



UNIVERSIDADE
DE VIGO

ESCOLA SUPERIOR DE ENXEÑERÍA INFORMÁTICA

Memoria do Proxecto Fin de Carreira que presenta

D. Cristina Gómez Alonso

para a obtención do Título de **Enxeñeiro en Informática**

**"Análisis de Metodologías para el desarrollo de Sistemas Multi-Agente
y caso de estudio sobre la plataforma HeCaSe2"**



Outubro, 2007

Proxecto Fin de Carreira N°: ENI-195

Director/a: Eva Lorenzo Iglesias, **Codirector:** David Isern Alarcón

Área de coñecemento: Linguaxes e Sistemas Informáticos

Departamento: Informática

*Me gustaría dedicar este proyecto,
muy especialmente, a David Isern, por su
incondicional apoyo y el aporte continuo de
conocimientos para la realización y mejora
del presente documento.*

*Agradecer, también, su ayuda a mis padres,
mi hermana, Eva Lorenzo, Toni Moreno,
José Doval y Ángel.*

TABLA DE CONTENIDOS

1	INTRODUCCIÓN	1
1.1	IDENTIFICACIÓN DEL PROYECTO.....	1
1.2	ESTRUCTURA DE LA DOCUMENTACIÓN.....	1
1.3	ORIGEN DEL PROYECTO.....	1
1.4	OBJETIVOS BÁSICOS DEL PROYECTO	2
2	METODOLOGÍAS PARA EL DESARROLLO DE SMA: ESTADO DEL ARTE	3
2.1	CLASIFICACIÓN DE METODOLOGÍAS	3
2.2	COMPARATIVA DE METODOLOGÍAS	4
3	METODOLOGÍA INGENIAS	7
3.1	ORÍGENES Y EVOLUCIÓN	7
3.2	META-MODELOS	8
3.2.1	<i>Agente</i>	9
3.2.2	<i>Tareas y Objetivos</i>	9
3.2.3	<i>Interacciones</i>	10
3.2.4	<i>Entorno</i>	10
3.2.5	<i>Organización</i>	11
3.3	CICLO DE VIDA.....	12
3.4	IDK (INGENIAS DEVELOPMENT TOOLKIT)	13
4	ANÁLISIS Y DISEÑO DE UN SMA EN INGENIAS.....	15
4.1	PLATAFORMA HECase2.....	15
4.1.1	<i>Servicios Implementados</i>	15
4.1.2	<i>Arquitectura</i>	17
4.2	APLICACIÓN DE INGENIAS SOBRE HECase2.....	18
4.2.1	<i>Fase de INICIO del RUP</i>	19
4.2.1.1	<i>Análisis</i>	19
4.2.1.2	<i>Diseño</i>	22
4.2.2	<i>Fase de ELABORACIÓN del RUP</i>	23
4.2.2.1	<i>Análisis</i>	23
4.2.2.2	<i>Diseño</i>	34
4.2.3	<i>Fase de CONSTRUCCIÓN del RUP</i>	36
4.2.4	<i>Implementación</i>	37
5	PLANIFICACIÓN Y REQUERIMIENTOS DEL PROYECTO.....	38
5.1	PLANIFICACIÓN	38
5.1.1	<i>Estimación temporal previa</i>	38
5.1.2	<i>Estimación temporal real</i>	39
5.1.3	<i>Conclusiones de la estimación temporal</i>	41
5.2	HERRAMIENTAS EMPLEADAS	42
5.3	PRESUPUESTO	44
6	CONCLUSIONES Y TRABAJO FUTURO.....	45
6.1	CONCLUSIONES	45
6.2	DIFICULTADES ENCONTRADAS	45
6.3	TRABAJO FUTURO.....	46
7	ACRÓNIMOS.....	47
8	REFERENCIAS	48
9	ANEXO 1: NOMENCLATURA Y NOTACIÓN INGENIAS.....	49
9.1	NOMENCLATURA.....	49
9.2	NOTACIÓN	49
10	ANEXO 2: RESEARCH REPORT DEIM-RR-07-003	52

1 INTRODUCCIÓN

1.1 IDENTIFICACIÓN DEL PROYECTO

Título:	“Análisis de Metodologías para el desarrollo de Sistemas Multi-Agente y caso de estudio sobre la plataforma HeCaSe2”
Código:	ENI-195
Alumna:	Cristina Gómez Alonso DNI: 36176605-C
Directora:	Eva M ^a Lorenzo Iglesias Área de Lenguajes y Sistemas. Departamento de Informática Universidad de Vigo
Codirector:	David Isern Alarcón Área de Ciencias de la Computación e Inteligencia Artificial Universidad Rovira i Virgili
Fecha:	Octubre 2007

1.2 ESTRUCTURA DE LA DOCUMENTACIÓN

La documentación del proyecto se presenta dividida en dos bloques:

Análisis de Metodologías para SMA (Sistemas Multi-Agente): describe el estado del arte de las metodologías para la Ingeniería del Software Orientada a Agentes. El trabajo realizado a lo largo de este proyecto incluye un análisis específico de las metodologías más actuales. Su finalidad es mostrar una comparativa que facilite la selección de la metodología más adecuada para un problema específico que se desea resolver mediante Sistemas Multi-Agente.

Aplicación de la Metodología INGENIAS a la Herramienta HeCaSe2: detalla las fases de análisis y diseño de la estructura de la Herramienta HeCaSe2 (Health Care Service (versión 2)) empleando la Metodología INGENIAS.

1.3 ORIGEN DEL PROYECTO

El desarrollo de sistemas complejos distribuidos basados en tecnologías multi-agente (SMA) requiere el uso de alguna metodología durante las fases de análisis y diseño. La principal ventaja de su utilización es que el conocimiento adquirido al desarrollar aplicaciones se podrá *reutilizar o adaptar a nuevos proyectos*.

Actualmente, la propuesta de metodologías orientadas al desarrollo de SMA es muy amplia y compleja, ya que son actualizadas con frecuencia por sus desarrolladores y el

surgimiento de nuevas propuestas es común entre los grupos de investigación en la Inteligencia Artificial a nivel mundial.

El grupo de investigación “**iTAKA**” (*“Intelligent Technologies for Advanced Knowledge Acquisition”*) del Departamento de Informática de la Universidad Rovira i Virgili de Tarragona, donde la alumna se encuentra colaborando actualmente, está especialmente interesado en que se realice un estudio sobre la situación actual de estas metodologías, estudio del que se puedan extraer conclusiones acerca de las ventajas e inconvenientes de cada una de ellas, y poder establecer una serie de parámetros que permitan a desarrolladores de sistemas distribuidos, escoger la mejor aproximación dadas las características del proyecto a implementar.

Además, resulta de gran interés para el grupo realizar una aplicación práctica de los conocimientos adquiridos en el estudio anterior. Para ello, se propone también la aplicación de una metodología concreta sobre una plataforma de SMA, HeCaSe2 (Health Care Services, versión 2) desarrollada por uno de los miembros de iTAKA, David Isern Alarcón, en el marco de su tesis doctoral.

1.4 OBJETIVOS BÁSICOS DEL PROYECTO

Como se ha mencionado anteriormente, el principal objetivo de esta proyecto es analizar el estado del arte de las metodologías orientadas al desarrollo de SMA. De esta forma se mejora el desarrollo de aplicaciones (cualitativamente y reduciendo del tiempo de rediseño posterior) gracias a la toma de decisiones en una fase anterior a la propia implementación.

Este objetivo conlleva una serie de necesidades que se deben de cubrir eficazmente:

- Qué son los agentes y qué características debe tener una metodología para diseñarlos.
- Conocimiento de los objetivos generales de las metodologías para el desarrollo de Sistemas Multi-Agente.
- Estudio comparativo de las metodologías existentes; clasificación y evaluación de las diferentes herramientas.
- Profundización en las metodologías más destacables (análisis de las aplicaciones para las que son particularmente idóneas las diferentes herramientas).

Como resultado de este estudio, el proyecto muestra la aplicación de una metodología *óptima* a un caso de estudio, valorando su usabilidad y analizando la herramienta de soporte que se oferta. De la misma forma, se muestra que aún hay una distancia entre la formalización de un SMA y su implementación concreta.

2 METODOLOGÍAS PARA EL DESARROLLO DE SMA: ESTADO DEL ARTE

Este apartado del proyecto ha sido publicado como report técnico con el título de “*Software Engineering Methodologies to Develop Multi-Agent Systems: State-of-the-Art*” en la Universidad Rovira i Virgili en el mes de Agosto del presente año.

Este estudio se ha incluido como anexo al final del documento, y también se encuentra disponible en la dirección web:

http://deim.urv.cat/~itaka/CMS/images/pdf/report_aose.pdf

2.1 CLASIFICACIÓN DE METODOLOGÍAS

La jerarquía que se presenta en la siguiente figura es la que se ha considerado que clasifica las metodologías actuales. Se han añadido las metodologías en las que se ha profundizado por considerarlas más relevantes:

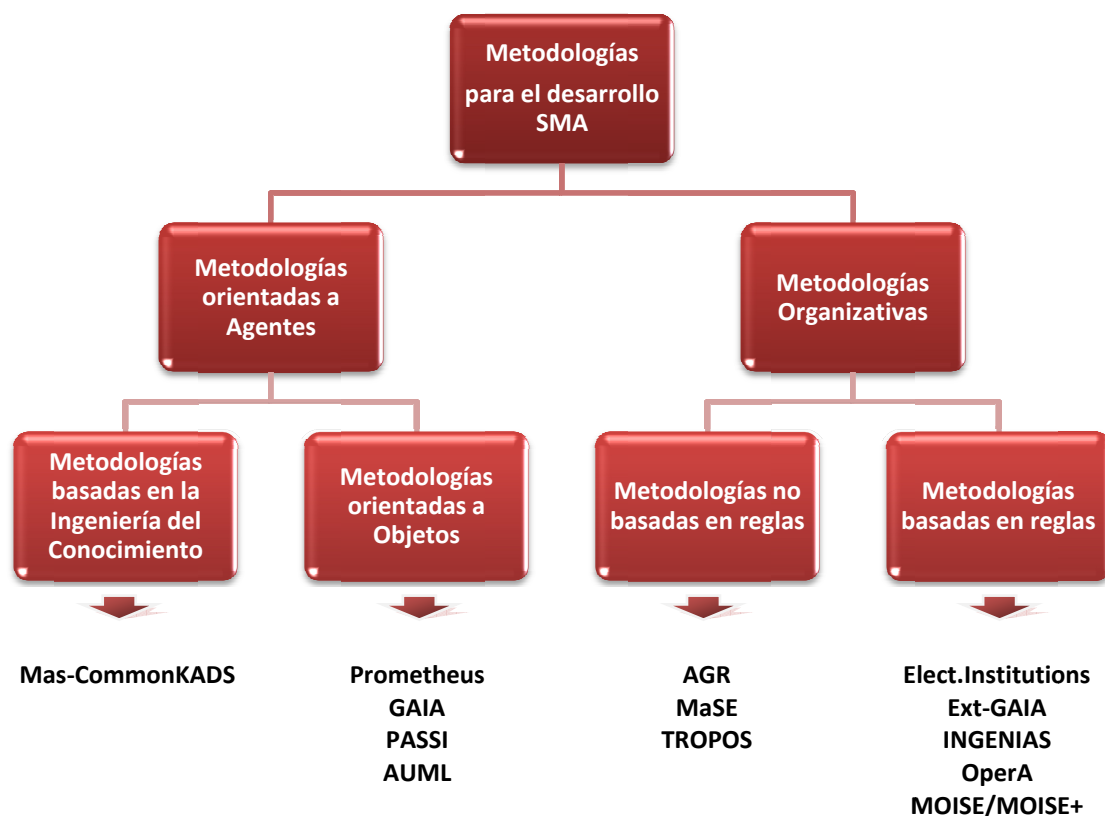


Fig. 1. Clasificación de metodologías para SMA

Los motivos que han llevado a bifurcar la clasificación en dos tendencias ha sido la nueva orientación para las metodologías que se propone más centrada en la organización de los agentes, abandonando el enfoque anterior que consideraba los agentes como entes individuales que pertenecen a un grupo pero que colaboran para lograr sus propios objetivos.

Dentro de la rama clásica de **“Metodologías orientadas a Agentes”**, se distinguen las **“Metodologías orientadas a Objetos”**, que tomando como referencia las metodologías OO las extienden para cubrir las necesidades de detalle de los SMA, es decir, se establece un paralelismo entre agentes y objetos, frente a las **“Metodologías basadas en la Ingeniería del Conocimiento”** que basan su aplicaciones de SMA en la descripción del proceso de adquisición de conocimiento por parte de los agentes software que las integran.

Por otro lado, las **“Metodologías Organizativas”** se distinguen entre aquellas que tienen en cuenta el concepto de regla o norma social en su especificación, que se han denominado **“Metodologías basadas en reglas”**, o las que no lo valoran, **“Metodologías no basadas en reglas”** (Más información: pág. 22 del report “Classification of Organizational Methodologies”).

Como se refleja en la parte inferior de la Figura 1, se han analizado en el report un total de **13 metodologías**, las cuales se han clasificado: cinco en la rama centrada en los agentes (Prometheus, GAIA, PASSI, AUML y MAS-CommonKADS) y ocho en la rama organizativa (AGR, MaSE, TROPOS, Electronic Institutions, Extended-GAIA, INGENIAS, OperA y MOISE/MOISE+). Además de las presentadas, se añaden descripciones breves de otras metodologías que no se consideraron tan sobresalientes, pero de las que si se indican referencias a otros artículos por si fuesen de interés para el lector (como por ejemplo: Civil Agent Societies, HarmonIA, SODA...).

2.2 COMPARATIVA DE METODOLOGÍAS

Además de la clasificación y descripción breve de las metodologías más relevantes, se incluye una comparativa (o *framework*) que analiza las principales características distintivas.

Los criterios que se han valorado se muestran agrupados en cinco clases:

- **Conceptos y propiedades** (Concepts and properties): evalúan los principales conceptos y propiedades que una metodología orientada a agentes debería de implementar.
 - *Autonomía (Autonomy)*: expresa la habilidad de un agente de resolver un problema de forma autónoma.
 - *Comunicación (Communication)*: describe el modelo de comunicación usado, como por ejemplo, basado en mensajes o memoria.
 - *Cooperación (Cooperation)*: explica como una meta común es alcanza por los agentes.
 - *Adaptabilidad (Adaptability)*: muestra cambios en los agentes según el entorno u otros agentes.
 - *Pro-actividad (Pro-activity)*: si la metodología permite al diseñador representar esta característica.
- **Lenguaje de modelado** (Modelling language): tratan sobre los niveles de formalización y expresividad de la notación propuesta.
 - *Formalización/Precisión (Formalisation)*: indica la claridad de los modelos definidos y si la herramienta dispone de algún lenguaje de representación o formalización.
 - *Expresividad (Expresiveness)*: permite expresar los datos y el flujo de datos dentro del sistema.
 - *Abstracción (Abstraction)*: crea diferentes niveles de detalle de los modelos.

- **Modelos** (Model-related): evalúan las capacidades de los modelos presentados por las metodologías.
 - *Cobertura del ciclo de vida (Coverage)*: conjunto de fases que son cubiertas por el ciclo de vida de la metodología.
 - *Complejidad (Complexity)*: mide el nivel de esfuerzo para aprender y usar la metodología.
 - *Continuidad temporal (Temporal)*: expresa los cambios de los agentes a lo largo del tiempo
 - *Interacción humano-aplicación (Human-computer)*: SMA quizás requieren el intercambio de información con usuarios (humanos) de entrada y/o salida. Esta interacción debería ser diseñada y representada apropiadamente.

- **Organización** (Organizational): evalúan las relaciones sociales entre las comunidades de agentes.
 - *Sistemas abiertos (Open systems)*: permite representar la incorporación / supresión de nuevos agentes/recursos dinámicamente.
 - *Topología (Topology)*: las relaciones entre los agentes deberían de poder ser expresadas con diferentes paradigmas. Metodologías podrían limitarse exclusivamente a uno o ser independientes. (Más información: pág. 19-21 del Anexo 2, “Paradigmas Organizacionales”)
 - *Normas sociales (Social norms)*: especifica a gran nivel los patrones de comunicación entre agentes o grupos de agentes.

- **Soporte** (Supportive feature): proporcionan ciertas consideraciones sobre las herramientas de soporte.
 - *Herramientas software (Software)*: expresa si existe alguna herramienta CASE diseñada para la metodología (por ejemplo, librerías de agentes, componentes, arquitecturas o soporte técnico).
 - *Disponibilidad de ejemplos (Examples)*: ayuda útil durante el aprendizaje o implementación de cualquier metodología.
 - *Empleo en proyectos (Projects)*: expresa la madurez de una metodología según su uso en proyectos.

Cada metodología se valora en todos estos criterios con un rango de 6 puntos: “++” (muy alto o completamente de acuerdo), “+” (alto o de acuerdo), “~” (medio o no especificado explícitamente por los autores), “-” (bajo o en desacuerdo), “--” (muy bajo o completamente en desacuerdo), “n.a.” (not available (no disponible)). A excepción de “Cobertura del ciclo de vida” que se indica mediante “A/D/I” (Análisis/Diseño/Implementación) según las fases del ciclo de vida que se cubran.

La tabla comparativa resultante se presenta en la figura siguiente (también disponible en el report en la pág.40):

	MAS-CommonKADS	PROMETHEUS	GAIA	PASSI	AUML	AGR	MASE	TROPOS	E-INSTITUTIONS	EXTENDED GAIA	INGENIAS	OPERA	MOISE	MOISE+
<i>Concepts and properties</i>														
Autonomy	+	++	+	++	-	+	+	+	+	+	++	+	++	++
Communication	+	++	++	++	++	+	+	+	+	++	++	+	++	++
Cooperation	+	+	+	+	-	+	+	≈	+	+	++	-	+	+
Adaptability	+	n.a.	-	-	-	+	-	≈	+	-	+	-	≈	≈
Pro-activity	+	++	++	-	--	+	--	-	+	++	++	-	≈	≈
<i>Modelling language</i>														
Formalisation	-	++	++	≈	+	-	-	+	+	++	++	≈	+	+
Expressiveness	-	≈	+	≈	+	-	-	+	+	≈	+	≈	+	+
Abstraction	-	+	+	+	+	-	-	-	+	+	+	+	+	+
<i>Model-related</i>														
Coverage	A/D	A/D/I	A/D	A/D/I	A/D	A/D	A/D/I	A/D/I	D	A/D	A/D/I	A/D	A/D	A/D
Complexity	≈	++	-	n.a.	-	-	-	--	≈	n.a.	++	≈	+	+
Temporal	n.a.	≈	-	-	++	--	-	-	++	≈	+	-	+	+
Human-computer	++	-	-	--	--	--	--	--	--	--	-	--	--	--
<i>Organizational</i>														
Open systems	+	-	--	+	-	-	+	+	++	+	++	-	+	+
Topology	+	-	+	--	-	-	-	+	-	-	++	+	+	+
Social norms	≈	-	-	-	≈	-	-	--	+	++	≈	+	+	++
<i>Supportive feature</i>														
Software	-	++	--	++	--	--	++	++	+	--	++	--	-	-
Examples	-	+	+	+	-	-	+	++	+	+	+	--	-	-
Projects	-	++	+	+	-	≈	+	++	+	+	++	-	-	-

Fig. 2. Comparativa de Metodologías para el desarrollo de SMA

Gracias a esta tabla, se puede analizar de forma global el estado del arte de las metodologías para SMA. Las conclusiones que se pueden extraer son:

- La mayoría de las metodologías cubren con éxito las propiedades y conceptos de los Sistemas Multi-agente.
- La formalización y precisión del lenguaje de modelado resulta en ciertas metodologías ambiguo, como, por ejemplo, en MAS-CommonKADS, AGR o MaSE; aunque en otras, como Prometheus, GAIA, Extended-GAIA e INGENIAS, se presenta muy bien detallado evitando las confusiones a los analistas.
- La mayoría de las metodologías permiten la abstracción a bajo nivel de sus diagramas.
- En ciclo de vida del proceso de desarrollo solamente cinco de las trece metodologías ofrecen directrices para la fase de implementación (Prometheus, PASSI, MaSE, Tropos e INGENIAS) y pocos permiten representar la evolución del sistema en el tiempo mediante sus modelos.
- Exclusivamente la metodología MAS-CommonKADS permite detallar la interacción humano-aplicación.
- En el aspecto organizacional, como ya se detallo en la jerarquía presentada en el apartado anterior, cinco metodologías representan una orientación centrada en el agente y ocho una orientación centrada en la organización global del sistema. A nivel de criterios se pueden observar como las normas sociales son analizadas (explícita o implícitamente) por las metodologías que cubre las 6 últimas columnas ("Metodologías basadas en reglas"). Comentar que la única diferencia en este análisis entre las metodologías Moise y Moise+, es su análisis deóntico, ya que Moise+ obta por profundizar en el detalle de los permisos y obligaciones de los roles (Más información: pág. 35-37 del report).
- Uno de los aspectos críticos de las metodologías es su uso en proyectos de gran envergadura o la existencia de ejemplos que ayuden a inexpertos desarrolladores a aprender el proceso de desarrollo y la correcta realización de sus diagramas.

3 METODOLOGÍA INGENIAS

Conforme al estudio del estado del arte de las metodologías para SMA realizado en la primera parte de este proyecto, en la segunda mitad se considera oportuno profundizar en una metodología concreta para exponer su proceso de aplicación. La metodología seleccionada ha sido **INGENIAS** por poseer las siguientes cualidades:

- Facilita al desarrollador las etapas de análisis y diseño ya que describe un detallado proceso de desarrollo.
- Especifica la organización de los agentes a diferentes niveles e indicando sus objetivos y tareas.
- Detalla el entorno en el que se encuentran los agentes.
- Describe los objetivos y tareas de los agentes de forma individual y colectiva.
- Muestra detalladamente el intercambio de mensajes entre agentes (interacciones).
- Especifica el estado mental del agente en base a hechos, creencias, eventos y objetivos; sus modificaciones y los roles que interpreta.
- Incluye diversos ejemplos que sirven de guía a desarrolladores.
- Proporciona una herramienta CASE (IDK (*INGENIAS Development Kit*)) con editor gráfico y generador de código, el cual incluye un módulo para JADE (*Java Agent Development*), uno de los frameworks de desarrollo de agentes más comunes, que emplea Java y está basado en el estándar FIPA¹ (*Foundation for Intelligent Physical Agents*)).

En el resultado comparativo de las metodologías realizado previamente se observa que INGENIAS ha sido la mejor cualificada según los criterios propuestos. Para nuestro caso concreto de estudio, HeCaSe2, INGENIAS cubre eficazmente todas las necesidades del SMA. No se podría afirmar que INGENIAS fuese la metodología óptima para cualquier sistema multi-agente ya que depende del sistema concreto y de la finalidad del desarrollador. INGENIAS no sería recomendable si se desease:

- Presentar un esquema del sistema poco detallado, ya que INGENIAS implica mucha profundización.
- Mostrar directamente la interacción usuario(humano) con el sistema porque es un aspecto que no se cubre.
- Mostrar aspectos dinámicos de la organización (como por ejemplo, la formación de grupos dinámicamente), ya que no se consideran.

En el ámbito de los sistemas abiertos, INGENIAS podría realizar la especificación de la aplicación, pero la integración de agentes externos y su interoperabilidad no podría ser cubierta por el generador de código ya que las comunicaciones por JADE se realizan mediante el envío de mensajes de contenido binario.

3.1 ORÍGENES Y EVOLUCIÓN

INGENIAS(Gómez-Sanz, 2002), surge en el año 2002 en el grupo de investigación “GRASIA!” (“*Grupo de Agentes Software: Ingeniería y Aplicaciones*”) de la Universidad Complutense de Madrid como mejora de las características de la metodología Message.

¹ <http://www.fipa.org>

A la anterior le aporta mayor nivel de detalle, cohesión e integración entre los meta-modelos, e incorpora la representación del entorno del sistema.

Hasta la fecha, en INGENIAS se han ido refinando conceptos de nomenclatura y detalle de los diagramas, así como la inclusión de nuevos módulos que amplían las opciones de la herramienta IDK. Una nueva versión (INGENIAS2) está prevista para el año 2008 como resultado de la colaboración de las universidades Complutense de Madrid, Murcia y Vigo.

3.2 META-MODELOS

INGENIAS propone para el modelado de un SMA **cinco meta-modelos** que se van refinando a lo largo del proceso de desarrollo². Estos meta-modelos resultan dependientes entre sí y se deben de realizar en el orden propuesto en el apartado posterior “3.3 Ciclo de Vida”, consiguiendo una definición y mejora de la especificación del SMA de forma iterativa.

