Lengua Española. Real Academia Española. XXI Edición, (1992).

[Romero, 98] Paola Romero Guillén, "Análisis y Diseño Orientado a Objetos" Capitulo II Métodos y Modelos, Instituto Tecnológico de la Laguna .

[Rumbaugh et al, 91] J. Rumbaugh, M.Blaha, W. Premerlani, and V. F. Eddy. Object-Oriented Modeling and Design. Prentice-Hall, 1991.

[Russell, 96] Russell, S: Inteligencia Artificial: un enfoque moderno. Prentice - Hall. México, (1996).

[Santaolaya et al, 98] René Santaolaya Salgado, Olivia G. Fragoso Diaz, Máximo López Sánchez, "MODELO DE DESARROLLO DE SOFTWARE BASADO EN COMPONENTES", Centro Nacional de Investigación y Desarrollo Tecnológico (cenidet), Mexico

[Scott et al, 02] Scott A. DeLoach, Eric T. Matson, Yonghua Li ; Applying Agent Oriented Software Engineering to Cooperative Robotics, Department of Computing and Information Sciences, Kansas State University, Proceedings of the The 15th International FLAIRS Conference (FLAIRS 2002). Pensacola, Florida. May 16-18, 2002.

[Shoham, 93] Y. Shoham. Agent-oriented programming. Artificial Intelligence, 1993.

[Thompson et al, 99] Craig Thompson C. and Odell J., Agent Technology Glossary "OMG Agent WG Technology Green Paper", Object Management Group URL: http://www.objs.com/agility/tech-reports/9909-agent-glossary.html .

[Wærn et al, 93] A. Wærn, K. Höök, and R. Gustavsson. The common kads communication model. Technical report ESPRIT Project 5248 KADS-II/M3/TR/SICS/006/V.2.0, Swedish Institute of Computer Science and ERITEL. (1993).

[Webster, 96] Webster's Encyclopedic Unabridged Dictionary of the English Language. Ed. Gramercy (1996).

[Wirfs-Brock et al, 90] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. Designing Object-Oriented Software. Prentice-Hall, 1990.

[Wood et al, 01] Mark F. Wood Scott A. DeLoach An Overview of the Multiagent Systems Engineering Methodology Department of Electrical and Computer Engineering Air Force Institute of Technology, January 2001.

[Wooldridge et al, 95] Wooldridge, M. and Jennings, N. R.: Intelligent agents: Theory and practice. The Knowledge Engineering Review, 10(2):115–152, (1995).

[Wooldridge et al, 98] Wooldridge M. y Jennings N.R.: Pitfalls of Agent-Oriented Development. Actas de conferencia. Proceedings of the Second International Conference on Autonomous Agents. 385-391.1998.

[Wooldridge et al, 00] Wooldridge, M., Jennings, N. R., Kinny, D. The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems (2000)

[Yourdon, 91] Edward Yourdon, Object-Oriented Analysis, 1991

[Zambonelli et al, 01] Zambonelli, F., Jennings, N. R., Omicini, A., Wooldridge, M. . Agent-Oriented Software Engineering for Internet Applications, (2001). Coordination of Internet Agents, Springer Verlag.

[Zorrilla et al, 99] Santiago Zorrilla, Miguel Torres, Amado Luiz Cervo, Pedro Alcino Bervian. "METODOLOGÍA DE LA INVESTIGACIÓN". Mc.Graw Hill, 1999.

[Zavala, 00] Zavala R. Diseño de un Sistema de Información Geográfica sobre internet. Tesis de Maestría en Ciencias de la Computación. Universidad Autónoma Metropolitana-Azcapotzalco. México, D.F. En prensa. 2000.