Debido a las dependencias existentes, en el trabajo original de INGENIAS (Gómez-Sanz, 2002) se aconseja aplicar manualmente unos test de validación para cada modelo que aseguren su consistencia con respecto a otros modelos. Estas pruebas se han formulado como requisitos que indican para cada elemento si debe ser o no usado en otro meta-modelo.

A continuación se describen brevemente las características de cada uno de los meta-modelos (Henderson-Sellers B. and Giorgini P., 2005)(Mas, 2005).



Fig. 3. Meta-modelos de un SMA en INGENIAS

² Se ha añadido al final del documento un anexo explicativo de la nomenclatura y notación empleada en los meta-modelos.

3.2.1 AGENTE

Describe la funcionalidad de cada agente según sus responsabilidades (tareas que debe ejecutar), finalidad (objetivos que debe de perseguir) y capacidades (roles que interpreta).

El comportamiento de un agente viene definido por: su **estado mental** (información que permite a los agentes la toma de decisiones: hechos, creencias, eventos y objetivos), **gestor del estado mental** (proporciona las operaciones sobre los elementos de su estado mental y sus relaciones) y **procesador del estado mental** (decide la tarea a ejecutar, es decir, determina la evolución del estado mental).

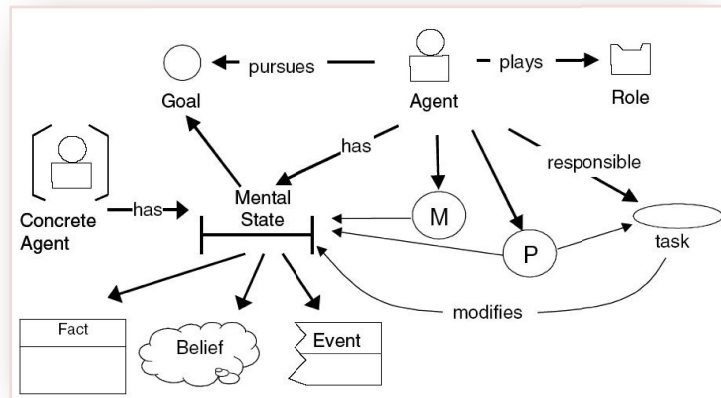


Fig. 4. Elementos del Meta-Modelo del Agente

3.2.2 TAREAS Y OBJETIVOS

Considera la descomposición en tareas según los objetivos, describiendo a su vez las entradas (precondiciones), salidas (postcondiciones), recursos y módulos software necesarios para su satisfacción. Este meta-modelo está basado en el “principio de racionalidad” ya que su propósito es justificar la ejecución de tareas en base a objetivos.

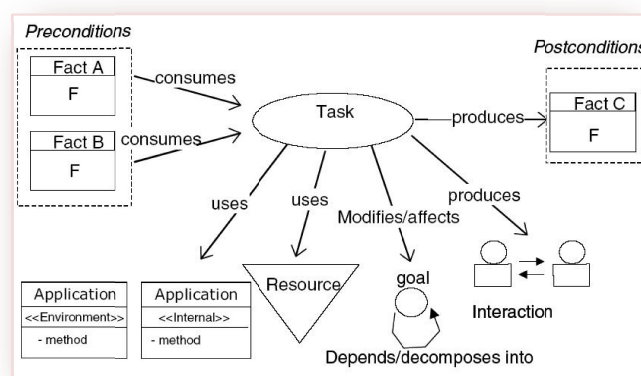


Fig. 5. Elementos del Meta-Modelo de Tareas y Objetivos

3.2.3 INTERACCIONES

Maneja el intercambio de información o consultas entre agentes(o agentes y humanos). Requiere la identificación de los elementos siguientes: **actores** (iniciador y colaboradores), **especificación** (detalle de la construcción de la interacción en tiempo de ejecución), **contexto** (objetivos y estados mentales de los participantes) y **naturaleza de la interacción** (relación entre los participantes).

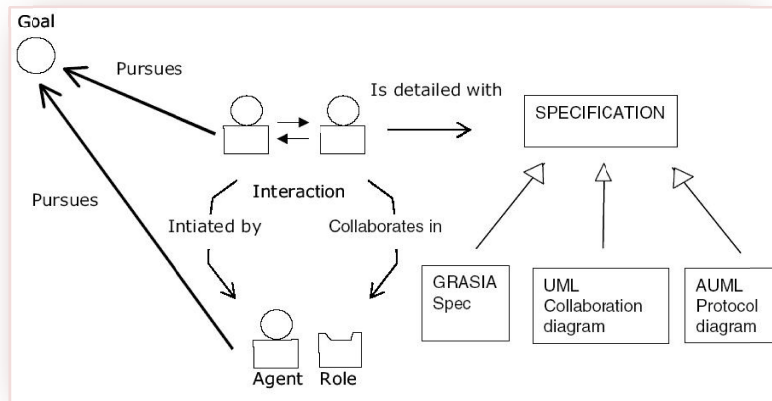


Fig. 6. Elementos del Meta-Modelo de Interacciones

La especificación de la interacción puede realizarse mediante tres tipos de diagramas: **GRASIA!**, **Colaboración UML** y **Protocolo AUMI**. El problema de los dos últimos es que no están orientados propiamente al modelado de interacciones entre agentes. Así, la justificación de por qué se está ejecutando la interacción, detalles acerca de por qué se están aceptando ciertos mensajes y por qué transcurre la interacción de un modo concreto, no son fácilmente expresables. Por ello, los desarrolladores han generado una *especificación GRASIA* adaptada a su uso dentro de la metodología. Esta última variante de especificación da cabida a la representación del estado mental del iniciador y de un colaborador en una unidad de interacción.

3.2.4 ENTORNO

Define qué existe alrededor del nuevo sistema y cómo lo percibe cada agente (efectos sobre sus acciones y control). El entorno está integrado por: **recursos** (consumibles o no consumibles), **otros agentes** o **aplicaciones** (internas o del entorno según sea software propio o no). Este meta-modelo no pretende modelar estos componentes, sino simplemente indicar la existencia de asociaciones entre elementos (agentes/grupos/organizaciones) del sistema en desarrollo con entidades ajenas.

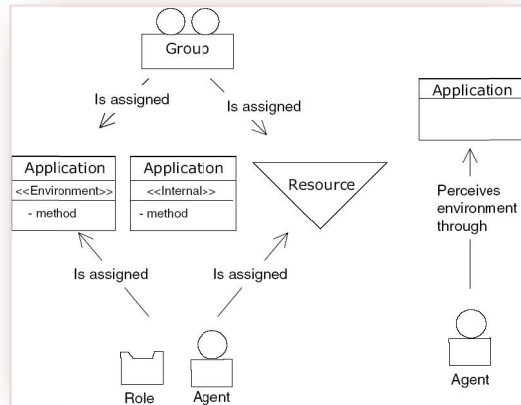


Fig. 7. Elementos del Meta-Modelo de Entorno

3.2.5 ORGANIZACIÓN

Describe el marco donde coexisten agentes, recursos, tareas y objetivos. Para ello es necesario definir la estructura, funcionalidad y relaciones sociales del sistema. Propone una descomposición del sistema en **grupos** (conjunto de agentes, roles, recursos y aplicaciones que comparten características comunes) y **flujos de trabajo** (asociaciones de tareas e información general sobre su ejecución).

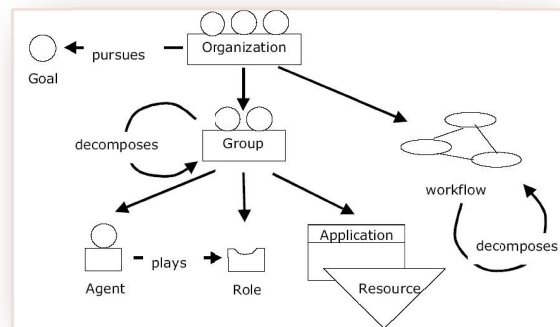


Fig. 8. Elementos del Meta-Modelo de Organización a nivel estructural

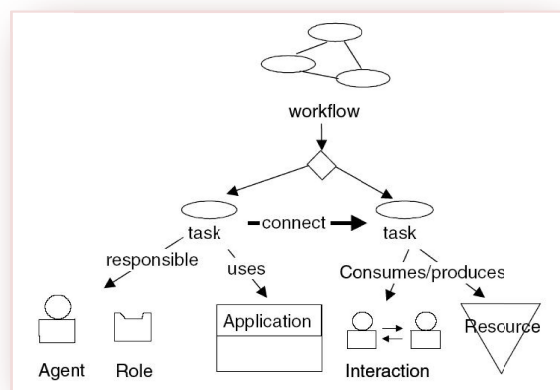


Fig. 9. Elementos a nivel de Flujo de Trabajo

3.3 CICLO DE VIDA

INGENIAS propone seguir como guía de desarrollo el Rational Unified Process (RUP) en las fases de Análisis y Diseño. La relación entre los meta-modelos y sus etapas se presenta en la siguiente tabla:

	Inicio	Elaboración	Construcción
Análisis	◆ Generar casos de uso. ◆ Esbozar la arquitectura con un modelo de organización. ◆ Generar modelos del entorno para trasladar la captura de requisitos a los modelos.	◆ Refinar casos de uso. ◆ Identificar realizaciones de los casos de uso con modelos de interacciones. ◆ Generar modelos de agente para detallar los elementos de la arquitectura. ◆ Continuar con los modelos de organización identificando flujos de trabajo y tareas. ◆ Modelos de tareas y objetivos para generar restricciones de control (objetivos principales, descomposición de objetivos). ◆ Refinar modelo de entorno para incluir nuevos elementos.	◆ Estudiar resto de casos de uso.
Diseño	◆ Generar un prototipo con herramientas de prototipado rápido.	◆ Centrar el modelo de organización en el desarrollo de flujos de trabajo. ◆ Llevar las restricciones identificadas a modelos de tareas y objetivos para dar detalles acerca de las necesidades y resultados de las tareas y su relación con los objetivos del sistema. ◆ Expresar la ejecución de tareas dentro de modelos de interacción. ◆ Generar modelos de agente para detallar patrones de estado mental.	◆ Generar nuevos modelos de agente o refinar los existentes. ◆ Depurar la organización centrando el desarrollo en las relaciones sociales.

Fig. 10. Adecuación de las etapas del RUP a los meta-modelos presentados por INGENIAS

3.4 IDK (INGENIAS DEVELOPMENT TOOLKIT)

Esta herramienta de soporte oficial de la metodología INGENIAS se caracteriza por permitir el desarrollo rápido de aplicaciones y la verificación de la especificación según las necesidades de la implementación³.

Se presenta dividida en dos partes:

- **Editor:** genera especificaciones de un SMA usando aspectos genéricos de agentes.
- **Generador de código:** facilita la transición de la especificación a la implementación de un SMA generando partes de código a partir de la interpretación de los datos contenidos en diagramas.

Por defecto, se encuentra incluido un módulo generador de código en JADE. Se trata de un módulo en desarrollo que traduce tareas, estado mental, mecanismo de selección de tareas y guardas de interacciones, pero dejando en manos del desarrollador las tareas activas que deben ejecutarse, la información que debe ir en cada entidad mental y las interacciones que desea iniciar. También incluye una verificación de la especificación incluyendo código redundante para asegurar que la implementación de los protocolos sigue lo establecido.

Aparte de este módulo para JADE, se dispone de otros módulos adicionales para diferentes plataformas, como HTML (para documentación), JADE Leap (para la ejecución de agentes en PDAs) o SOAR (para la resolución genérica de problemas (por ejemplo, en juegos)).

La adición de nuevos módulos se permite de forma automática, garantizando la finalidad de este framework que es el desacople de la especificación del SMA de la implementación concreta, sin comprometer el desarrollo en una tecnología determinada.



Fig. 11. Diferentes vistas de IDK. Definición de meta-modelos y generación de código

³ Este software se encuentra disponible en la dirección: <http://ingenias.sourceforge.net>. Se recomienda la lectura del manual para su instalación, uso, adición de módulos extra y comprensión de ejemplos (GRASIA!, 2004).

El empleo de esta herramienta CASE simplifica la complejidad del proceso de desarrollo propuesto por INGENIAS, garantiza la validez entre todos los meta-modelos diseñados y permite una clara comprensión y revisión de las funcionalidades que el sistema ha de cumplir una vez desarrollado. En consecuencia, la calidad del software final seguramente será superior.

A pesar de sus ventajosas cualidades, la aplicación todavía presenta ciertos errores que le otorgan el calificativo de inestable en su última versión (2.6) y que retrasan de forma moderada el análisis y diseño propuesto según la metodología. Éstos han sido: fallos en la carga de proyectos (aplicación bloqueada en estado “Loading”) que hace necesario el uso de backups anteriores y, por tanto, supone una pérdida del trabajo realizado hasta la fecha; inactividad del botón “undo” (deshacer) para retroceder a una fase anterior al eliminar cualquier objeto de forma equivocada y aparición de la ventana de advertencia de pérdida de información al pulsar el botón “cancelar” en una descripción o asignación de valores cuando no se han efectuado cambios.

Obviando estas faltas, que se deben a una falta de refinamiento del aplicativo, en líneas generales, la única limitación irresoluble que presenta el empleo de IDK es su implantación en grandes equipos de trabajo. Esta imperfección tiene su origen en el enfoque de la metodología ya que el reparto del diseño de los meta-modelos resulta muy complejo debido a las relaciones existentes entre los diagramas. Se trata, por tanto, de un aspecto fundamentado en la base de la metodología y no en la aplicación en sí, por lo que no se desmerecen sus características y funcionalidades generales.

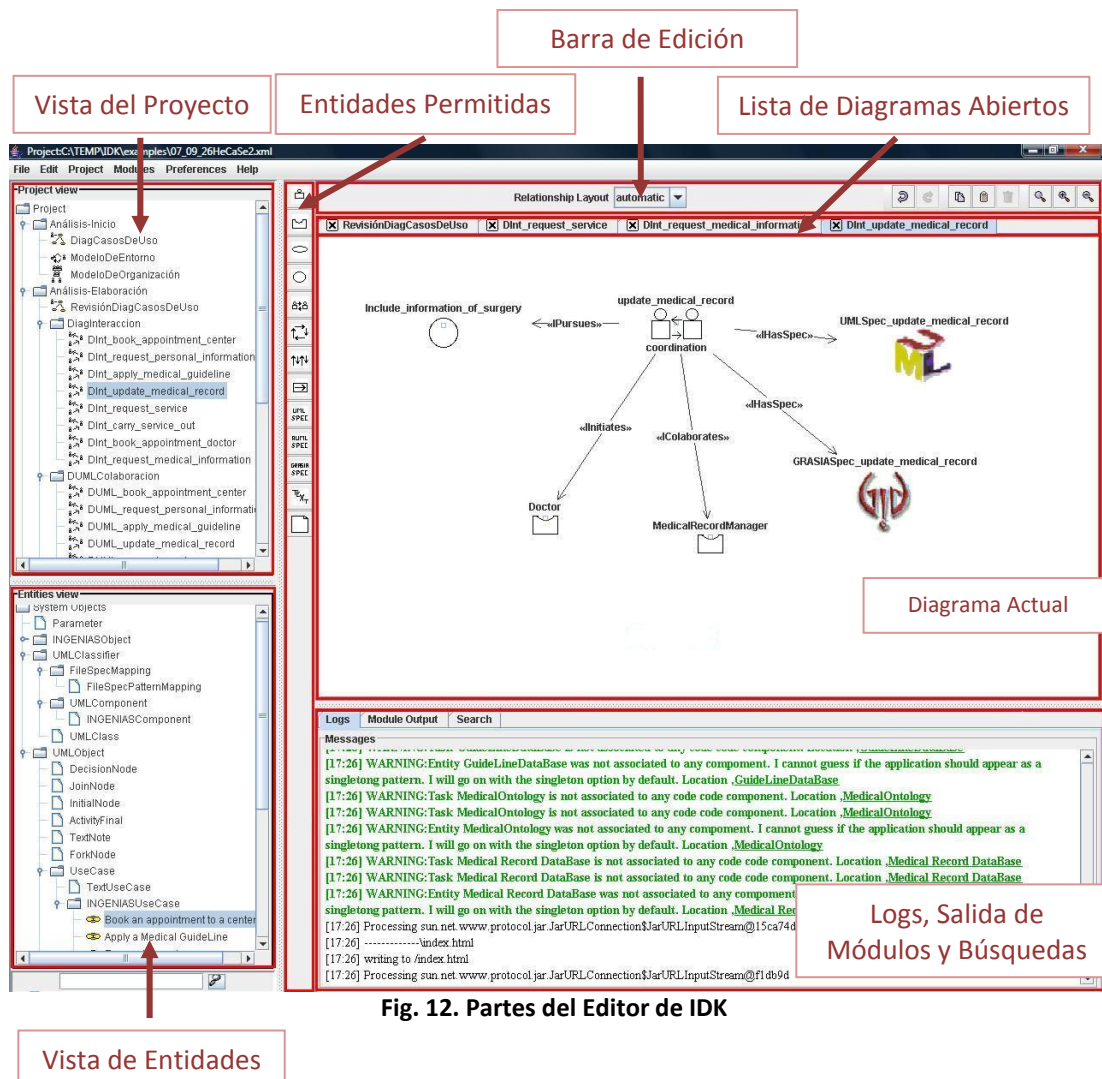


Fig. 12. Partes del Editor de IDK

4 ANÁLISIS Y DISEÑO DE UN SMA EN INGENIAS

En esta sección se muestra la aplicación de la metodología INGENIAS a un Sistema Multi-Agente específico, HeCaSe2.

HeCaSe es una plataforma orientada a proveer servicios sanitarios a habitantes y visitantes de una ciudad. Los usuarios pueden acceder a su historial, localizar información sobre los centros médicos de la ciudad y realizar reservas para especialistas. Los médicos, a su vez, disponen de accesos y modificaciones a los historiales de los pacientes durante una visita y pueden también solicitar la realización de pruebas, de forma que los resultados sean almacenados automáticamente en el historial del paciente y estén disponibles para la siguiente visita. En una versión posterior (HeCaSe2), se ha ampliado su funcionalidad, permitiendo el empleo de Guías de Práctica Clínica.

En el siguiente apartado se resumen las principales características de la herramienta y posteriormente se muestra la aplicación de INGENIAS para detallar las fases de análisis y diseño.

4.1 PLATAFORMA HECASe2

La atención sanitaria es un dominio donde algunos investigadores han aplicado diferentes técnicas y algoritmos de la Inteligencia Artificial. En concreto, la finalidad de la aplicación de Sistemas Multi-Agente es permitir implementar sistemas distribuidos con mayores beneficios frente a los centralizados. Además, empleando esta aproximación se pueden reutilizar sistemas/implementaciones existentes e incrementar así, sus funcionalidades generales.

Los principales beneficios de la aplicación de los agentes al área de la salud son:

- a) La tecnología de los agentes ofrece plataformas avanzadas para la construcción de sistemas expertos para la asistencia del personal médico en su trabajo, y
- b) Los sistemas de agentes distribuidos tienen el potencial de mejorar las acciones en las instituciones sanitarias, donde los fallos en comunicación y coordinación son importantes focos de error.

De esta forma, a continuación se presentan las características básicas de la plataforma de agentes HeCaSe que modela instituciones médicas y pacientes para ofrecer una serie de servicios al ciudadano.

4.1.1 SERVICIOS IMPLEMENTADOS

HeCaSe (Health Care Services) es una plataforma desarrollada por el grupo de SMA de la Universidad Rovira i Virgili. Su primera versión se diseñó entre 2001 y 2003 dentro del proyecto europeo *AgentCities* y su objetivo era informatizar una serie de servicios sanitarios:

- Solicitud de información sobre centros médicos disponibles en una cierta área geográfica.
- Reserva de una hora de visita para un médico especialista.
- Acceso al historial médico de un paciente.
- Actualización de los datos de un paciente por un doctor para introducir los resultados de un examen médico tras una visita.
- Acceso ubicuo a la información.

Estas funcionalidades implican un conjunto de requisitos implícitos de la aplicación:

- Uso de una ontología específica para el dominio médico.
- Implantación de medidas de seguridad en el acceso a los datos médicos de los usuarios para que ningún agente no autorizado pueda consultarlos ni modificarlos.
- Modelaje de una estructura básica de los centros médicos locales mediante agentes con diferentes roles (cada centro dispone de un conjunto de departamentos médicos, y en cada uno de ellos trabaja un grupo de doctores).
- Configuración de perfiles de usuario según su categoría para poder ofertar un servicio personalizado considerando sus preferencias y disponibilidad horaria.

En la segunda versión de esta herramienta se incluyen además:

- Gestión de Guías de Práctica Clínica. En inglés, Clinical Guideline (GL, en adelante) es una propuesta para doctores en la resolución de un determinado problema. Es una representación diseñada por expertos en el área que indican una serie de acciones, cuestiones y decisiones para una patología concreta, garantizando la calidad en los servicios.
- Automatización de los servicios médicos. Un doctor establece las próximas visitas según la disponibilidad de las pruebas que se necesiten para el diagnóstico y/o tratamiento de una enfermedad. Los agentes encargados de estos análisis (internos o externos al departamento del doctor) actualizan directamente el historial del paciente con los resultados.

La combinación de todas estas funcionalidades permite obtener un sistema que coordina servicios hospitalarios complejos y que representa una mejora de la gestión de todos los recursos médicos.

4.1.2 ARQUITECTURA

La arquitectura Multi-Agente de HeCaSe2 es la siguiente:

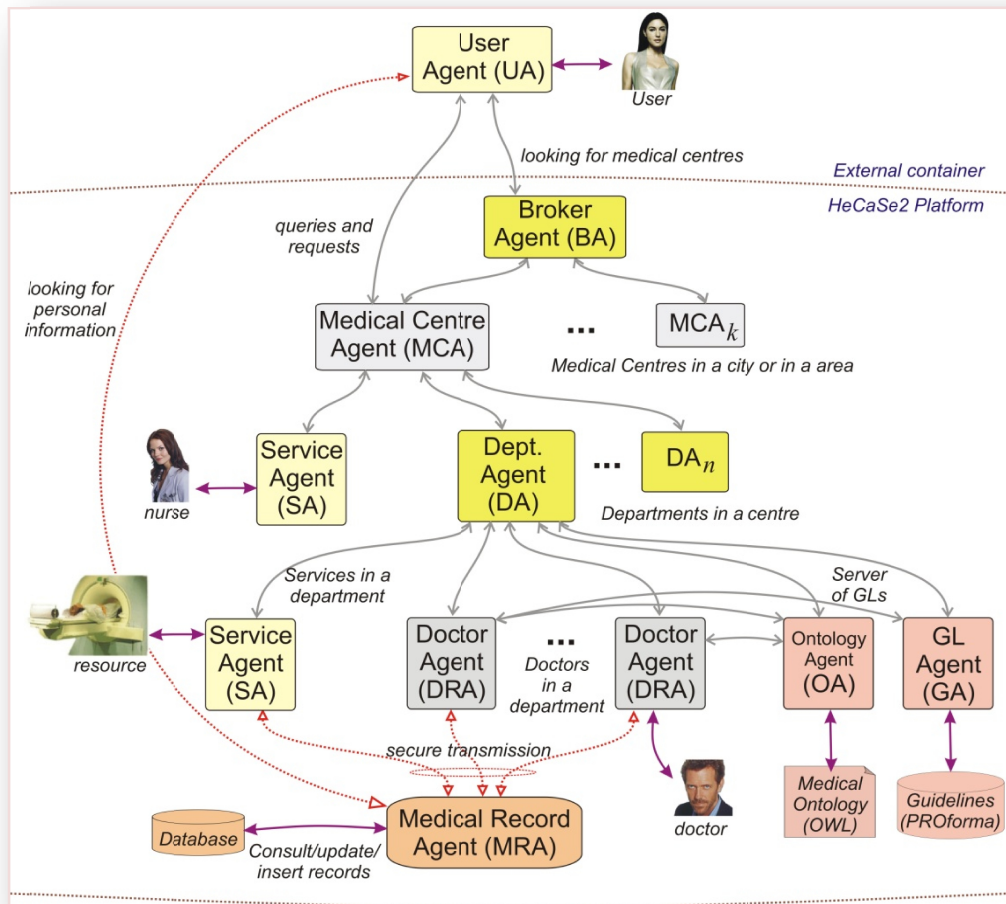


Fig. 13. Arquitectura HeCaSe2

En la cima, los pacientes son representados en el sistema mediante **"User Agents"**(UAs). Cualquier UA puede comunicarse con el **"BrokerAgent"**(BA). El BA es el nexo de unión entre usuarios y centros médicos y se emplea para descubrir información en el sistema. Todos los UAs pueden solicitar al BA centros médicos que satisfagan ciertos criterios. El BA abarca los centros métodos localizados en la ciudad o en un área. Cualquier usuario puede acceder al sistema a través de un **"Medical Centre Agent"**(MCA) que centraliza y monitoriza los accesos desde el exterior. Un MCA controla todos sus departamentos, representados median te **"Department Agents"** (DAs) y un conjunto de servicios generales (**"Service Agents"**(SAs)). Cada departamento está formado por diversos médicos (**"Doctor Agents"**(DRAs)) y más servicios específicos (también modelados como SAs). Además, en cada departamento existe un **"Guideline Agent"** (GA) que desempeña todas las acciones relacionadas con las GL, como por ejemplo, buscar el caso más adecuado, actualizando las modificaciones realizadas por un médico, etc. Este GA contiene solamente las guías relacionadas con el departamento donde se encuentra situado (el conocimiento no es visible para la entidad que lo emplea) aunque si és necesario (patologías complejas) se pueden intercambiar GLs con otros departamentos. Cada departamento también contiene un **"Ontology Agent"** (OA) que proporciona el acceso a la ontología médica diseñada y

complementa la información facilitada por el GA para el seguimiento de un GL por parte de los doctores. En la parte inferior de la arquitectura se encuentra el **“Medical Record Agent”** (MRA) que almacenan todos los registros médicos de los pacientes del centro, y el cual da un servicio seguro de control de acceso a los datos, así como de su transmisión.

4.2 APLICACIÓN DE INGENIAS SOBRE HECase2

Tal como se ha descrito anteriormente, el ciclo de vida de INGENIAS basado en el RUP propone seis etapas (tres en el Análisis y tres en el Diseño) que van creando y refinando los meta-modelos.

En la **fase Inicial**, el desarrollador necesita realizar un estudio profundo del sistema que desea modelar, de forma que sea capaz de identificar sus funcionalidades básicas y actores involucrados y así poder diseñar, en el primer paso del Análisis, el “Diagrama de Casos de Uso” (cuya finalidad es garantizar que el sistema a desarrollar es posible). A continuación, se realiza un esbozo de la organización y el entorno que tendrá el sistema. En el Diseño, se recomienda realizar un prototipo sencillo de la aplicación.

En la **fase de Elaboración** se profundiza en las características de los agentes, sus tareas y objetivos a alcanzar. Se comienza por refinar el diag. De Casos de Uso, agrupando las funcionalidades y asociando modelos de interacciones para estudiar como se llevan a cabo (con especificaciones realizadas en diagramas de Colaboración UML). Para conocer mejor los agentes de la organización de la fase anterior, se elaboran modelos de agentes que muestran objetivos propios, roles que interpretan y tareas a realizar. Todos los objetivos y tareas identificados se organizan dentro de modelos de tareas y objetivos. Posteriormente, se amplía el modelo de organización para recoger estos aspectos y estructurar las tareas en flujos de trabajo. En el Diseño, se aumenta el nivel de detalle refinando las interacciones (con Diagramas GRASIA!), los flujos de trabajo y tareas, el control del agente y el logro de los objetivos del sistema.

En la **fase de Construcción** se amplía el sistema con nuevos casos de uso secundarios siguiendo las mismas actividades que en la fase anterior. Los modelos resultantes no han de modificar sustancialmente la visión global que se tiene del sistema. En el diseño, los nuevos casos de uso pueden requerir de dependencias sociales (subordinación y/o cliente-servidor) que se tendrán que especificar.

Posteriormente, una vez concluidas estas fases, se obtiene un esquema genérico de las clases necesarias para la implementación del SMA mediante el módulo generador de código incluido en IDK.

En los apartados consecutivos, se presentan los diagramas de resultado de cada una de las fases sobre el caso de estudio concreto. Se ha utilizado para su realización, el editor de la herramienta IDK.

Para la elaboración de esta sección han sido de gran ayuda la tesis doctoral del docente investigador J.J. Gómez-Sanz (Gómez-Sanz, 2002) y los ejemplos disponibles en la web <http://grasia.fdi.ucm.es/ingenias/>.

4.2.1 FASE DE INICIO DEL RUP

4.2.1.1 ANÁLISIS

Los resultados del Análisis en la fase de Inicio han sido:

- ◆ Diagrama de Casos de Uso
- ◆ Modelo de Organización
- ◆ Modelo de Entorno

DIAGRAMA DE CASOS DE USO:

Los casos de uso que se han identificado en la aplicación HeCaSe2 han sido:

- **Book an appointment to a center** (*Reservar una cita a un centro*): un usuario puede reservar una cita tras especificar un centro médico y el departamento de consulta. Selecciona el médico que prefiera según su disponibilidad.
- **Book an appointment with the doctor** (*Reservar de una cita con el médico*): durante una visita médica, el doctor fija la siguiente consulta con el paciente. Se confirma la disponibilidad del usuario.
- **Request personal information** (*Solicitar información personal*): un usuario consulta sus datos personales e informes de sus visitas médicas.
- **Request medical information** (*Solicitar información médica*): un usuario consulta información genérica sobre centros médicos.
- **Request patient list** (*Solicitar lista de pacientes*): un médico desea su lista de visitas.
- **Apply Medical GuideLine** (*Aplicar guía de práctica clínica*): un médico aplica una GL (Guideline, Guía de Práctica Clínica) para ratificar la dolencia de un paciente.
- **Update medical record** (*Actualizar un registro*): tras una visita, un médico debe de actualizar el registro médico del paciente con los nuevos datos de la consulta.
- **Request a service** (*Solicitar un servicio*): un médico asigna un servicio a un paciente para conocer los resultados de una prueba concreta que él no puede realizar. Se comprueba si se puede solicitar el servicio a nivel de departamento, y en caso negativo, se solicita a nivel de centro médico.
- **Carry a service out** (*Realizar un servicio*): un gestor de servicio (de nivel de departamento o de centro) actualiza el registro médico de un paciente con el resultado del servicio médico efectuado.

Debido a que HeCaSe2 es una aplicación profundamente estudiada (indicar que se ha comenzado a desarrollar en el año 2003), la lista de casos de uso no se verá alterada ni ampliada en la fase de Construcción de este estudio. Esta circunstancia no imposibilita la inclusión de nuevas funcionalidades a la plataforma a posteriori, sino que indica que no tendrán lugar en este caso concreto. De estimarse necesarias en un futuro, podrían darse dos situaciones:

- ✓ Que este reajuste fuese secundario, por lo que bastaría con extender la fase de Construcción;
- ✓ Que la nueva funcionalidad se considerase primaria, por lo que habría que recomponer el Diagrama de Casos de Uso Inicial y, con ello, todo el análisis y diseño subsiguiente.

Se expone la relación de los casos de uso con los actores del sistema en el siguiente diagrama:

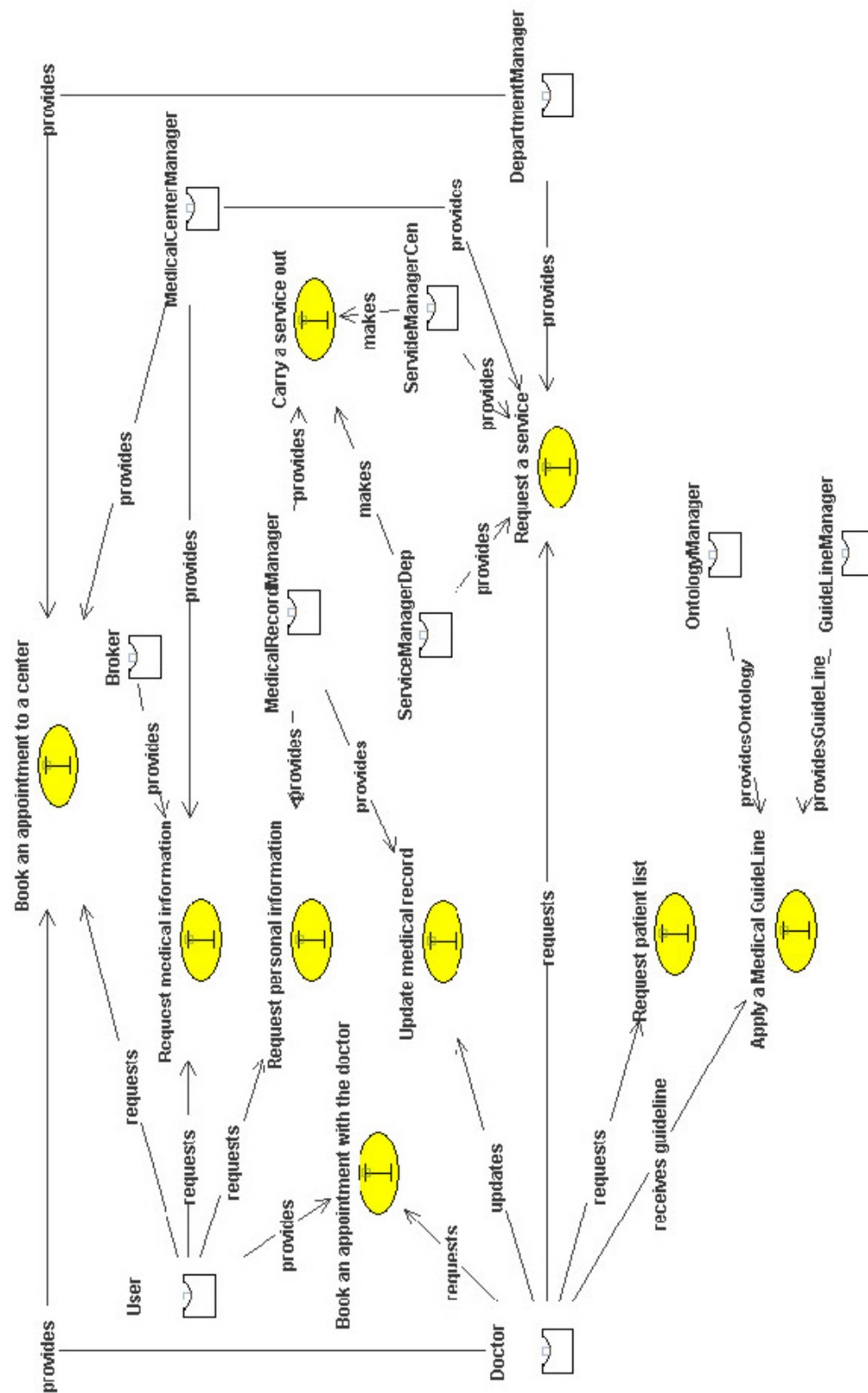


Fig. 14 . Diagrama de Casos de Uso Inicial

MODELO DE ORGANIZACIÓN:

El sistema se presenta jerarquizado. En el nivel organizacional, se presenta la Corporación Médica que controla la Administración, los pacientes y los centros médicos.

La Administración está integrada por el Broker (Enlace Usuario-Centros) y el gestor de los registros médicos. Por otro lado, cada centro médico presenta su propio gestor, un conjunto de servicios y un conjunto de departamentos. A su vez, cada departamento dispone también de su propio gestor, un conjunto de servicios propios, un conjunto de médicos asociados, un gestor de las Guías de Práctica Clínica de su área y un gestor de su Ontología específica.

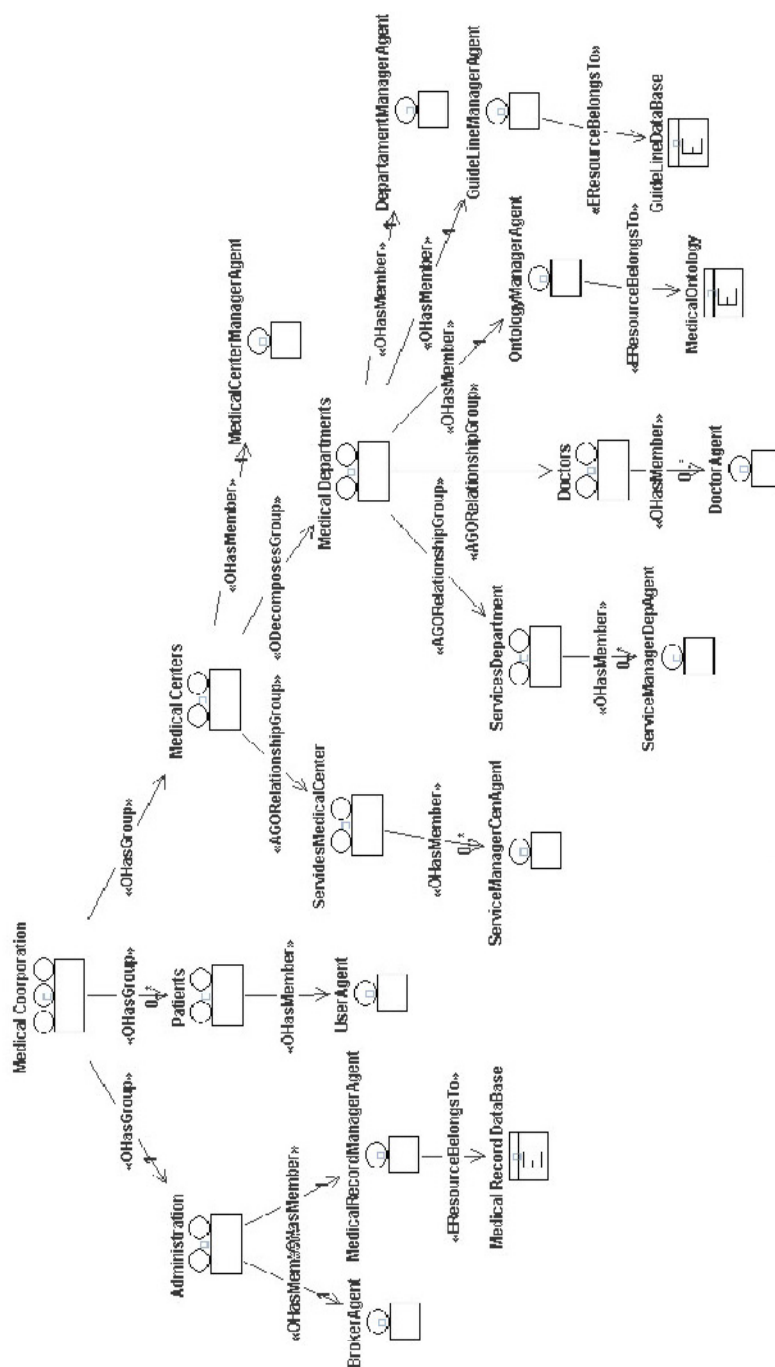


Fig. 15. Modelo de Organización

MODELO DE ENTORNO:

El sistema se interrelaciona con tres aplicaciones, en este caso, **Bases de Datos** (“*Data Base*”): una para los registros médicos (“*Medical Record*”), otra de las Guías de Práctica Clínica (“*GuideLine*”) y otra para las Ontologías (“*Medical Ontology*”). La primera es única para todo el sistema, mientras que las otras dos existen para cada departamento de los centros médicos.

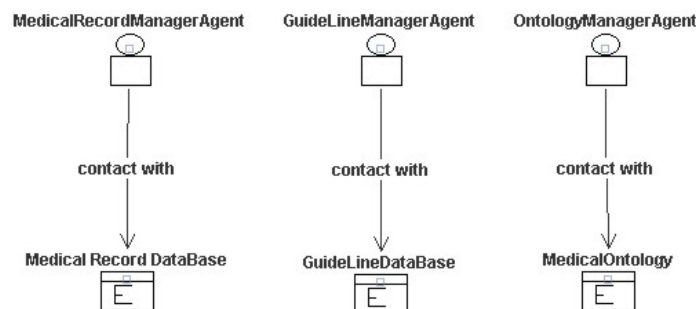


Fig. 16. Modelo de Entorno

4.2.1.2 DISEÑO

En esta fase, se recomienda diseñar un prototipo base del sistema final. Como la plataforma HeCaSe2 es una aplicación existente, ya se conocen sus características básicas e interfaces de usuario finales, que son coincidentes.

Este hecho no quiere significar que el sistema completo final haya de corresponderse con el real, ya que probablemente gracias a la aplicación de la metodología previa a la implementación, el resultado del esqueleto de código interno final de los agentes sea más escalable y mejor estructurado.

Dos capturas de la plataforma HeCaSe2 se presentan a continuación:

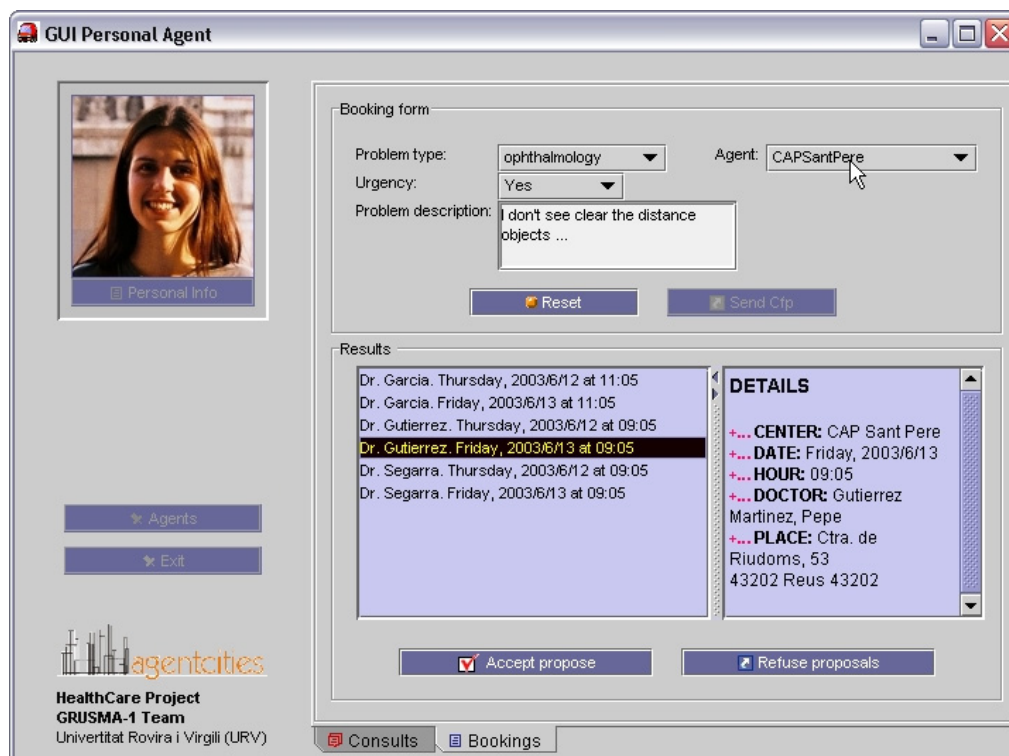


Fig. 17. GUI de HeCaSe2. Usuario PACIENTE (USER). Reserva de cita

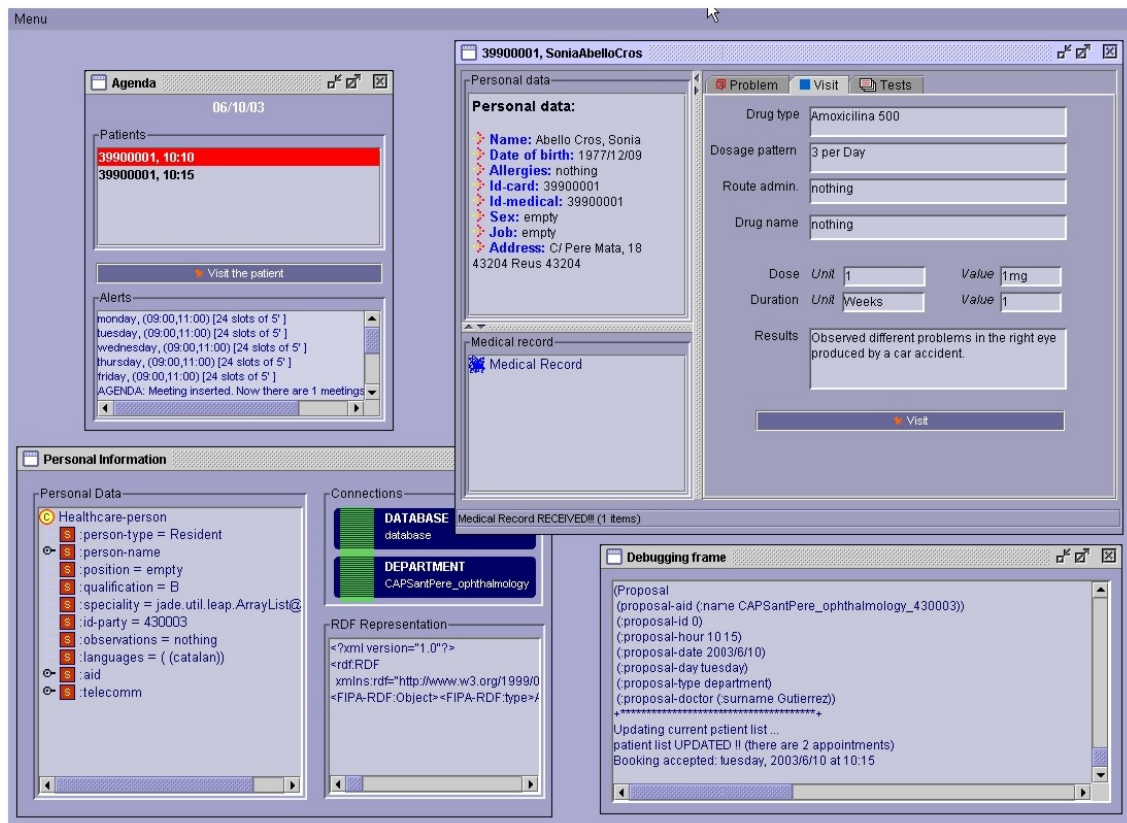


Fig. 18. GUI de HeCaSe2. Usuario DOCTOR. Informe de Visita

4.2.2 FASE DE ELABORACIÓN DEL RUP

4.2.2.1 ANÁLISIS

Los resultados del Análisis en la fase de Elaboración han sido:

- ◆ Revisión de los Casos de Uso
- ◆ Modelo de Interacciones
- ◆ Diagramas de Colaboración UML
- ◆ Modelo de Agentes
- ◆ Modelo de Objetivos y Tareas
- ◆ Diagramas de Tareas
- ◆ Modelo de Organización (refinado con workflows)

REVISIÓN DE LOS CASOS DE USO:

Se comprueba el Diagrama de Casos de Uso realizado en el Análisis Inicial y se agrupan los casos de uso conforme a su analogía. Además, para cada caso de uso que requiera una interacción entre agentes, ésta se debe de especificar. (Según los casos de uso descubiertos previamente, el único que no desencadena una interacción es “*Request patient list*” ya que el médico consulta su propia lista de pacientes sin establecer comunicación).

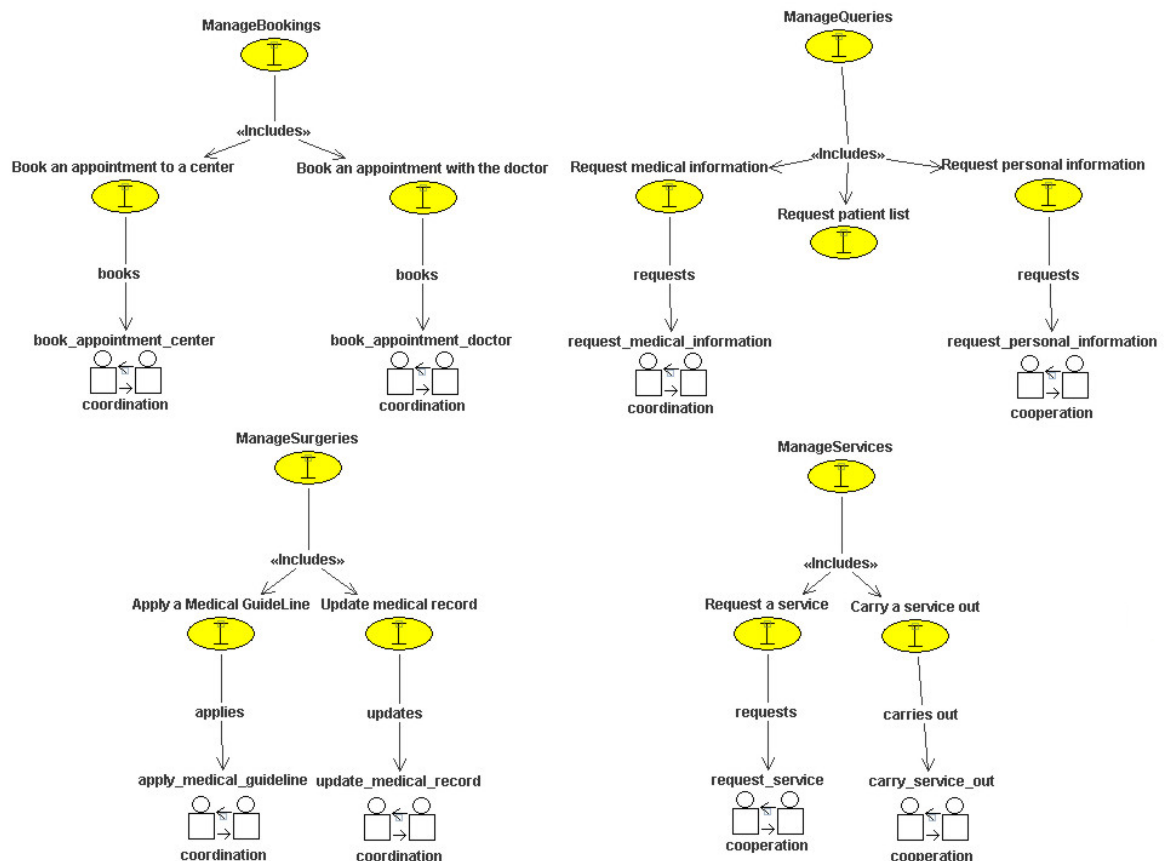


Fig. 19. Revisión de los Casos de Uso

MODELO DE INTERACCIONES:

Para cada comunicación entre dos o más agentes se define un Modelo de Interacciones. Los más interesantes han sido los que se derivan de los casos de uso “*Book an appointment to a center*” (v. Fig. 20) y “*Request a service*” (v. Fig. 21) puesto que son los que más agentes de diferente tipo involucran.

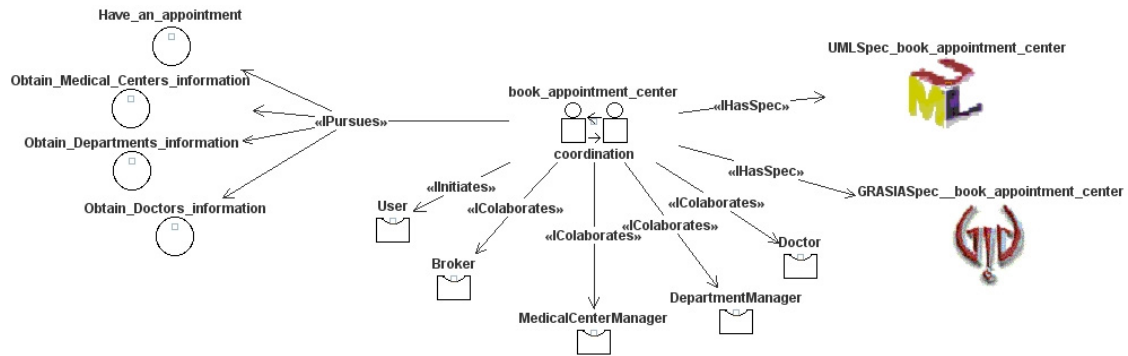


Fig. 20. Modelo de Interacción "Book an appointment to a center"

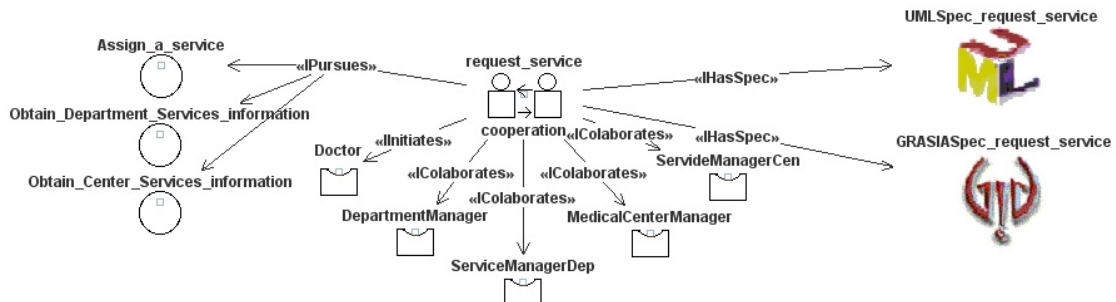


Fig. 21. Modelo de Interacción "Request a service"

En términos generales, las relaciones de interacción han de cubrir eficazmente los siguientes objetivos:

Interacción	Objetivos
Book an appointment to a center	Have an appointment Obtain Medical Centers information Obtain Departments information Obtain Doctors information
Book an appointment with the doctor	Fix next visit Date
Request Medical information	Obtain Medical information
Request Personal information	Obtain Personal information Obtain Medical Record information (<i>User</i>) Obtain Medical Record (<i>MedRecordMang</i>)
Apply Medical GuideLine	Apply GuideLine (<i>Doctor</i>) Obtain GuideLine (<i>GuideLineMang</i>) Obtain Ontology term (<i>OntologyMang</i>)
Update Medical Record	Include surgery information (<i>Doctor</i>) Obtain Medical Record (<i>MedRecordMang</i>) Update Medical Record with surgery information (<i>MedRecordMang</i>)
Request a Service	Assign a service Obtain Department Services information Obtain Medical Center Services information
Carry a Service out	Make Department Service Make Medical Center Service Add service result to a Medical Record

DIAGRAMAS DE COLABORACIÓN UML DEL MODELO DE INTERACCIONES:

Para cada interacción definida anteriormente, en esta etapa se describe una especificación inicial mediante Diagramas de Colaboración UML. Estos diagramas presentan de forma clara la relación los pasos de la interacción. Están etiquetados siguiendo las recomendaciones de UML, indicando el número de secuencia y guardas para el envío de cada mensaje.

Se muestran a continuación algunos diagramas de colaboración UML realizados para esta aplicación. El primero de ellos ("Book an appointment to a center") ha sido la especificación más compleja de los Modelos de Interacción. En él intervienen un total de 5 roles y se han obviado los repudios para facilitar su comprensión. Los dos siguientes son más simples y reflejan el intercambio de mensajes entre 2 o 3 roles respectivamente.

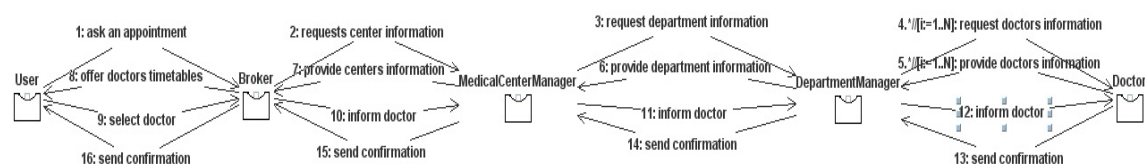


Fig. 22. Diagrama de Colaboración UML "Book an appointment to a center" (sin repudio)

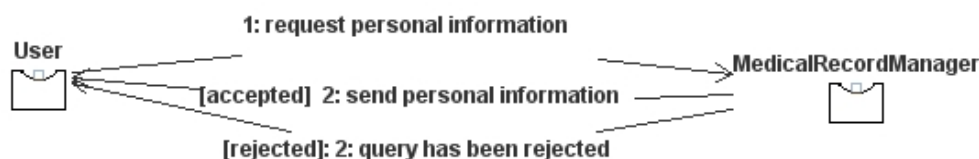


Fig. 23. Diagrama de Colaboración UML "Request Personal information"

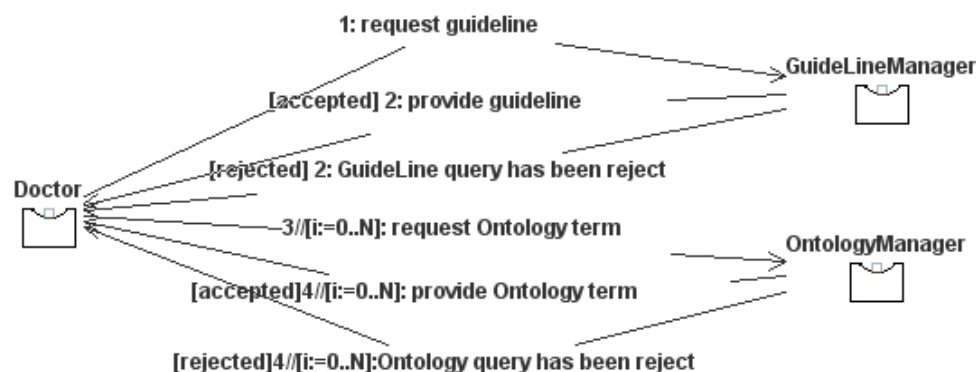


Fig. 24. Diagrama de Colaboración UML "Apply GuideLine"

MODELO DE AGENTES:

Los agentes identificados en el sistema, se presentan asociados a los siguientes objetivos finales:

Agente	Objetivos
User	Have an appointment Obtain Personal information Obtain Medical Record information Obtain Medical information
Doctor	Fix next visit Date Include surgery information Obtain patient list Assign a service Apply GuideLine
Broker	Obtain Medical Centers information
Medical Center Manager	Obtain Departments information Obtain Medical Center Services information
Department Manager	Obtain Doctors information Obtain Department Services information
Service Medical Center Manager	Make Medical Center Service
Service Department Manager	Make Department Service
GuideLine Manager	Obtain GuideLine
Ontology Manager	Obtain Ontology term
Medical Record Manager	Add service result to a Medical Record Update Medical Record with surgery information Obtain Medical Record

Se incluyen los modelos de algunos agentes de interés junto su declaración textual de autonomía e inteligencia:

▪ **Agente USER:**

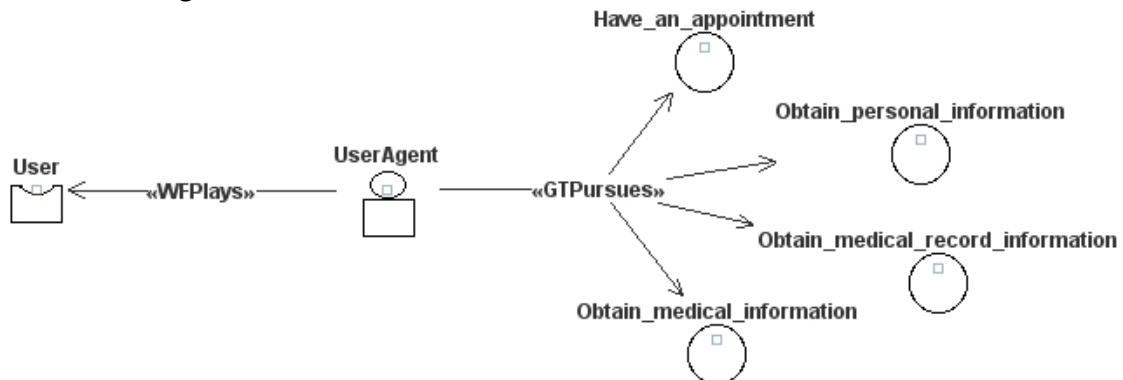


Fig. 25. Modelo de Agente "UserAgent"

Autonomía	Su objetivo es prestar todos los servicios a un paciente.
Inteligencia	Su funciones principales son: ◆ Determinar un horario de consulta compatible con la disponibilidad del paciente. ◆ Mostrar todos son datos (tanto estáticos (datos personales) como dinámicos (registro médico)). ◆ Mostrar toda la información referente a centros médicos, departamentos y médicos.

▪ **Agente DOCTOR:**

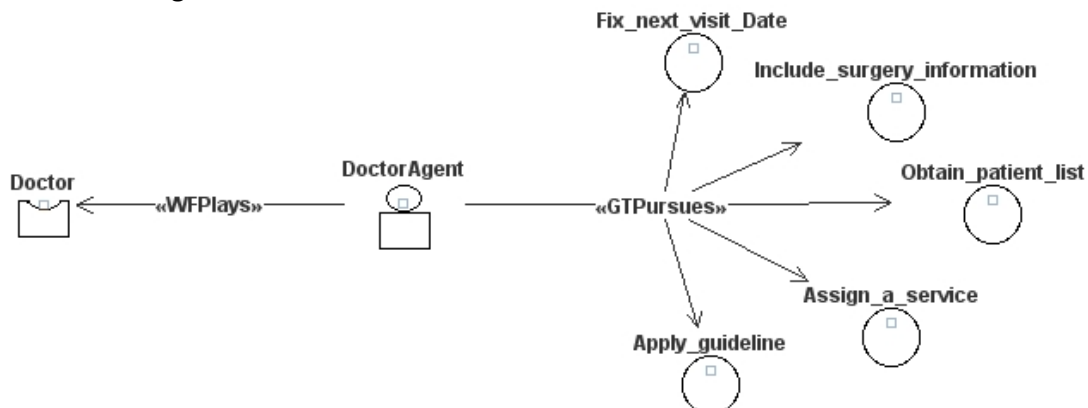


Fig. 26. Modelo de Agente "DoctorAgent"

Autonomía	Su objetivo es prestar todos los servicios a un médico.
Inteligencia	Sus funciones principales son: ◆ Capacidad para determinar un horario compatible para la siguiente visita de un paciente. ◆ Actualización de los registros médicos de los usuarios tras una visita. ◆ Disponibilidad de consulta de su lista de pacientes. ◆ Asignación de servicios (a nivel centro médico y a nivel departamento). ◆ Aplicación de GLs.

▪ **Agente DEPARTMENT MANAGER:**

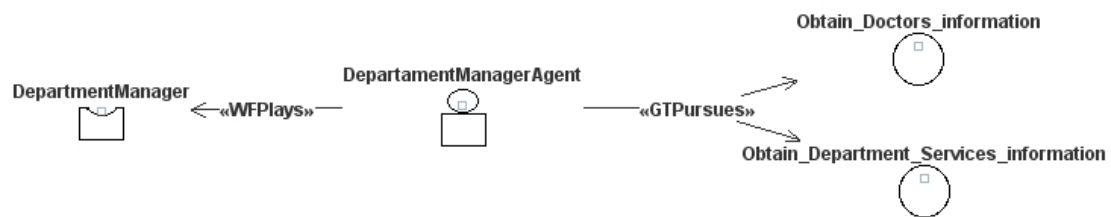


Fig. 27. Modelo de Agente "DepartmentManagerAgent"

Autonomía	Su objetivo es gestionar los departamentos médicos facilitando información sobre los médicos y servicios en él existentes.
Inteligencia	Su función principal es proporcionar información sobre los médicos y servicios de su departamento.

DIAGRAMAS DE OBJETIVOS:

A continuación se muestra la relación existente entre todos los objetivos del sistema. Los objetivos definidos se han obtenido tanto de los Diagramas de Agentes como de los Diagramas de Interacciones:

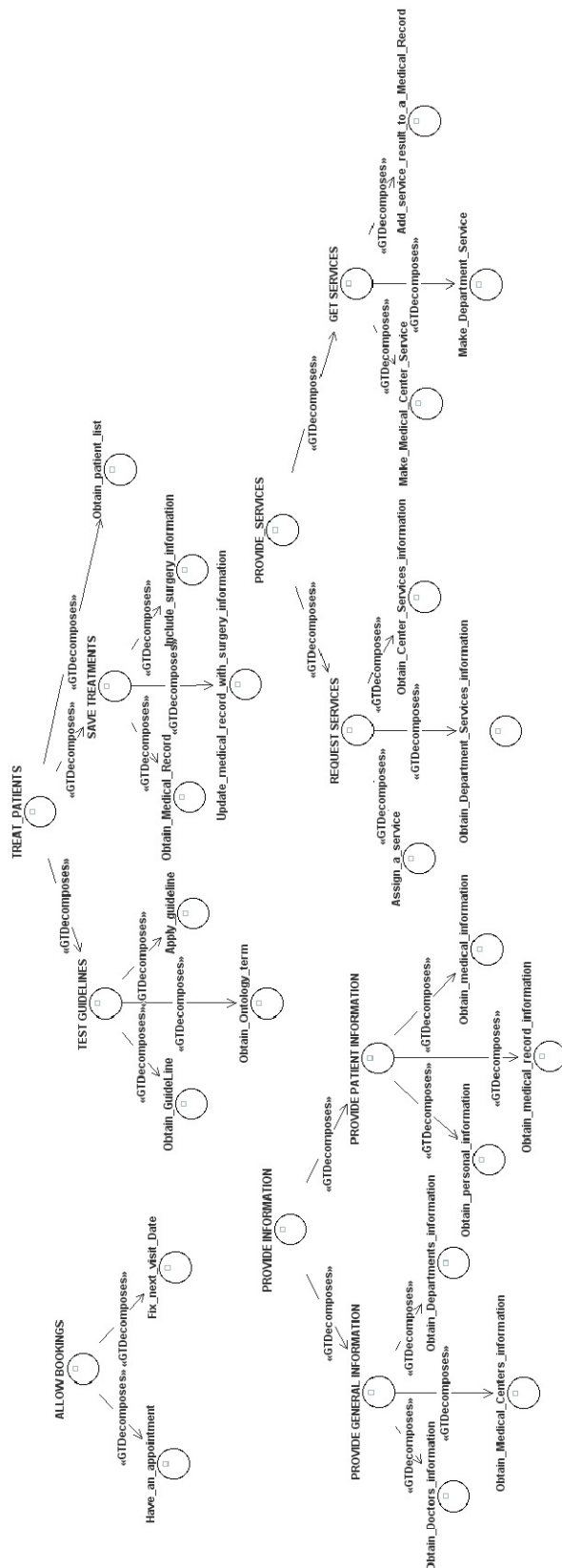


Fig. 28. Modelo de Objetivos

ASOCIACIÓN DE TAREAS A OBJETIVOS:

Objetivos	Tareas
Have an appointment	Book an appointment
Obtain personal information	Query personal information
Obtain Medical Record information	Request Medical Record information
Obtain Medical information	Request Medical information
Fix next visit Date	Fix next visit Date
Include surgery information	Include surgery information
Obtain patient list	Query patient list
Assign a service	Assign a service
Apply GuideLine	Apply GuideLine
Obtain Medical Centers information	Request Medical Centers information
Obtain Departments information	Request Departments information
Obtain Medical Center Services information	Request Medical Center Services information
Obtain Doctors information	Request Doctors information
Obtain Department Services information	Request Department Services information
Make Medical Center Service	Make Medical Center Service
Make Department Service	Make Department Service
Obtain GuideLine	Query GuideLine
Obtain Ontology term	Query Ontology term
Add service result to a Medical Record	Add service result to Medical Record
Update Medical Record with surgery information	Update Medical Record with Surgery information
Obtain Medical Record	Query Medical Record

DIAGRAMAS DE TAREAS:

Cada tarea se especifica para mostrar su/s objetivo/s, precondiciones, postcondiciones, recursos y módulos software.

Del conjunto de tareas surgidas en esta etapa y listadas anteriormente, se han seleccionado tres para exponer sus diagramas y aclarar sus relaciones de dependencia. Éstas han sido:

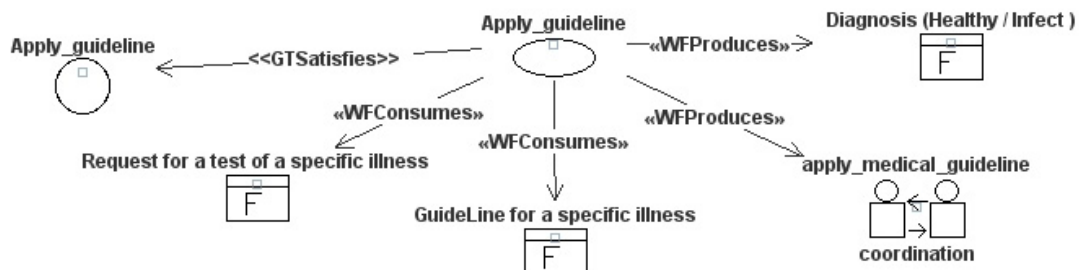


Fig. 29. Diagrama de la Tarea "Apply Guideline"

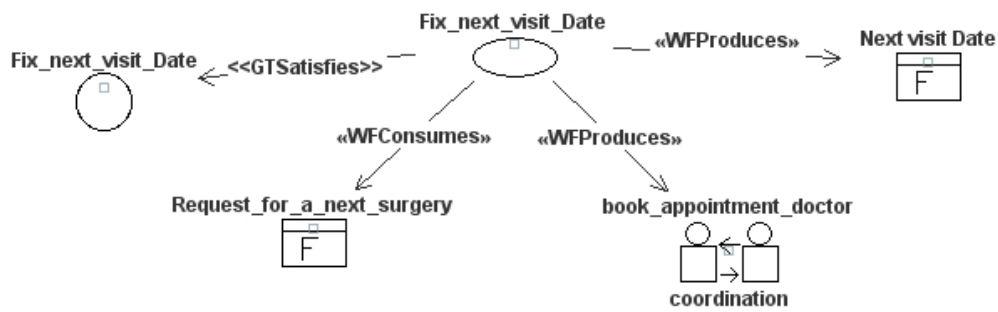


Fig. 30. Diagrama de la Tarea "Fix next visit Date"

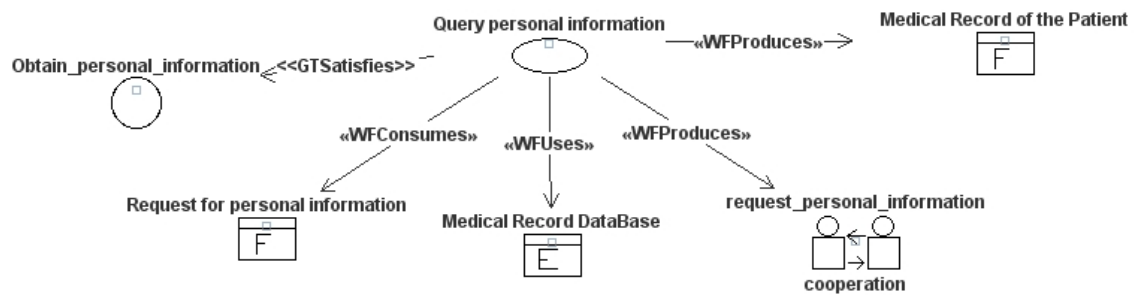


Fig. 31. Diagrama de la Tarea "Query personal information"

NUEVOS ELEMENTOS IDENTIFICADOS EN EL ENTORNO:

El Modelo de Entorno se ha de revisar con respecto al diseñado en la fase Inicial, pero no se han detectado modificaciones que alteren su diseño inicial.

MODELO DE ORGANIZACIÓN REFINADO (INDICANDO WORKFLOWS):

En este modelo se incluyen los flujos de trabajo asociados a la organización.

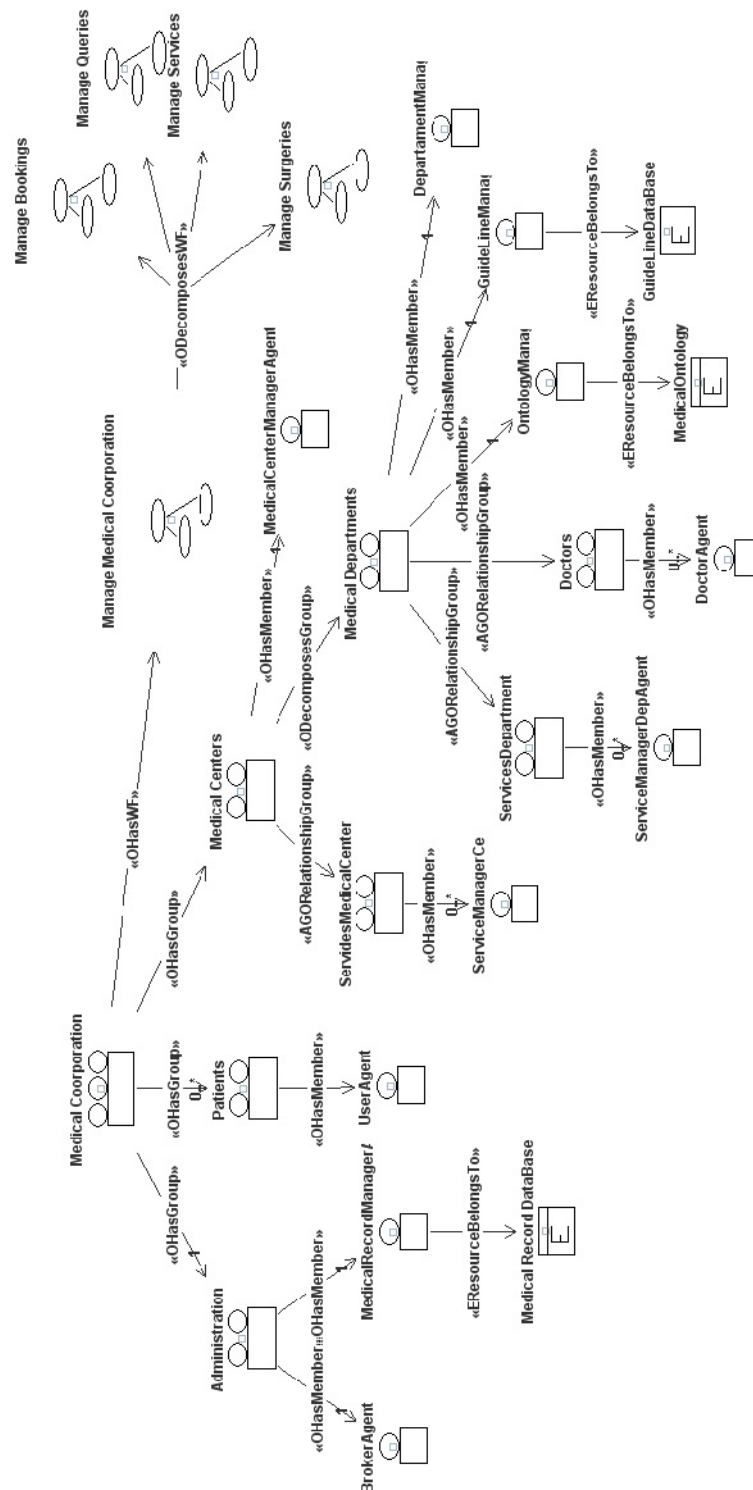


Fig. 32. Modelo de Organización con WorkFlows

4.2.2.2 DISEÑO

Los resultados del Diseño en la fase de Elaboración han sido:

- ◆ Diagramas de Especificación GRASIA! del Modelo de Interacciones
- ◆ Descomposición de Flujos de Trabajo (Modelo de Objetivos y Tareas)
- ◆ Modelo de Entorno (refinado)⁴
- ◆ Patrones del estado mental de los agentes

DIAGRAMAS DE ESPECIFICACIÓN GRASIA! DEL MODELO DE INTERACCIONES:

Las especificaciones GRASIA! permite especificar con un mayor nivel de detalle las interacciones gracias al empleo de “unidades de interacción” (que representan el paso de un mensaje, la lectura/escritura en un espacio compartido, una invocación remota o una referencia a otra interacción). Se integran, además, las tareas involucradas en la interacción.

En la siguiente figura se expone la especificación GRASIA! para el Modelo de Interacción “Update Medical Record information” efectuada por un médico tras la visita de un paciente.

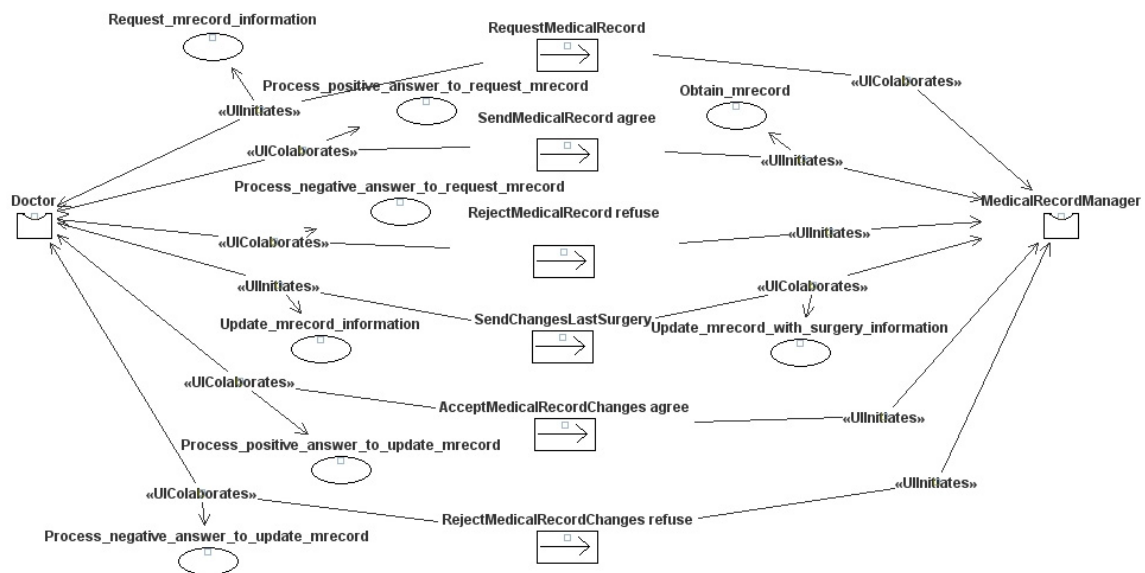


Fig. 33. Especificación GRASIA! "Update Medical Record"

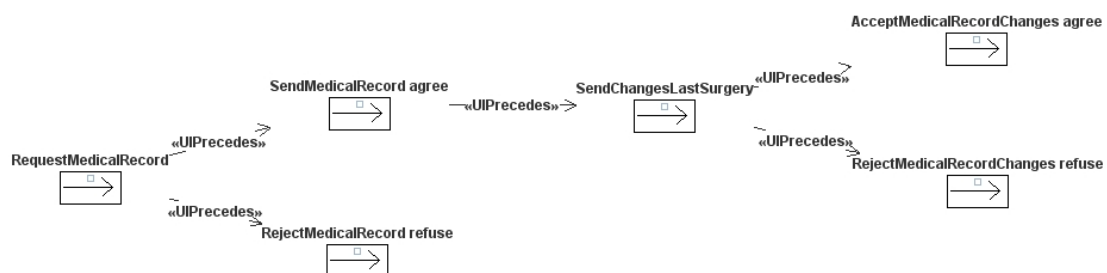


Fig. 34. Relaciones de precedencia entre las Unidades de Interacción de la especificación GRASIA! "Update Medical Record"

Como se puede observar, en estos diagramas aparecen nuevas tareas no identificadas anteriormente (relacionadas con el role Doctor). Este factor se debe al refinamiento ya que se

⁴ En esta etapa tampoco ha sido necesario volver a refinar el Modelo de Entorno con respecto al diseñado anteriormente.

corresponden con subtareas de la tarea “*Include surgery information*” (tarea global realizada por el Doctor en esta interacción).

DESCOMPOSICIÓN DE LOS FLUJOS DE TRABAJO (“WORKFLOWS”):

Debido al gran número de tareas existentes, se requiere una organización de las mismas en flujos de trabajo. Esta estructuración en forma de jerarquía muestra como el nivel de detalle que se va alcanzando es superior.

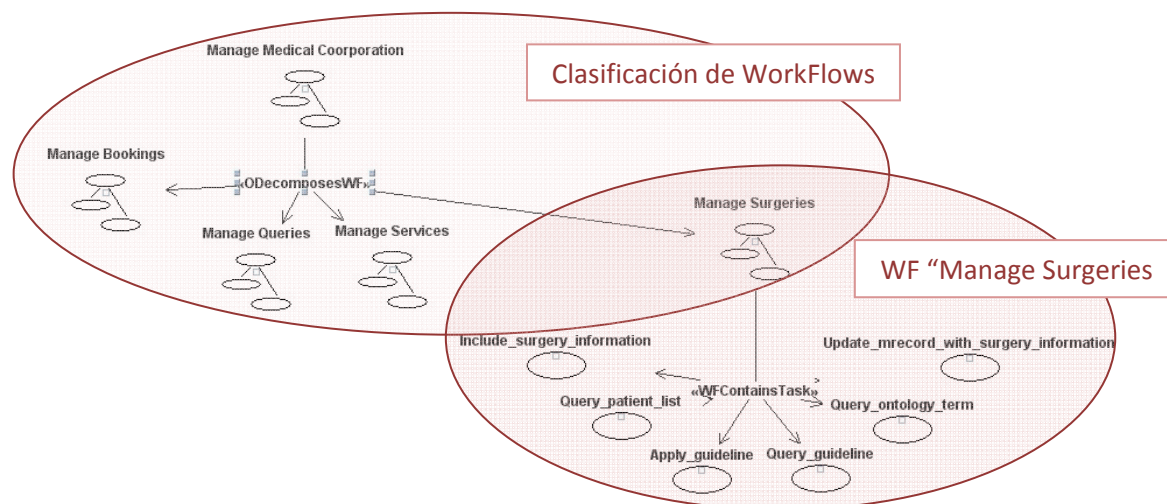


Fig. 35. Clasificación y Descomposición de Flujos de Trabajo

En la tabla siguiente se visualiza la clasificación en flujos de trabajo que se ha considerado y las tareas integran cada grupo:

Flujos de Trabajo (WorkFlows)	Tareas
Manage BOOKINGS	Book an appointment Fix next visit Date
Manage QUERIES	Query personal information Request Medical Record information Request Medical information Request Medical Centers information Request Departments information Request Doctors information Query Medical Record
Manage SERVICES	Assign a service Make Medical Center Service Make Department Service Add service result to Medical Record Request Medical Centers Services information Request Departments Services information
Manage SURGERIES	Include surgery information Query patient list Apply GuideLine Query GuideLine Query Ontology term Update Medical Record with surgery information

PATRONES DEL ESTADO MENTAL DE LOS AGENTES:

Cada modelo de agente se puede refinar mediante *Patrones del Estado Mental*. Gracias a éstos, se puede definir el estado mental de un agente como una agregación de entidades mentales dependientes del momento concreto en su vida en el que se encuentre (inicial o intermedio).

Debido al gran número de situaciones posibles existentes en este caso de estudio, podrían definirse una gran cantidad de diagramas de estados mentales para cada uno de los agentes incluidos.

Un caso interesante es el que se produce en actualización de un registro médico de un paciente por un especialista:

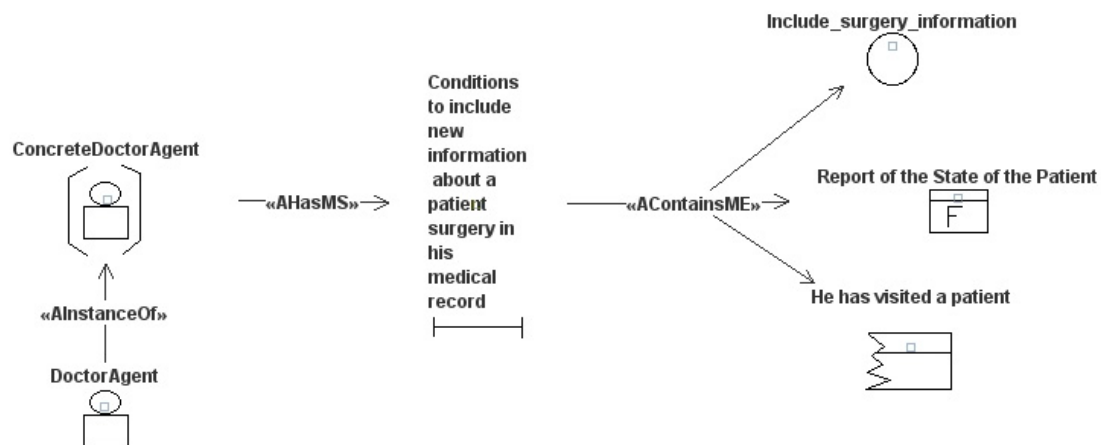


Fig. 36. Ejemplo del estado mental requerido por el Agente Doctor para actualizar el registro médico de un paciente tras una visita

4.2.3 FASE DE CONSTRUCCIÓN DEL RUP

Esta fase no se ha realizado para este caso de estudio debido a que no han surgido ampliaciones ni/o modificaciones sobre las especificaciones planteadas inicialmente. El ámbito de la plataforma HeCaSe2 ha estado perfectamente definido y delimitado desde el comienzo, por lo que en la fase de Construcción ya se han tenido que valorar todas las funcionalidades del sistema, sin que posteriormente, surgiesen nuevos cambios.

Este factor ha simplificado la aplicación de la metodología INGENIAS, aunque no por ello la fase de Elaboración ha sido menos costosa, tal y como queda constancia en el presente documento. El reenfoque de una plataforma según la aplicación de una metodología tan detallada como es INGENIAS, supone la valoración de muchos aspectos que durante una implementación directa han pasado inadvertidos.

4.2.4 IMPLEMENTACIÓN

Esta herramienta CASE permite obtener una estructura del proyecto a desarrollar a partir de los diagramas y meta-modelos realizados en las fases de Análisis y Diseño gracias al módulo “*INGENIAS Agent Framework Generator*” (IAF). Aunque el código no es definitivo, simplifica en gran medida la labor de implementación y facilita la distribución de las tareas dentro de un equipo de trabajo.

Las nuevas clases generadas son específicas del dominio del problema. De los meta-modelos diseñados en el editor se toman las entradas y salidas de las tareas, el despliegue de agentes, los objetivos, las entidades que forman parte del estado mental y las comunicaciones.

Esta parte dependiente del dominio aparece como instancias de las plantillas XML de *modules/srciaif/templates* que se depositan en los directorios:

- **outputjade:** instancias modificables por los desarrolladores,
- **outputperm:** instancias no sobre-escribibles por los desarrolladores,

y han sido rellenas con información extraída en la especificación.

Para su correcto funcionamiento es necesario que se incluyan al proyecto las clases independientes de la aplicación específica, es decir, reutilizables. En concreto, éstas son:

- Carpeta **lib:** contiene los ficheros jar básicos para cualquier aplicación de SMA (por ejemplo: jade.jar, jadeTools.jar, http.jar o iiop.jar).
- Fichero **_modaif.jar:** binarios de la carpeta *modules/srciaif* que contienen el código reutilizable de los agentes según el IAF (gestor de estado mental y procesador de estado mental). Se encuentra en la carpeta **ext** de IDK.

5 PLANIFICACIÓN Y REQUERIMIENTOS DEL PROYECTO

5.1 PLANIFICACIÓN

El proyecto se estructuró inicialmente en las siguientes fases:

Parte 1: Estado del Arte de Metodologías de Agentes

- ✓ Estudio de las metodologías existentes para el desarrollo de Sistemas Multi-Agente y generación de una jerarquía según aspectos comunes/no comunes.
- ✓ Síntesis de las principales metodologías.
- ✓ Comparativa.
- ✓ Modelaje iterativo e incremental que amplíe/retoque todos los aspectos presentados.

Parte 2: Caso práctico de aplicación

- ✓ Estudio de la herramienta HeCaSe2.
- ✓ Selección de la metodología óptima para dicha plataforma.
- ✓ Estudio en profundidad de la metodología seleccionada.
- ✓ Realización del análisis y diseño.
- ✓ Modelaje iterativo e incremental que amplíe/retoque las fases anteriores.

Posteriormente se advirtió que era necesario incluir una tercera parte que tuviese como objetivo la redacción, estructuración y presentación de toda la información obtenida previamente. Este nuevo apartado abarca las siguientes fases:

Parte 3: Redacción documentación

- ✓ Redacción.
- ✓ Revisiones Documentación Final.

A continuación se muestran las estimaciones previa y real del desarrollo del proyecto.

5.1.1 ESTIMACIÓN TEMPORAL PREVIA

Task Name	Duration	Start	Finish
☒ Duración PFC	232 days	Mon 16/10/06	Tue 04/09/07
☒ PART 1 "Análisis de Metodologías para MAS"	189 days	Mon 16/10/06	Fri 06/07/07
☒ Estudio de las metodologías existentes	163 days	Mon 16/10/06	Thu 31/05/07
Estudio de las "Agent-Oriented Methodologies"	30 days	Mon 16/10/06	Mon 27/11/06
Estudio de las "Agent-Oriented Methodologies" (Cont)	20 days	Thu 01/02/07	Thu 01/03/07
Estudio de las "Organizational Methodologies"	45 days	Thu 29/03/07	Thu 31/05/07
Evaluación/Comparativa	40 days	Thu 01/02/07	Thu 29/03/07
Evaluación / Comparativa (Cont)	25 days	Tue 01/05/07	Tue 05/06/07
Redacción Reporte Técnico URV	33 days	Mon 21/05/07	Thu 05/07/07
Presentación Report URV	1 day	Thu 05/07/07	Fri 06/07/07
☒ PART 2: "Caso de estudio plataforma HeCaSe2"	45 days	Mon 02/07/07	Fri 31/08/07
Introducción caso práctico: HeCaSe2	5 days	Mon 02/07/07	Fri 06/07/07
Elección Metodología para caso práctico	3 days	Mon 09/07/07	Wed 11/07/07
Aplicación Metodología para caso práctico	36 days	Fri 13/07/07	Fri 31/08/07
Revisión Documentación Completa PFC	2 days	Sat 01/09/07	Tue 04/09/07

Fig. 37. Tabla de tareas del proyecto (Estimación Previa)

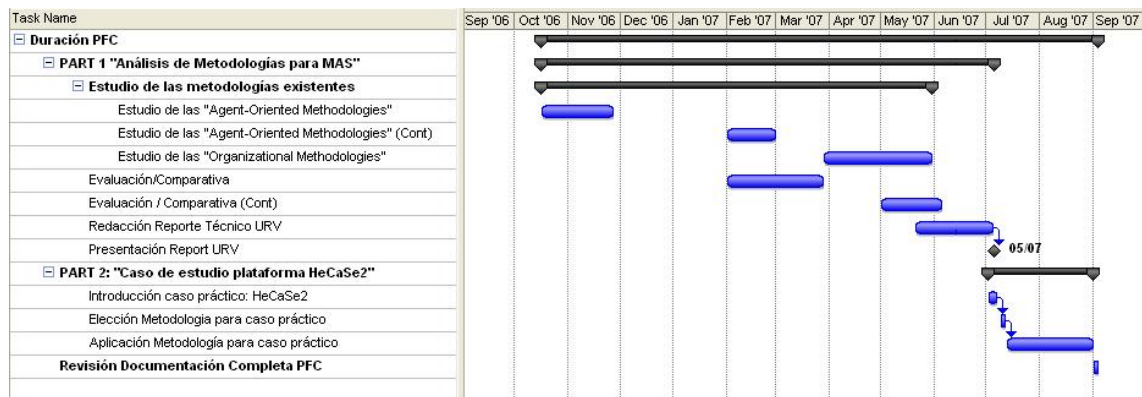


Fig. 38. Diagrama de Gantt (Estimación Previa)

5.1.2 ESTIMACIÓN TEMPORAL REAL

Task Name	Duration	Start	Finish
Duración PFC	252 days	Mon 16/10/06	Tue 02/10/07
PART 1 "Análisis de Metodologías para MAS"	206 days	Mon 16/10/06	Tue 31/07/07
Estudio de las metodologías existentes	163 days	Mon 16/10/06	Thu 31/05/07
Estudio de las "Agent-Oriented Methodologies"	30 days	Mon 16/10/06	Mon 27/11/06
Estudio de las "Agent-Oriented Methodologies" (Cont)	20 days	Thu 01/02/07	Thu 01/03/07
Estudio de las "Organizational Methodologies"	45 days	Thu 29/03/07	Thu 31/05/07
Evaluación/Comparativa	40 days	Thu 01/02/07	Thu 29/03/07
Evaluación / Comparativa (Cont)	25 days	Tue 01/05/07	Tue 05/06/07
Redacción Reporte Técnico URV	50 days	Mon 21/05/07	Mon 30/07/07
Presentación Report URV	1 day	Mon 30/07/07	Tue 31/07/07
PART 2: "Caso de estudio plataforma HeCaSe2"	36 days	Wed 01/08/07	Wed 19/09/07
Introducción caso práctico: HeCaSe2	5 days	Wed 01/08/07	Wed 08/08/07
Elección Metodología para caso práctico	1 day	Wed 08/08/07	Thu 09/08/07
Aplicación Metodología para caso práctico	30 days	Thu 09/08/07	Wed 19/09/07
PART 3: "Redacción Documentación"	35 days	Wed 15/08/07	Tue 02/10/07
Redacción	15 days	Wed 15/08/07	Tue 04/09/07
Revisiones Documento Final	20 days	Tue 04/09/07	Tue 02/10/07

Fig. 39. Tabla de tareas del proyecto (Estimación Real)

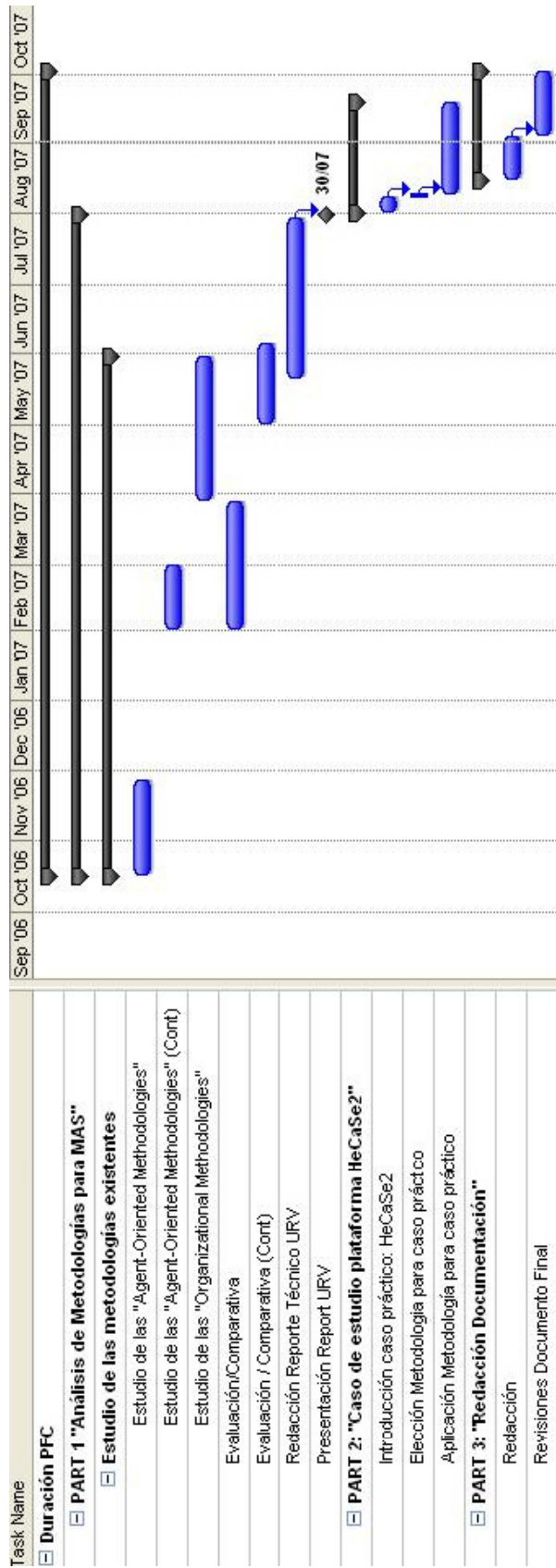


Fig. 40. Diagrama de Gantt (Estimación Real)

5.1.3 CONCLUSIONES DE LA ESTIMACIÓN TEMPORAL

Como se puede observar en los diagramas anteriores (v. Figs. 39, 40), ha existido una leve demora en la estimación que se expuso inicialmente (v. Figs.37, 38). El incremento del tiempo requerido para la primera parte debido a su complejidad retardó el comienzo de la segunda. Así mismo, la agregación de la tercera etapa de redacción y revisión que no se había especificado, que juntamente con las anteriores alteraciones, hizo necesario posponer la fecha de finalización del proyecto un mes más sobre la valoración original.

Es importante puntualizar que aunque el periodo completo de dedicación a este proyecto se aproxima a un año, durante este espacio de tiempo la alumna no ha podido dedicarse de manera exclusiva a dicha labor, aproximándose a una media laboral de 20 horas/semanales que se extendió en los últimos meses a una media de 40 horas/semanales.

5.2 HERRAMIENTAS EMPLEADAS

Para la realización de este proyecto fue necesario disponer del siguiente software completamente gratuito:

- **Microsoft Windows XP Professional Edition:** sistema operativo basado en su predecesor Windows 2000. Las letras "XP" provienen de la palabra *experience* . Windows XP es una línea de sistemas operativos desarrollada por Microsoft, orientada a cualquier entorno informático incluyendo computadoras domésticas o de negocios, computadoras portátiles, las llamadas "Tablet PC" y *media center*. Es el primer sistema operativo de Microsoft orientado al consumidor que se construye con un núcleo y arquitectura de Windows NT y que se encuentra disponible en versiones para PC de 32 y 64 Bit.
Las ediciones de Windows XP más comunes son la edición HOME (destinada al hogar) y la PROFESSIONAL (con características adicionales). Windows XP a diferencia de sus versiones anteriores presenta mejoras en la estabilidad, eficacia de Windows y reajustes de la Interfaz gráfica de usuario (GUI).
(Web: <http://www.microsoft.com/windows/products/windowsxp/default.msp>)
- **Microsoft Office 2007:**
 - **Word:** editor de documentos cuyas principales características son: interfaz Office Fluent, estilos rápidos de modificación de formatos y tablas, temas de documento, diagramas SmartArt y nuevo motor gráfico.
(Web: <http://office.microsoft.com/es-hn/word/default.aspx>)
 - **Project:** programa de administración de proyectos.
(Web: <http://office.microsoft.com/es-hn/project/default.aspx>)
- **Suscripción Springer Online:** Suscripción a la mayoría de *Journals* de esta editorial. Incluye la serie LNCS (*Lecture Notes in Computer Science*) y LNAI (*Lecture Notes in Artificial Intelligence*).
(Web: <http://www.springerlink.com>)
- **Adobe Reader 7.0 y PDFCreator 0.8.0:** lector y generador de documentos en formato pdf respectivamente.
(Webs: <http://www.adobe.com/products/acrobat/readstep2.html> , <http://sourceforge.net/projects/pdfcreator/>)
- **Ghostscript 8.56 y GSView 4.8:** intérprete y GUI de código PostScript (PS). El primero permite la conversión de este tipo de documentos a otros formatos, su visualización e impresión. Además del tratamiento de documentos PS, también es intérprete de código PDF, conversor de PDF a PS y proporciona una librería que implementa las capacidades gráficas como operaciones primitivas del lenguaje PostScript. El segundo, haciendo uso del primero, ofrece una interfaz de usuario más intuitiva y simple.
(Web: <http://pages.cs.wisc.edu/~ghost/>)
- **MiKTeX 2.6 y TeXnicCenter 7.01:** sistema compositor de tipos e IDE (*'Integrated Development Environment'*) para el desarrollo de documentos LaTeX en Windows.
(Webs: <http://miktex.org/> , <http://www.toolscenter.org/>)

- **JabRef 2.2:** aplicación gráfica gestora de bases de datos de referencias bibliográficas en formato BibTeX (estándar para la bibliografías en LaTeX). Este software proporciona una interfaz sencilla para la edición de los ficheros BibTeX, importa datos de bases de datos científicas on-line y gestiona y busca nuevos fichero BibTeX para su adición. Programado en Java requiere de la JavaVM para su ejecución.
(Web: <http://jabref.sourceforge.net/>)
- **TortoiseSVN 1.4.3:** cliente de código abierto para el control de versiones SubVersion (SVN). Los ficheros se almacenan en un *repositorio* equivalente a un servidor de ficheros ordinario, pero que recuerda los cambios que se hayan hecho a ficheros y directorios. Permite la recuperación de versiones antiguas y la evaluación de historiales de cambios. Su GUI es intuitiva y fácil de usar, evitando la ejecución de comandos por consola.
(Web: <http://tortoisesvn.net/>)
- **CorelDRAW Graphics Suite X3 (v13):** programa de diseño gráfico multiplataforma, utilizado en el ámbito de las artes gráficas, y desarrollado por Corel Corporation.
(Web: <http://www.corel.com/>)
- **Mozilla Firefox 2.0.0.6:** navegador web multiplataforma con interfaz gráfica de usuario desarrollado por la Corporación Mozilla y voluntarios externos. Existen en la actualidad las versiones 3.0 y 4.0 beta.
(Web: <http://www.mozilla.com/en-US/firefox/>)

5.3 PRESUPUESTO

El presupuesto de este proyecto, se divide en coste del hardware, software y recursos humanos necesarios para la realización del mismo.

Tipo	Item	Importe con IVA ⁵ (€)	Coste Aplicado ⁶ con IVA ⁷ (€)
Software	Microsoft Windows XP Professional Edition	0 ⁸ (Licencia URV: Sistema ELMS)	0
	Microsoft Office 2007 (Word, PowerPoint, Project)	~993 ⁹ (Agosto 2007)	~225
	Suscripción Springer Online	0 (Licencia URV)	0
	Adobe Reader 7.0 y PDFCreator 0.8.0	0	0
	Ghostscript 8.56 y GSView 4.8	0	0
	MiKTeX 2.6 y TeXnicCenter 7.01	0	0
	JabRef 2.2	0	0
	TortoiseSVN 1.4.3	0	0
	CorelDRAW Graphics Suite X3 (13)	642 (Septiembre 2007)	135,50
	Mozilla Firefox 2.0.0.6	0	0
Hardware	Portátil (Intel Pentium M 730, 512 MB DDR, Monitor 15.4")	1.100 (Mayo 2005)	275
	Multifuncional (HP F380 20ppm en negro, 14ppm en color y resolución máx.de 4.800 x 1.200 ppp)	99 (Julio 2006)	24,75
	suministros y papel	~100	~100
Recursos Humanos	Investigador/ Desarrollador (20 y 40 horas/semanales)	7 meses x 600 + 3 meses x 1.000 = 7.200	7.200
Total			~8.000 €

⁵ IVA: Impuesto sobre el Valor Añadido (16%)

⁶ Se ha realizado una amortización a 4 años del material hardware y software, de forma que en este periodo se podrían realizar aproximadamente 4 proyectos, por lo que le correspondería un valor del 25% total.

⁷ IVA: Impuesto sobre el Valor Añadido (16%)

⁸ Sistema ELMS (E-academy License Management System) de la URV para MSDN Academic Alliance: http://msdn30.e-academy.com/elms/Storefront/Storefront.aspx?campus=uriv_eim&np1=112 (permite la descarga vía web).

⁹ Precios extraídos de la dirección: <http://www.microsoft.com/canada/pricelists/> (y convertidos de dólares canadienses a euros).

6 CONCLUSIONES Y TRABAJO FUTURO

6.1 CONCLUSIONES

La realización de este proyecto ha supuesto para el alumno una aportación continua de conocimientos del área de los Agentes Software y Sistemas Multi-Agente de Inteligencia Artificial y, fundamentalmente, en el apartado de las metodologías.

En la actualidad, no se ha hallado documento alguno que presente una síntesis actualizada de todas las metodologías de referente para el desarrollo de SMA, esquematizando las principales características de cada una de las metodologías y brindando referencias al lector para que amplíe el estudio al presentado, por si fuese de su interés.

Gracias a la formalización del SMA a través de una metodología conocida, se pueden documentar todas las acciones descritas anteriormente para cada tipo de agente, así como detectar problemas de coordinación o simplificar procesos complejos en los que se deben intercambiar muchos mensajes diferentes agentes del sistema. De esta manera, también se pueden identificar ciertos patrones de comunicación para extender a otros sistemas.

En el caso concreto de este proyecto, al concluir la última etapa del ciclo de vida de la metodología INGENIAS, se dispone de todos los meta-modelos completos y de diagramas adicionales que complementan la comprensión de la aplicación HeCaSe2. Además, si el proyecto se extiende con otras funciones y/o agentes, esta ampliación podría hacerse de forma más organizada y razonable. Esta cualidad sería notablemente útil si se deseara aplicar el generador de código para obtener los esqueletos de los agentes específicos del dominio, ya que el SMA final generado sería más estructurado y escalable.

6.2 DIFICULTADES ENCONTRADAS

Inicialmente, cuando se comenzó el report técnico a primeros de Octubre de 2006, la idea propuesta como proyecto final de carrera resultó muy atrayente, aunque se desconocía el grado de dificultad que ésta implicaba.

El área de las metodologías para SMA es muy compleja debido a la inexistencia de estandarización y la basta cantidad de documentos existentes, por lo que esta situación hizo necesario un esfuerzo y trabajo extraordinarios para poder presentar una memoria interesante, completa y actualizada.

La jerarquía propuesta se obtuvo tras la profundización en el área después de meses de análisis, ya que los investigadores no tienden a ofertar clasificaciones globales del área, sino características y detalles de las metodologías específicas que ellos proponen. El framework o comparativa añadida al final del documento muestra las características comunes y no comunes, dotando al report de una mayor calidad.

Por otra parte, la segunda mitad del trabajo se centra en un estudio práctico sobre la metodología INGENIAS. En este ámbito cabe destacar el problema inherente a desarrollar con el máximo detalle los diferentes meta-modelos y, por otra, la ambigüedad de los diferentes diagramas. Poder detallar el caso práctico usando INGENIAS se ha podido realizar gracias a la colaboración de los desarrolladores de la Universidad Complutense de Madrid que resolvieron (puntualmente) diferentes cuestiones técnicas surgidas a lo largo de la implementación.

6.3 TRABAJO FUTURO

Este proyecto deja abierta una nueva línea de investigación que podría tener aplicación en un futuro próximo y resultar de interés para investigadores expertos en el área, ya que el trabajo presentado resulta especialmente motivador si se desea profundizar en las metodologías existentes para SMA.

Gracias a la clasificación propuesta y la comparativa realizada, sería muy interesante enfocar el estudio hacia la propuesta de una metodología estandarizada en el área. Igual que se ha establecido UML para el modelaje de aplicaciones orientadas a objetos, conseguir una unificación entre los investigadores y desarrolladores de SMA a nivel mundial representaría un avance muy significativo.

Por otro lado, una optimización de las capacidades de generación de código mediante la herramienta de soporte a la metodología INGENIAS permitiría aprovechar esta cualidad para reducir en mayor medida la complejidad de la implementación de nuevos SMA. En este sentido, los desarrolladores de INGENIAS están trabajando en una nueva versión de IDK, por lo que este PFC se debería actualizar a la nueva versión según lo profundos que sean los cambios. También sería interesante ver si las modificaciones mejoran tanto la calidad del código como la versatilidad para añadir nuevas características fácilmente.

7 ACRÓNIMOS

CASE: Computer Aided Software Engineering

FIPA: Foundation for Intelligent Physical Agents

GRASIAI: Grupo de Agentes Software: Ingeniería y Aplicaciones (Universidad Complutense de Madrid)

GUI: Graphical User Interface

GWAI: Grupo Web de Agentes Inteligentes (Universidad de Vigo)

HeCaSe: HEalth CAre SErvices

IDE: Integrated Development Environment

IDK: INGENIAS Development Kit

iTAKA: Intelligent Technologies on Advanced Knowledge Acquisition (Universidad Rovira I Virgili)

JADE: Java Agent Development framework

RUP: Rational Unified Process

SMA: Sistemas Multi-Agente (en inglés, **MAS** (Multi-Agent Systems))

Metodologías para SMA:

AGR: Agent-Group-Role

AUML: Agent Unified Modelling Language

E-Institutions: Institutionalized Electronic Organisations (Electronic Institutions)

MaSE: Multi-Agent systems Software Engineering

MOISE: Model of Organization for multi-agent SystEms

Opera: Organizations per Agents

PASSI: Process for Agent Societies Specification and Implementation

8 REFERENCIAS

1. Gómez-Sanz, J. (2002). *Modelado de Sistemas Multi-Agente*. Madrid: Dpto.Sistemas Informáticos y Programación, Universidad Complutense de Madrid.
2. GRASIA!, R. G. (2004). *INGENIAS Development Kit: Tutorial and Manual*. Madrid, Spain: Facultad de Informática, Universidad Complutense.
3. Henderson-Sellers B. and Giorgini P. (2005). *Agent-Oriented Methodologies*. Idea Group Publishing.
4. Isern, D. (2005). *Agent-Based Management of Clinical Guidelines*. Barcelona: Technical University of Catalonia, LSI-UPC.
5. Isern, D., & Moreno, A. (2004). *Distributed Guideline-Based Health Care System*. Budapest, Hungary: 4th International Conference on Intelligent Systems, Design and Applications, IEEE Press, pp.145-150.
6. Isern, D., Sánchez, D., & Moreno, A. (2007). *An Ontology-Driven Agent-Based Clinical Guideline Execution Engine*. Amsterdam, The Netherlands: In Proc. of AIME'07. R. Bellazzi, A. Abu-Hanna, and J. Hunter (Eds.), pp. 49-53.
7. Isern, D., Sánchez, D., Moreno, A., & Valls, A. (2003). *Personalisation of Medical Services*. In e-Health: Application of Computing Science in Medicine and Health Care. I.Rudomín, J.Vázquez-Salceda and J.L.Díaz de León Santiago (eds), pp. 18-31.
8. Mas, A. (2005). *Agentes Software y Sistemas Multi-Agente: conceptos, arquitecturas y aplicaciones*. Prentice-Hall.
9. Moreno, A., & Isern, D. (2002). *Accessing Distributed Health-Care Services Through Smart Agents*. Nancy(France): 4th International Workshop on Enterprise Networking and Computing in Health Care Industry (HEALTHCOM'02), pp. 34-41.
10. Moreno, A., Isern, D., & Sánchez, D. (2003). *Provision of Agent-Based Health Care Services*. AI Communications, Special Issue on Agents in Healthcare, 16(3), pp.135-218.
11. Moreno, A., Valls, A., Isern, D., & Sánchez, D. (2006). *Applying Agent Technology to Healthcare: The GruSMA Experience*. IEEE Intelligent Systems , vol. 21, no. 6, pp. 63-67.
12. Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*. Liverpool: John Wiley and Sons, LTD.

Páginas web de interés¹⁰:

- ◆ GRASIA!: <http://grasia.fdi.ucm.es/SP/index.php>
- ◆ GWAI: <http://gwai.ei.uvigo.es/>
- ◆ INGENIAS: <http://grasia.fdi.ucm.es/ingenias/>
- ◆ iTAKA: <http://deim.urv.cat/~itaka/>
- ◆ Metodologías de Agentes (GWAI): <http://ma.ei.uvigo.es/isoa/index.php>
- ◆ Nick Jennings webpage: <http://www.ecs.soton.ac.uk/~nrj/index.html>

¹⁰ La consulta de estas direcciones ha sido confirmada a fecha de 30/09/07.

9 ANEXO 1: NOMENCLATURA Y NOTACIÓN INGENIAS

9.1 NOMENCLATURA

Los meta-modelos de INGENIAS presentan una nomenclatura específica para identificar las relaciones de forma que se indique en el propio nombre su procedencia. Estas siglas identificativas previas se muestran en la siguiente tabla:

Relación	Siglas
Flujo de Trabajo (Work Flow)	WF + Identificador
Agente (Agent)	A + Identificador
Interacción (Interaction)	I + Identificador
Unidad de interacción (Unit of Interaction)	UI + Identificador
Objetivos y Tareas (Goal Task)	GT + Identificador
Social (AGent Operation)	AGO + Identificador
Organización (Organization)	O + Identificador
Entorno (Environment)	E + Identificador

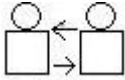
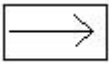
Por cada asociación entre una entidad *relationship* y un *object* ese crea una instancia con el nombre de la relación pero antecedido por la letra **R** y terminado en una letra **D** u **O**, para indicar el sentido de la relación (D de Destino, O de Origen). En caso de tratarse de relaciones n-arias, el nombre se rescribe haciendo que después del nombre de la relación aparezca un identificador significativo del uso dado al extremo concreto.

Relación-Objeto	Siglas
IdentificadorRolAsociaciónBinaria	R + NombreRelación (O D)
IdentificadorRolAsociaciónNaria	R + NombreRelaciónIdentificador (O D)

9.2 NOTACIÓN

A continuación se muestra el significado de los elementos más importantes que integran los meta-modelos INGENIAS:

Imagen	Elemento	Etiquetado	Descripción
	Agent	Nombre	Representa un agente del SMA.
	Application	Nombre y operaciones soportadas	Representa un sistema externo con el que se interactúa. No se trata de otro agente ni recurso.
	Belief	Nombre e información acerca de la suposición	Agrupar un conjunto de declaraciones que son supuestas.

	Event	Nombre y slots identificados	Indica una alteración que se ha de cumplir sobre la aplicación.
	Fact	Nombre y slots identificados	Describe la información que un agente acepta como fiable.
	Goal	Nombre	Muestra un estado deseable que el agente quiere alcanzar.
	Group	Nombre	Estructura los elementos de una organización. Contiene otros grupos, roles, agentes, aplicaciones o recursos.
	INGENIAS Use Case	Nombre, precondiciones y postcondiciones	Refina un caso de uso UML añadiendo información acerca de las precondiciones, postcondiciones e interacciones que pueden surgir.
	Interaction	Nombre y naturaleza (cooperación, planificación, negociación...)	Representa una interacción entre 2 o más agentes o roles detallando su finalidad.
	Interaction Unit	Nombre y acto del habla (request, inform or not-understand)	Refleja un mensaje de información intercambiado entre 2 agentes/roles que interaccionan.
	Mental State Manager	Nombre	Gestiona el estado mental del agente añadiendo y suprimiendo entidades mentales y garantizando que se mantenga la consistencia.
	Mental State Processor	Nombre	Representa las capacidades de decisión del agente.
	Organization	Nombre	Agrupar agentes, roles y recursos para alcanzar una o varias metas. Su estructura se refina mediante grupos.
	Resource	Nombre, cantidad disponible y límites sup. e inf. admisibles	Indica requisitos no funcionales. En el caso de que no se cumplan los límites del recurso, éste se deshabilita.
	Role	Nombre	Representa un contenedor de funcionalidades (ejecución de tareas y participación en interacciones).

	Task	Nombre	Encapsula acciones o algoritmos no distribuidos. Genera cambios sobre el estado mental del agente que la ejecuta. También puede ser asignada a roles.
	Workflow (Flujo de trabajo)	Nombre	Abstrae el proceso automatizado de actividades e identifica a sus responsables.

10 ANEXO 2: RESEARCH REPORT DEIM-RR-07-003

- Título:** “Software Engineering Methodologies to Develop Multi-Agent Systems: State-of-the-Art”
- Authors:** Cristina Gómez Alonso, David Isern Alarcón, Antonio Moreno Ribas
- Date:** August 2007
- Number:** DEIM-RR-07-003
- Language:** English
- Pages:** 46
- Abstract:** One of the most fundamental obstacles to large-scale take-up of agent technology is the lack of mature software development methodologies for agent-based systems. This drawback complicates the analysis, design and implementation of this kind of distributed applications, and, avoids the reusing of pieces of software from one project to another. Methodologies typically consist of a set of methods, models, and techniques that facilitate a systematic software development process, improving quality of the software product. An analysis and classification of existing methodologies allows multi-agent system (MAS) developers to select the most suitable approach for a specific application. Organizational ones are oriented to open-systems where cooperation is essential. Agent-based ones are oriented to close-systems and have been more developed, being more complete. To summarize, a comparison shows the advantages and drawbacks of each category and approach.
- Key-words:** Agent-based Software Methodologies, AOSE, multi-agent systems, MAS, large-scale development, life cycle development of MAS.
- Web pages:** http://deim.urv.cat/~itaka/CMS/images/pdf/report_aose.pdf

RESEARCH REPORT

Software Engineering Methodologies to Develop Multi-Agent Systems : State-of-the-art

DEIM-RR-07-003

Cristina Gómez, David Isern and Antonio Moreno

iTAKA Research Group - Intelligent Technology for Advanced
Knowledge Acquisition
Department of Computer Science and Mathematics
Universitat Rovira i Virgili
<http://deim.urv.cat/~itaka/>

Copyright © 2007 Gómez Isern Moreno / Universitat Rovira i Virgili
Permission is granted to copy, distribute and/or modify this document under the terms
of the GNU Free Documentation License, Version 1.2 or any later version published by
the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and
no Back-Cover Texts.

Software Engineering Methodologies to develop Multi-Agent Systems: State-of-the-art

Cristina Gómez, David Isern, and Antonio Moreno

iTAKA Research Group - Intelligent Tech. for Advanced Knowledge Acquisition
Department of Computer Science and Mathematics
Universitat Rovira i Virgili
43007 Tarragona, Catalonia
{cristina.gomez,david.isern,antonio.moreno}@urv.cat

Last Revision: September 3, 2007

Abstract. One of the most fundamental obstacles to large-scale take-up of agent technology is the lack of mature software development methodologies for agent-based systems. This drawback complicates the analysis, design and implementation of this kind of distributed applications, and, avoids the reusing of pieces of software from one project to another. Methodologies typically consist of a set of methods, models, and techniques that facilitate a *systematic* software development process, improving quality of the software product. An analysis and classification of existing methodologies allows multi-agent system (MAS) developers to select the most suitable approach for a specific application. Organizational ones are oriented to describe the cooperation between agents, which could be from different MAS as in open-systems. Agent-based ones are oriented to close-systems and have been more developed, being more complete. To summarise, a comparison shows the advantages and drawbacks of each category and approach.

1 Methodologies to develop Multi-Agent Systems

As other disciplines in the computer science, the use of general rules to design any system or the adoption of a general pattern to design and implement them, allow researchers to share and reuse knowledge in a easy way, and at the end, improve the quality of the product. Sophisticated distributed applications as agent-based systems requires an appropriate *methodology* to fulfil competitive and adaptative applications [40].

An agent is an *autonomous* and *proactive* system that evolves in an environment capable of interacting with other agents or the environment in order to satisfy its objectives [73]. From the software engineering point-of-view, MAS are based on open architectures that allow new agents to *dynamically* join and leave the system. Agents act *autonomously* showing *proactive* behaviours not completely predictable *a priori* [40]. In Fig. 1 are depicted all the elements related to a multi-agent system. Two main elements constitutes a MAS: the *agents* and the *environment*. The former exchange information according to some organizational patterns and rules. An agent could act alone

without interaction with any other agent, or in coalition with other agents (by cooperating or negotiating or simply in a coordinated way) in order to solve a problem from different parts. The latter is the environment where the agent lives (e.g. the Web, a data base). Agents can or cannot interact with it depending on its goals and skills.

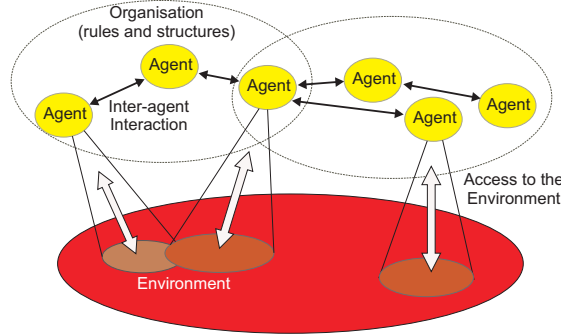


Fig. 1. Elements and interactions in a Multi-agent System ([73])

A *methodology* is a body of methods employed by a discipline and it is composed by two main components: a) a description of the process elements, and b) the work products and their documentation. In addition, it is possible to add more issues to the methodology as to allow the description of social structures, the use of tools to perform a project management, to define measuring methods to for both assess quantitatively and qualitatively evaluation, and to employ IDE or CASE tools to facilitate/represent the analysis and definition of MASs [52]. From the agent-oriented point-of-view, any agent methodology needs to contain sufficient abstractions to model and support agents and MASs, which (probably) are organised in a society of agents playing roles and exchanging information following predefined protocols [40]. Agents acts concurrently with a multiple sources of control. The intelligence can be seen as a peculiar form of control independence, and conversations as a peculiar form of interaction.

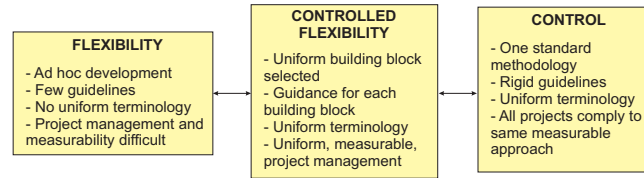


Fig. 2. Categories of methodological approaches ([40])

All methodologies followed from authors can be classified into three main categories: *flexible*, *controlled flexibility* and *controlled* (see Fig. 2). The first set of guidelines are followed by most of the developers of MAS. These applications are *ad hoc*

according the requirements of the problem, and the quality of the resulting applications is (at least) questionable because the applications are difficult to reuse and share. The next kind of methodologies is a step forward to adopt general rules of implementation. The application is divided into different parts that could be designed, managed and documented independently. The last kind of methodologies labelled as controlled allows to manage the whole project and evaluate the quality with different objective measures. Moreover, that system can be reused and shared because the terminology used is standard and all parts are well documented [40].

Basically, a methodology aims to prescribe all the elements necessary for the development of a software system, especially in the context of commercial applications. In nineties, Iivari *et. al.* [46] analysed and classified the existing information software methodologies in five main groups, and then, Henderson and Giorgini [40] added a sixth one as agent-oriented approach. In Table 1 are summarised the main features of all these methodologies.

Table 1. Information Software development methodologies (ISDM) classification (adapted from [40, 46])

<i>Methodology</i>	<i>Features</i>
Structured Approach	A step by step process at the detailed level of analysis and design activities. Separation of the essential model from the implementation model; Careful documentation to make the development process visible; Graphical notations; Top down partitionable transformation / process models to hide complexity; Unambiguous, minimally redundant graphic specification; Balancing of models; Design modules with high cohesion and weak coupling
Information Modelling	Incremental conceptual schema design; To provide an approach for enterprisewide development of information systems (databases) which enables coordinated development of integrated application systems and their long-term evolution
Sociotechnical Approach	Self-design of a work system; Minimal critical specification; Open-ended design process; Fit between the social and technical subsystems; Joint optimization; Redundant functions
Object-Oriented Approach	Seamless analysis, design and implementation; Encapsulation; Information (implementation) hiding; Object and class; Inheritance; Polymorphism; Communication between objects; Iterative and incremental development; Reuse
Soft Systems Approach	To provide a learning methodology to support debate on desirable and feasible changes; User of notional system modules called <i>human activity systems</i> to illuminate different Weltanschauungen which may be applied to any social system; An information system is a system to support the truly relevant human activity system
Agent-Oriented Approach	To design a system composed by several entities with heterogeneous beliefs, desires and intentions; Agents; Roles and protocols; Autonomous and proactive behaviours; Communication-based acts; Identify organisations and high level patterns

The closest well-known methodologies which could be applied to that domain could be the object-oriented ones, but these methodologies cannot represent the interaction between agents and its communication acts in order to achieve a common goal. For that reason, last ten years, several researchers investigated and promoted some methodologies in the agent area, but, in contrast to OO approaches, these lines of research come from the university and not from the industry. It is the most important drawback of that non mature research area.

The purpose of the MAS methodologies is propose a set of guidelines to accomplish the phases of the life cycle of any MAS application. According to RUP (Rational Unified Process [47], Fig. 3), the life cycle starts with a planning of the requirements, the analysis and design detail the performance of the system (architecture and identify all functions and procedures to execute). Then, implementation translates the design documents into a system. Finally, all the funtions and procedures are testing to guarantee its correctness. Moreover, these phases are non lineal and if any discrepancy is noticed, it is compulsory to return to past phases.

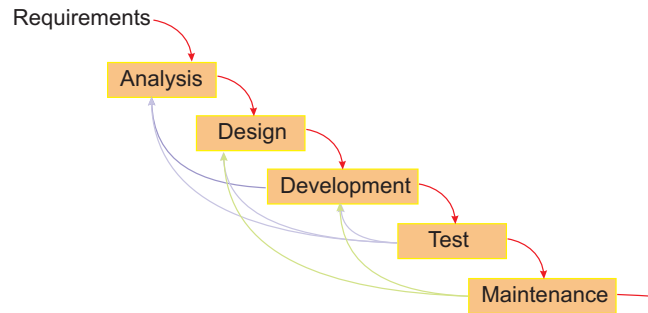


Fig. 3. Phases of software engineering

However, even if practitioners follow such methodologies during the design phase (see Fig. 3), there are difficulties to translate these specifications in a specific language, partly due to the lack of maturity in both methodologies and programming tools. Luck *et. al.* [52] updated the 2004 Garner's analysis of technologies and applications report ([29]) in order to identify and locate different agent-based applications (see Fig. 4). Concretely, MAS methodologies were located as a trigger technology just before to the peak of inflation expectations. Recently, Gartner's 2006 analysis classifies *intelligent agents* under the class *Sliding Into the Trough* ([30]), and other related research areas as *Web Agents* or *Agent-Based integration* have been disappeared from that report.

Before to analyse several approaches in the area, a classification of the existing methodologies to develop MAS is required. Nowadays, there is no a widely accepted classification of those methodologies but most of the authors are agree with two main families named *agent-oriented* and *organizational*. The whole classification is shown in Fig. 5.

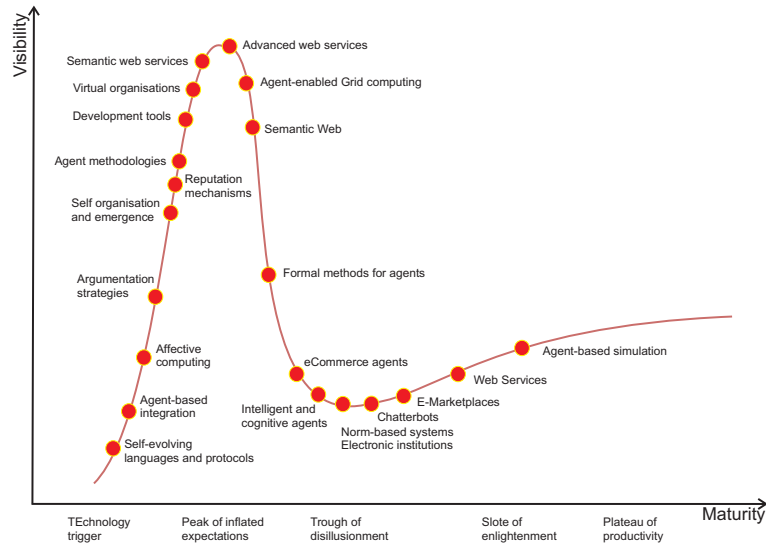


Fig. 4. AgentLink's adaptation of the technological Hype Cycle ([52])

The first classifications identified two different methodologies intended to design agent-oriented systems [74]. The former inherited the ideas of previous methodologies based on object-oriented systems. The latter, more flexible and open, adopted the ideas of knowledge engineering area and cognitive acquisition of knowledge. Recently, several authors have identified another branch that exploits the idea of agents are collections of entities that plays a social work [26, 41]. These organizational centred methodologies have been divided according if they follow a rules of behaviour or not [2].

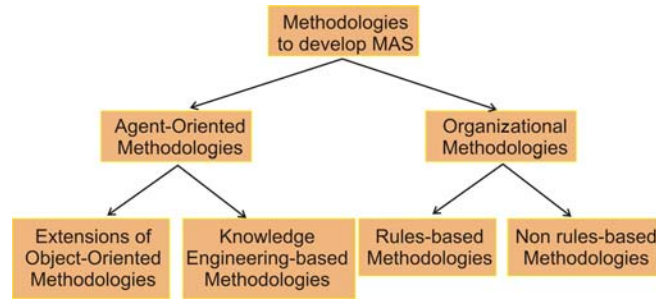


Fig. 5. Classification of methodologies to develop MAS

In the following, a deeply study of both main issues will be performed paying attention in the criteria that describe each approach.

2 Agent-Oriented Methodologies

As it was said previously, agent-oriented methodologies could be seen as a collection of rules that allow to describe the process elements and the work product and documentation. In addition, these methodologies should be able to represent the autonomous and proactive behaviour of agents.

These methodologies focus on agent-based system development distinguishing two main steps: analysis and design. A few works extend the design to the development using any general language programming or another designed to implement MAS.

2.1 Knowledge Engineering-Based Methodologies

Knowledge engineering methodologies can provide a good basis for MAS modelling since they deal with the development of knowledge based systems. Since the agents have cognitive characteristics, these methodologies can provide the techniques for modelling this agent knowledge. The definition of the knowledge of an agent can be considered as a *knowledge acquisition* process, and only this process is addressed in these methodologies.

Adapting knowledge engineering methodologies for the design of agent-based systems has certain advantages, as such methodologies provide techniques for modelling the agents' knowledge and knowledge acquisition processes. In addition, any existing tools, ontology libraries and problem-solving methods can be reused. However, such methodologies fail to address the distributed or social aspects of agents, or their reflective and goal-oriented attitudes, since a knowledge-based system is conceived as a centralised one [45].

MAS-CommonKADS MAS-COMMONKADS ([40, 45]) extends the CommonKADS methodology for knowledge-based systems by employing techniques from object-oriented methodologies as well as protocol engineering. There are three main phases. The first phase, called *Conceptualisation Phase*, deals with extracting the basic system requirements from the user. In the *Analysis Phase* a number of models are developed [68] (see Fig. 6):

- **Agent Model:** specifies the agent characteristics using use cases, problem statements, Responsibility Driving Design (RDD) and Class Responsibility Collaboration (CRC) techniques.
- **Task Model:** identifies and decomposes the tasks that the agents can carry out as in CoMoMAS functional analysis.
- **Coordination Model:** describes the agent interactions from use case scenarios. Coordination protocols are described by Event Flow Diagrams, State Transition Diagrams and Message Sequence Charts.
- **Expertise Model:** specifies different types of agent knowledge that they require in order to achieve their goals (*e.g.* domain knowledge, task knowledge, inference knowledge and problem-solving methods).
- **Organization Model:** describes MAS organization in terms of agent aggregation and inheritance.

- **Communication Model:** identifies the human-software agent interactions and the human factor for developing these user interfaces .

In the third phase, called *Design Phase*, and based on the previously developed models, a last model is created:

- **Design Model:** specifies infrastructure facilities, agent architecture, software and hardware required for MAS implementation.

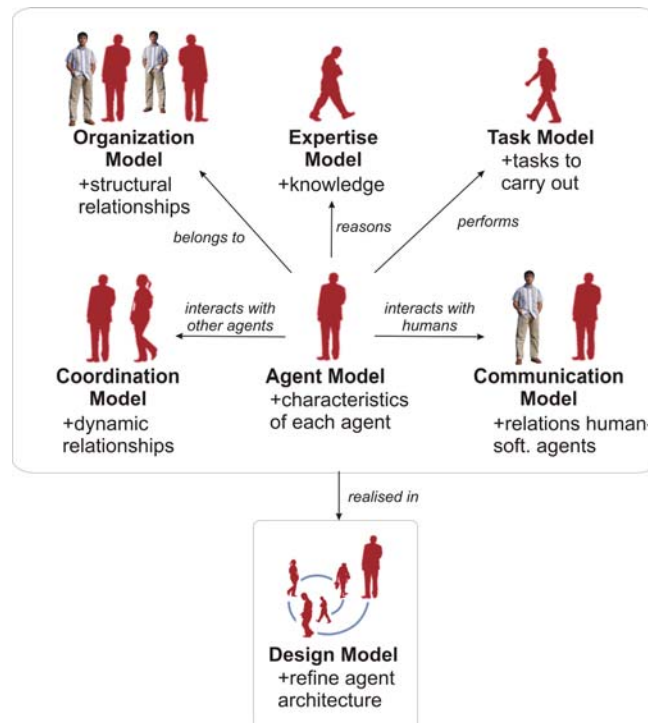


Fig. 6. MAS-COMMONKADS Models

2.2 Object Oriented Methodologies

Due the similarity between the object and agent paradigm, some researchers proposed the extension of object-oriented methodologies in the design of agent-based systems. A good examples of such methodologies are GAIA or PROMETHEUS ([40]). Agents are handled as *complex* objects with remote calls mechanisms, but proactivity or autonomy as well as, organizational patterns are difficult to design with those approaches [1, 56].

Prometheus PROMETHEUS [40, 60–62] is an start-to-end methodology that allows developers to analyse, design and implement prototypical MAS. Basically, it consists of three phases (see Fig. 7(a)):

- **System specification** identifies the basic *functionalities*, using *percepts* (inputs), *actions* (outputs) and use *case scenarios* of the target MAS.

The models defined in that phase could be a few paragraphs of descriptions and are intended to describe roughly the system. This is an iterative and non-linear process that is not necessary to be achieved in any specific sequence, but it is desirable to begin from the definition of the goals, then, the definition of the case scenarios, followed by design of the functionalities, and finally, to describe the interface of the system with the environment. A functionality encompasses a number of related goals, percepts that are relevant to it, actions that it performs, and data that it uses.

- **Architectural design** is focused on identify agents, events, interactions and shared data objects.

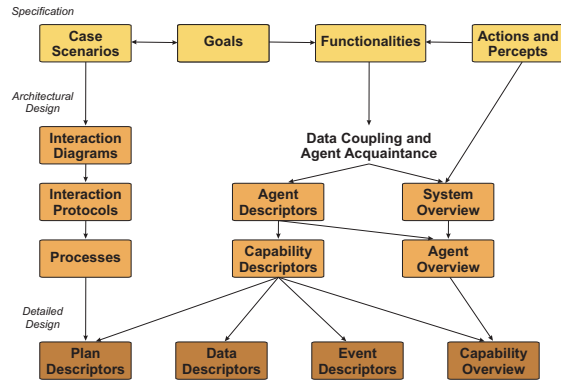
At first, this phase identifies the agent types by grouping agent functionalities through a *coupling diagram* and an *acquaintance agent* diagram. The resulting agents are described using *agent descriptors*. From the case scenarios, the interactions between agents are described in the *interaction diagrams* and the *interaction protocols*. The overall design of the system shows agents, percepts, actions, messages, and external data as nodes.

- **Detailed design** designs the internal details of each agent.

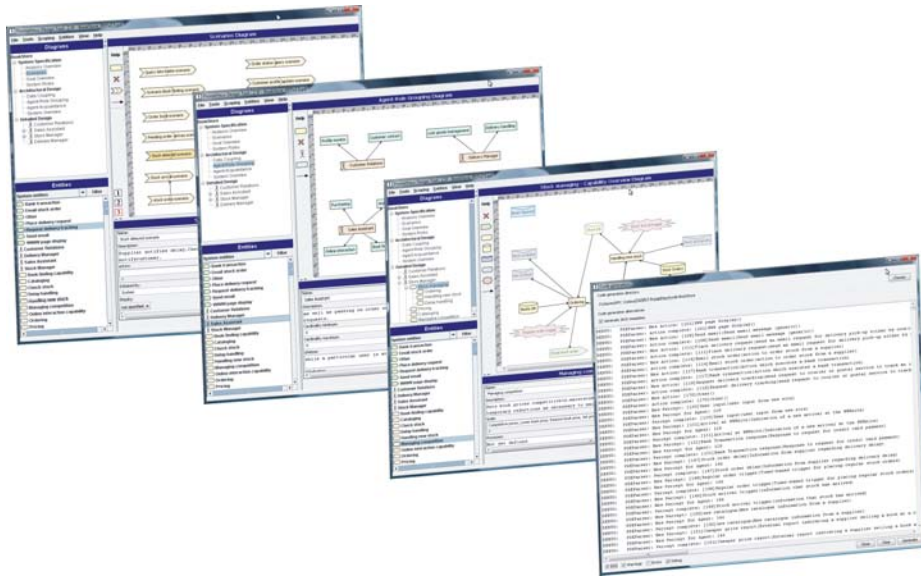
Each agent is composed of *capabilities*, which are in turn made up of lower-level capabilities, plans, internal events and data descriptors. These descriptions provide the details necessary to move into implementation. Exactly what are the appropriate details for these descriptors will depend on aspects of the implementation platform. For example if the context in which a plan type is to be used is split into two separate checks within the system being used (as is the case in JACKTM¹) then it is appropriate to specify these separately in the descriptor. Fields regarding what information an event carries assumes that events are composite objects able to carry information, and so on.

Each of those phases includes models that are intended to explain the *dynamics* of the system, graphical models to define the structure of the system and its (basic) components and textual models to provide a detailed description of all individual entities.

¹ JACKTM is an environment for building, running and integrating commercial-grade multi-agent systems using a component-based approach. JACKTM is a trademark of Agent Oriented Software Group (www.agent-software.com).



(a) The phases of the PROMETHEUS methodology [40]



(b) Screen shots of Prometheus Design Tool [62]

Fig. 7. PROMETHEUS overview

In addition, the authors of this methodology implemented a design tool called Prometheus Design Tool (PDT)² [62, 67]. That tool allows users to reuse and share different models and finally, obtain a core of prototypical agents coded using a agent-oriented language as JACK, but the models could be used in other environments. The methodology has been successfully tested in several domains such as industry, scheduling or debugging agent systems [40]. The Fig. 7(b) depicts some screen shots of the PDT under one example provided based on a electronic book store. The figure shows different views of each phase and a final window with the automatic generation of the code.

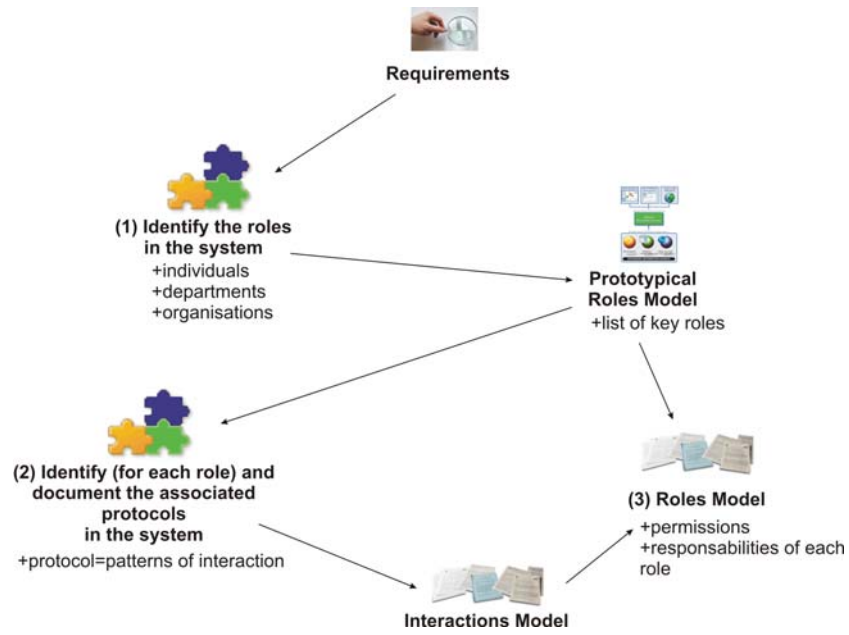
GAIA GAIA was one of the first methodologies created to assist agent-designers in the analysis and design phases [74]. The authors suggested a general framework to analyse distributed problems based on the idea of interacting roles between actors. Such complex systems implement a top-down agent identification process. First, they identify agents from roles or actors (high level analysis procedure) and their components (knowledge, behaviours, etc.). Finally, they identify interactions with other agents (low level design).

As result of applying that methodology, we have a set of models (documents) in both phases analysis and design that allows a programmer to begin the implementation.

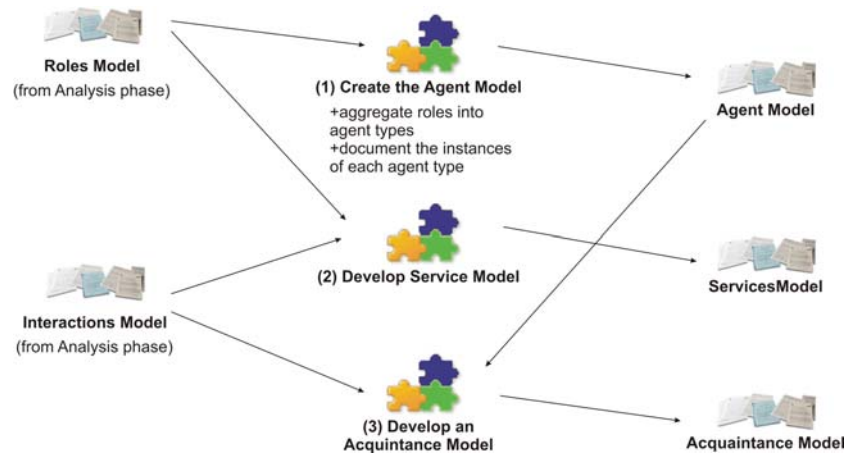
GAIA supports two phases of the development process:

- **Analysis phase:** identifies the system’s organisation (roles) and its associated protocols. Two models are created:
 - **Roles Model:** identifies what the actors should perform. Each role is defined by four attributes: responsibilities (dynamics or safety), permissions, activities and protocols.
 - **Interactions Model:** explains how the actors should interact in order to achieve the goals.
- **Design phase:** aggregates roles into several agent types, documents all instances of those agent types, identifies the services, and finally, identifies all acquaintance relationships. In Fig. 8(b) is depicted the whole design phase.
 - **Agent Model:** is defined using a simple agent type tree in which leaf nodes correspond to roles (as defined in the *roles model*), and other nodes correspond to agent types.
 - **Services Model:** is the most important document that allows to identify the services (function of the agent) associated with each agent role, and specify the main properties. Every activity (protocol that involves more than one agent) will correspond to a service, but not every service will correspond to an activity. The model identify: inputs, outputs, pre-conditions and post-conditions of each service.
 - **Acquaintance Model:** defines the communication links between agent types. It does not define what messages are sent or when messages are sent, and it is used to identify any potential communication bottleneck.

² Prometheus Design Tool (PDT) is freely available at <http://www.cs.rmit.edu.au/agents/pdt/> and recently the authors have been released the version 3.0.



(a) Step-by-step analysis procedure



(b) Step-by-step design procedure

Fig. 8. GAIA overview

Moreover, [54,55] describe a set of general rules called GAIA2JADE that summarises some recommendations or patterns to generate code agents implemented in JADE ([4]) from the GAIA models. That set of rules intend to help agent programmers to uniform and to ease the code generation. From the *agent model* the programmers have all agent types to implement. The *services model* shows all functions implemented in the system. The *acquaintance model* allows to identify the interactions between agents. Then, the most difficult documents to interpret are the *interactions* and the *roles model*. The authors propose to implement the each liveness rules, protocols and activities into some JADE behaviors (cyclic or one shot).

PASSI Process for Agent Societies Specification and Implementation, PASSI [11, 40], is a step-by-step requirement-to-code iterative methodology for designing and developing multi-agent societies integrating design models and concepts from both object-oriented (OO) software engineering and MAS, using an agent-oriented extension of the UML notation. One of the main characteristics of PASSI is that it proposes to use all the agent-oriented standards: FIPA Architecture [27], UML Modelling Language [58], and XML knowledge storage [71].

PASSI proposes five phases to its life cycle development (see Fig. 9):

- **System Requirements Model:** a model of the system requirements in terms of agency and purpose. It is composed of four phases
 - *Domain Requirements Description:* a functional description of the system using conventional use case diagrams.
 - *Agent Identification:* the phase of attribution of responsibilities to agents, represented as stereotyped UML packages.
 - *Role Identification:* a series of sequence diagrams exploring the responsibilities of each agent through role-specific scenarios.
 - *Task Specification:* specification of the capabilities of each agent with activity diagrams.
- **Agent Society Model:** a model of the social interactions and dependencies among the agents involved in the solution. Developing this model involves three steps:
 - *Ontology Description:* use of class diagrams and OCL constraints to describe the knowledge ascribed to individual agents and their communications.
 - *Role Description:* class diagrams are used to show the roles played by agents, the tasks involved, communication capabilities, and inter-agent dependencies.
 - *Protocol Description:* use of sequence diagrams to specify the grammar of each pragmatic communication protocol in terms of speech-act performatives.
- **Agent Implementation Model:** a classical model of the solution architecture in terms of classes and methods; the most important difference with the common object-oriented approach is that we have two different levels of abstraction, the social (multi-agent) level and the single-agent level. This model is composed of the following steps:
 - *Agent Structure Definition:* conventional class diagrams describe the structure of solution agent classes.

- *Agent Behaviour Description*: activity diagrams or state charts describe the behaviour of individual agents.
- **Code Model**: a model of the solution at the code level requiring the following steps to produce it:
- *Generation of code from the model using*:
 - * *PTK (PASSI ToolKit)*: that plug-in³ can export the multi-agent system model to AGENTFACTORY⁴ or generate the code for just the skeletons of the designed agents, behaviours, and other classes included in the project.
 - * *AGENTFACTORY*: it can create complex multi-agent systems by using patterns from a large repository and can also provide the design documentation of the composed agents.
 - *Manual completion of the source code*.
- **Deployment Model**: a model of the distribution of the parts of the system across hardware processing units and their migration between processing units. It involves one step:
- *Deployment Configuration*: deployment diagrams describe the allocation of agents to the available processing units and any constraints on migration and mobility.

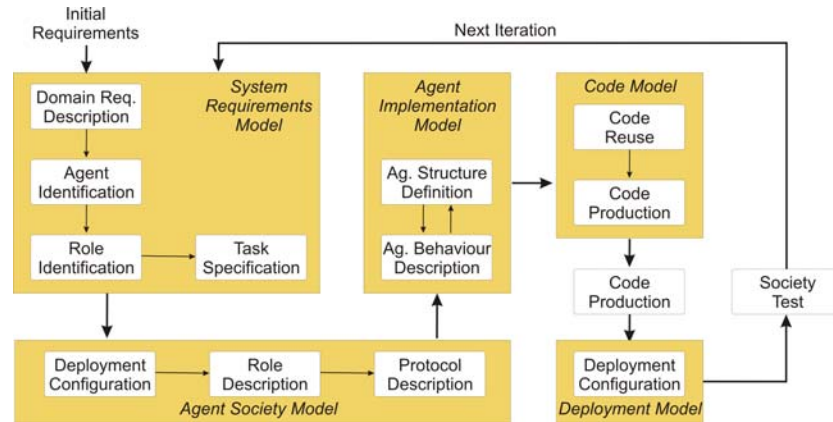


Fig. 9. Models and phases of PASSI methodology

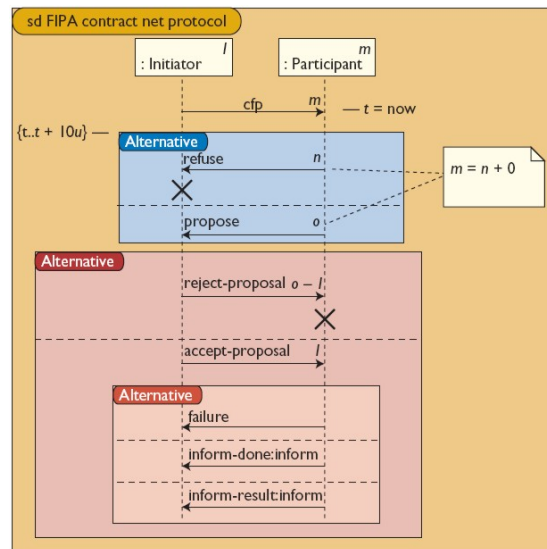
³ PASSI Toolkit is freely available at <http://sourceforge.net/projects/ptk>

⁴ AGENTFACTORY is a framework that supports a structured approach to the development and deployment of agent oriented-applications. It was developed by researchers from the School of Computer Science and Informatics at University College Dublin and it can be downloaded from <http://sourceforge.net/projects/agentfactory>.

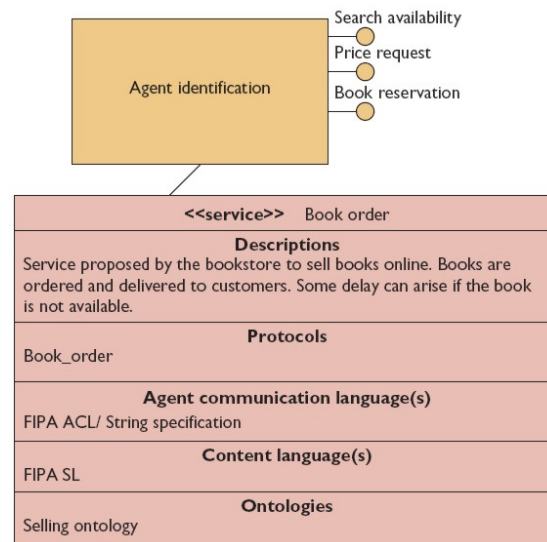
AUML Agent Unified Modelling Language (AUML) [27, 43, 44, 57] is a representation used as standard by FIPA to represent agent communication interactions and protocols. AUML is based on the object-oriented modelling representation as UML (a very recently release, updates the diagrams to UML 2.0) [58]. The main goal of AUML is to offer to developers a notation that it is used to analyse, design, and implement MAS. From the available diagrams offered by UML representation such as uses cases, packages, objects, collaboration, components, deployment, the utility for representing MAS is shown in two of those: *sequence diagrams* and *agent class diagrams*.

- **Sequence diagrams.** Sequence diagrams in MAS are diagrams which express the exchange of messages through protocols. Sequence diagrams have two dimensions: (i) the vertical dimension represents time; and (ii) the horizontal dimension represents different instances or roles. Messages in sequence diagrams are ordered according to a time axis. It is the standard de facto used by FIPA to represents all FIPA Interaction specifications [27].
- **Agent class diagrams.** Class diagram in UML shows a set of classes, interfaces, and collaborations and their relationships, and it is the most common diagram found in modelling object-oriented systems. AUML uses the same syntax but with different purposes because agents are not objects and the representation should allow to express knowledge, plans or used protocols in the agent-based system [3, 42]. The agent class diagram includes the name of the agent, a description of the internal attributes, a detailed description of all tasks implemented/allowed in each agent, a list of the offered methods, descriptions about the capabilities, services and supported protocols, information about the group where the agent is located, and an agent head automata to describe the internal behaviour.

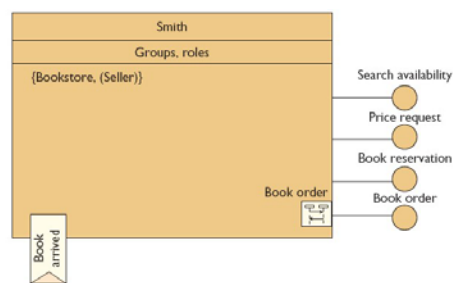
The main goal of AUML is to represent agent systems in terms of interaction. That interaction is explained through protocols and behaviours using UML-based diagrams. An evolution of AUML based on UML 2.0 offers more possibilities of expressiveness with new diagrams. The main advantage that the authors notice is that AUML is based on a well-known modelling language as UML and it could be easy to learn, but, unfortunately, it was only applied to describe FIPA interaction protocols and there are no tools to support the design.



(a) Sequence diagram of Contract Net protocol



(b) Service diagram



(c) Event diagram

3 Organizational Methodologies

A multi-agent system has two properties which seems controversial: a global purpose and autonomous agents. While the autonomy of the agents is essential for the MAS, it may cause the looseness of the global congruence. The organization of an MAS is used to solve this constraint conflict, the agents' behaviour towards global purposes.

Sichman *et al.* [64] introduced the basic principles of the organizations in agents:

- *Agents interactions may eventually create dynamic organizations* : Whenever the same interaction patterns are repeated several times, involving the same agents, these interactions may be captured by pre-established structures, thus avoiding the inherent complexity of bottom-up emergent organization formation. As a consequence, collective behaviour will be more efficient, since the organization formation is carried on a priori.
- *Agents organizations limits agents interactions, aiming to optimize the achievement of global goals*

Consequently, these dimensions of MAS make a virtuous circle: interactions build dynamic organizations, and pre-defined static organizations limit agents' interactions in order to achieve more efficiently the MAS global goals.

Ferber *et al.* [26] proposed three main principles to organizational agents:

Principle 1 The organizational level describes the “what” and not the “how”. The organizational level imposes a structure into the pattern of agent's activities, but does not describe how agents behave. In other terms, the organizational level does not contain any “code” which could be executed by agents, but provides specifications, using some kind of norms or laws, of the limits and expectations that are placed on the agents' behaviour.

Principle 2 No agent description and therefore no mental issues at the organizational level. The organizational level should not say anything about the way agents would interpret this level. Thus, reactive agents as well as intentional agents may act in an organization. In other words, ant colonies are as much organizations as human enterprises. Moreover, seen from a certain distance, or using an intentional stance it is impossible to say if the ants or the humans are intentional or reactive. Thus, the organizational level should get rid of any mental issues such as beliefs, desires, intentions, goals, etc. and provide only descriptions of expected behaviours.

Principle 3 An organization provides a way for partitioning a system, each partition (or groups) constitutes a context of interaction for agents. Thus, a group is an organizational unit in which all members are able to interact freely. Agents belonging to a group may talk to one another, using the same language. Moreover, groups establish boundaries. Whereas the structure of a group A may be known by all agents belonging to A, it is hidden to all agents that do not belong to A. Thus, groups are opaque to each other and do not assume a general standardization of agent interaction and architecture.

These principles have several consequences:

- i) An organization may be seen as a kind of dynamic framework where agents are components. Entering a group/playing a role may be seen as a plug-in process where a component is integrated into a framework.
- ii) Designing systems at the organizational level may leave implementation issues, such as the choice of building the right agent to play a specific role, left opened.
- iii) It is possible to realize true “Open System” where agent’s architecture is left unspecified.
- iv) It is possible to build secure systems using groups as “black boxes” because what happens in a group cannot be seen from agents that do not belong to that group.
- v) It is also possible to define security policies to keep undesirable agents out of a group.

Luck *et al.* [52] analysed the research area in different terms. They compared the current situation and devise some lines of future work in terms of short, medium or long term periods of time. Moreover, they classified the agent-based research into six categories such as industrial, standards, open communities, reasoning, learning and trust/reputation. Fig. 11 summarises all this data. The most important conclusions are:

- *Short-term future.* Participating agents have fewer goals in common, although their interactions will still concern a common domain, and the agents will be designed by the same team, and will share common domain knowledge. Increasingly, standard agent communication languages, will be used, but interaction protocols will be mixed between standard and non-standard ones.
- *Medium-term future.* MAS will permit participation by heterogeneous agents, designed by different designers or teams. Any agent will be able to participate in these systems. However, these open systems will typically be specific to particular application domains. System development will proceed by standard agent-specific methodologies. Agent-specific programming languages and tools will be increasingly used.
- *Long-term future.* The development of open MAS will span multiple application domains, and involve heterogeneous participants developed by diverse design teams. Agents will be able to learn the appropriate behaviour for participation in the course of interacting. Selection of communications protocols and mechanisms, and of participant strategies, will be undertaken automatically. Agents will be able to form, manage and dissolve dynamic coalitions.

	Short Term	Medium Term	Long Term
Industrial Strength software	Peer to peer Better development tools Agent UML Service oriented computing	Generic designs for coordination Libraries for agent-oriented development	Best practice in agent systems design
Agreed Standards	FIPA ACL Peer to peer Better development tools Service oriented computing Semantic description	Flexible business/trading languages Libraries of interaction protocols	Tools for evolutions of communications languages and protocols
Infrastructure for Open Communities	Web mining Data integration and Semantic Web Metadata	Semantic interaction Agent-enabled semantic web (services) Electronic Institutions Dynamic norms, roles, laws, organisations	Share, improved ontologies
Reasoning in Open Environments	Organisational views of agent systems	Enhanced understanding of agent society dynamics Theory and practice of argumentation strategies Norms and social structure Theory and practice of negotiation strategies	Automated eScience systems and other application domains
Learning Technologies	Adaptation Personalisation Hybrid technologies	Evolving Agents Self organisation Distributed learning	Run-time reconfiguration and re-design
Trust and Reputation	Security and verifiability for agents Reliability testing for agents Self-enforcing protocols	Norms and social structures Reputation mechanisms Formal methods for open agent systems Electronic contracts	Trust techniques for coping with malicious agents

Fig. 11. Agent technology comprises areas that will be addressed over different timescales

New orientation of the agent-oriented methodologies towards organizations is the main objective to medium-term future, whereas it will be the open systems to to long-term future. According to [38], this change requires the methodology employed to have two characteristics:

- a) *Architectural Independence* as the requirements of the system are dynamic, and the potential scope and complexity of the system vary through its lifetime, the methodology should not subscribe to any particular programming paradigm, programming language, agent architecture or implementation framework. Instead, different paradigms and architectures must be accommodated naturally.
- b) *Robustness and Scalability*. The use of methodologies must facilitate dynamic changes without compromising the system robustness. Scalability should be promoted to facilitate the growth of the open system to any given complexity. To achieve this, the methodology must support the modelling of complex and often dynamically formed social interactions between agents within the system.

3.1 Organizational Paradigms

Horling and Lesser [41] established that the major organizational paradigms used in multi-agent systems are:

Hierarchies Agents are conceptually in a tree-like structure, where agents higher in the tree have a more global view than those below them. More complex instances have multiple levels. Fox propose three types of hierarchical organizations: simple, uniform and multi-divisional. This approach is similar to divide-and-conquer with high efficiency. Its drawbacks are: fragility (prone to bottlenecks or single-point failures with potentially global consequences) and slowness in very tall structures (delays incurred by passing information across multiple levels) (Fig. 12(a)).

Holarchies Physical systems based on the nested and self-similar organization. Each grouping in these systems has a character derived but distinct from the entities that are members of the group. Each holon is composed of one or more subordinate entities, and can be a member of one or more superordinate holons. Many holonic structures also support connections between holons across the organization. Applied to domains where goals can be recursively decomposed into subtasks that can be assigned to individual holons. The challenge in creating a holonic organization revolves around selecting the appropriate agents to reside in the individual holons (Fig. 12(b)).

Coalitions Subset of agents formed by a purpose and dissolve when that need no longer exists. Related research has extended to longer-term agreements based on trust. Its organizational structure is typically flat and once formed, coalitions may be treated as an atomic entity. The agents in this group are expected to coordinate their activities in a manner appropriate to the coalition's purpose. Its main advantage is the ability to solve goals with requirements greater than any single agent can offer. Its drawback is the additional complexity incurred if the partitioning of agents is not disjoint (Fig. 12(c)).

Teams A number of cooperative agents which have agreed to work together towards a common goal. In comparison to coalitions, teams attempt to maximize the utility of the team (goal) itself. Its benefits are: the solution of larger problems, the ability to meet global constraints and economies of scale can also be realized. The drawback is increased communication. The challenges associated with team formation involve other three problems: determining how agents will be allocated to address the high-level problem, maintaining consistency among agents during execution and revising the team as the environment or agent population changes (Fig. 12(d)).

Congregations Groups of agents who have banded together into a typically flat organization in order to derive additional benefits. Unlike these other paradigms, congregations are assumed to be long-lived and are not formed by a single specific goal. They are formed among agents with similar or complementary characteristics to facilitate the process of finding suitable collaborators. Communication does not occur between agents in different congregations, although the groups are not

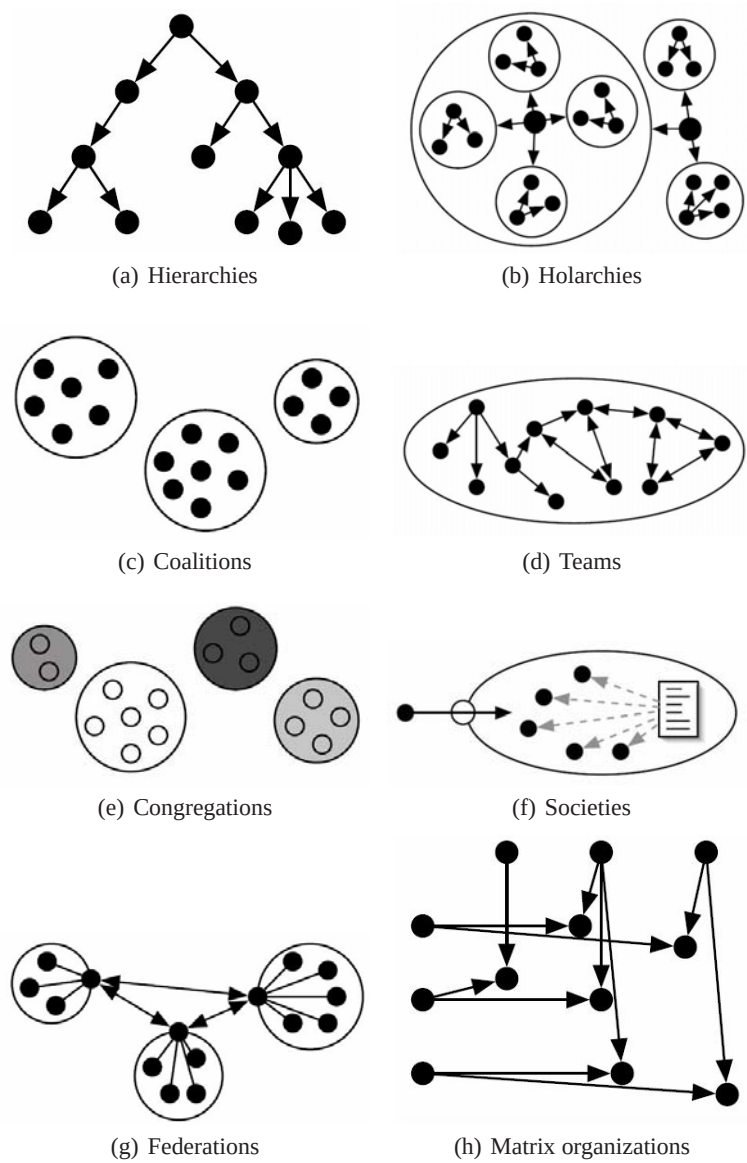


Fig. 12. Organizational paradigms

necessarily disjoint. It is necessary to facilitate the discovery of agent partners by restricting the size of the population that must be searched. The downside to this strategy is that the limited set may be overly restrictive, and not contain the optimal agents one might select given infinite resources. Congregation formation involves selecting or creating an appropriate group to join because they are more ideologically or capability driven, and there is usually no specific goal or task to unite them, one must first define how these groups may be differentiated (Fig. 12(e)).

Societies Agents will have different goals, varied levels of rationality, and heterogeneous capabilities; the societal construct provides a common domain through which they can act and communicate. A set of constraints (social laws or norms) is imposed on the behaviour of the agents. These are guidelines by which agents must act, which provides a level of consistency of behaviour and interface intended to facilitate coexistence. They must minimize conflicts and encourage efficient solutions. Indeed, most of research on agent societies is more concerned with how the concepts they embody can be used to facilitate the construction of large-scale, open agent systems in general. There are two aspects to the society formation problem. The first is to define the roles, protocols and social laws which form the foundation of the society. Given such a definition, the second problem is to implement the more literal formation of the society; that is, determining how agents may join and leave it (Fig. 12(f)).

Federations Group of agents which have ceded some amount of autonomy to a single delegate which represents the group (facilitator, mediator or broker). Group members interact only with this agent, which acts as an intermediary between the group and the outside world. This arrangement particularly useful for integrating legacy or an otherwise heterogeneous group of agents. As part of the allocation, the broker may decompose the problem into more manageable subtasks. This allows agents to take advantage of all the capabilities of the federation, without requiring knowledge of which agents perform a task or how they go about doing it. They are dynamically created in response to new task arrivals or requests from other groups using a contract net approach, or are statically created from agents in a common subsystem (e.g. tools, workers, etc.) (Fig. 12(g)).

Matrix Organizations Agents whose interrelationships can come from many directions, each with its own objectives, relative importance and pertinent characteristics. Relax the one-agent/one-manager restriction, by permitting many managers or peers to influence the activities of an agent. The matrix provides a graphical way to depict which managers can influence the activities of each agent (Fig. 12(h)).

3.2 Classification

Inside the *Organizational Methodologies*, there are two types of methodologies depending on the concept of organization:

- a) methodologies which only consider an organizational structure and propose common goals for the set of agents, and
- b) methodologies which, moreover, explicitly present *rules of behaviour* among agents, called **norms**.

There are different definitions of term *norm*. According to sociology,

a norm is a rule or standard of behaviour shared by members of a social group (Encyclopaedia Britannica).

According to philosophy,

a norm is an authoritative rule or standard by which something is judged and on that basis approved or disapproved (Columbia Encyclopaedia) [21].

According to agents,

norms are rules which express relationships and constraints between roles, between protocols, and between roles and protocols, that can drive the identification of the organizational structure [75].

In other words, norms define the *rights* and *obligations* of the agents related to the roles they play and their environment or area of activity. To some extent, organizational rules can be considered as liveness and the safety properties of the organization. Therefore, it is essential to consider *norms* to describe the rules of behaviour for the agents within the organization [14], although some methodologies do not allow these qualities.

3.3 Non-Rules-Based Organizational Methodologies

In the following, we shall analyse some methodologies that are considered *non-rules-based*. Examples of such methodologies are AGR , MASE and TROPOS .

AGR Agent-Group-Role (AGR) [26] is the evolution of the AALAADIN model. The AGR model is based on three main topics:

- **Agent.** An agent is an active, communicating entity playing roles within groups. An agent may hold multiple roles, and may be member of several groups. No constraints are placed upon the architecture of an agent or about its mental capabilities.
- **Group.** A group is a set of agents sharing some common characteristic. A group is used as a context for a pattern of activities, and is used for partitioning organizations. Two agents may communicate if and only if they belong to the same group, but an agent may belong to several groups. This feature will allow the definition of organizational structures.
- **Role.** The role is the abstract representation of a functional position of an agent in a group. An agent must play a role in a group, but an agent may play several roles. Roles are local to groups, and a role must be requested by an agent. A role may be played by several agents.

AGR propose three stages in its methodology:

- 1) Identify the main *groups* of the application (a set of similar agents or a function based system)
- 2) Build the overall *organizational structure*. Two kinds of diagrams are used:
 - *CheeseBoard Diagram* as a first idea of the organizational patterns (see Fig. 13). A *group* is represented as an oval, *agents* are represented as skittles that stands on the board and sometimes go through the board when they belong to several groups and a *role* is represented as a hexagon and a line links this hexagon to agents).

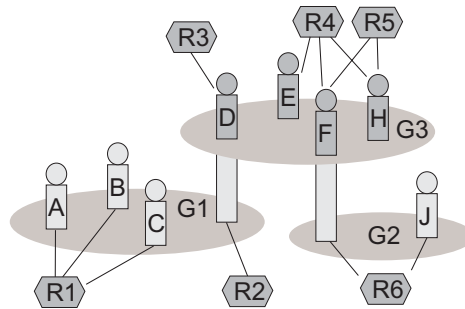


Fig. 13. AGR CheeseBoard notation

- *Organizational Structure Diagram* represents the static aspects of agents. Also, hierarchies and divisions.
- 3) Build the *organizational dynamic* of group creation and adhesion with *Organizational Sequence Diagrams* which get into the definition of roles in a functional way (variant of sequence diagrams (oriented to organizations)).

An AGR organizational structure for an overall process (or organisation) is a specification based on a definition of *groups*, *roles* and their relationships. An organisation as a whole is composed of a number of groups. A group structure identifies the roles and the *intragroup transfers* between roles. In addition, *intergroup role interactions* between roles of different groups specify the connectivity of groups within an organisation. *Agents* are allocated to roles; they realise the organisation. However, the aim of an organisation model is to abstract from any specific agent allocated. Therefore instead of particular agents, roles are used as abstract entities, defining properties agents should have when they are to function in a given role within an organisation.

The main characteristic of the AGR model is its *minimalist* structure-based view of organizations. In an AGR model an organization is specified as a role-group structure imposed on the agents. AGR also says that agents can have their joint behaviour orchestrated by interaction protocols, but the nature and the primitives to describe such protocols are left open [12]. Therefore, authors propose to use GAIA methodology to fill the roles and relate them to the general structure and MADKITplatform as a toolkit for practical implementations. In spite of these tools, the first phase results too difficult and a deepening in the initial requirements would be recommendable to identify easily the groups.

MaSE Multi-agent systems Software Engineering, MASE [72, 16, 17, 19], takes an initial system specification and produces a set of formal design documents in a graphically based style. The primary focus of MASE is to guide a designer through the software life cycle from a prose specification to an implemented agent system. MASE is independent of a particular multi agent system architecture, agent architecture, programming language, or message-passing system.

MASE is an iterative methodology across all phases with the intent that successive models add detail to the previous ones. Based on RUP [47], its development process phases are focus on (see Fig. 14(a)):

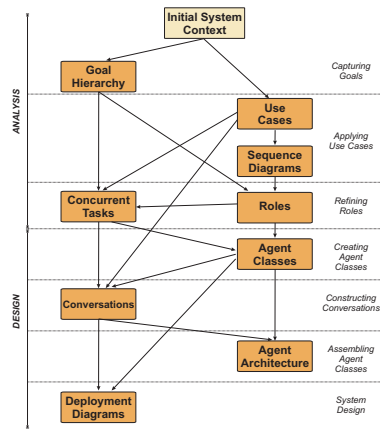
- **Analysis:** produces a set of roles whose tasks has to cover the initial system requirements:
 - *Capturing goals:* identifies the set of system goals and structures them in a *Goal Hierarchy Diagram* by importance.
 - *Applying Use Cases:* captures use cases from the initial system requirements and restructures them as a *Sequence Diagram*. At least, one sequence diagram is created from each use case. If there are several possible scenarios, multiple diagrams are created.
 - *Refining Roles:* transforms the structured goals of the *Goal Hierarchy Diagram* into roles, which define agent's classes and capture system goals during the design phase. The general case transformation is one-to-one; each goal maps to a role. Role definitions are captured in a traditional *Role Model*.

- **Design** : defines the overall system organization by transforming the roles and tasks into agent types and conversations:
 - *Creating Agent Classes*: identifies agent classes from component roles. The product of this phase is an *Agent Class Diagram* which depicts agent classes and the conversations between them. Mapping goals to roles is generally one-to-one between roles and agent classes, but it is not compulsory.
 - *Constructing Conversations*: defines a coordination protocol between two agents. A conversation consists of two *Communication Class Diagrams*, one each from the initiator and responder and it must support and be consistent with all sequence diagrams derived earlier.
 - *Assembling Agent Classes*: creates the internal of agent classes.
 - *System Design*: takes the agent classes and instantiates them as actual agents. It uses a *Deployment Diagram* to show the numbers, types and locations of agents within a system.

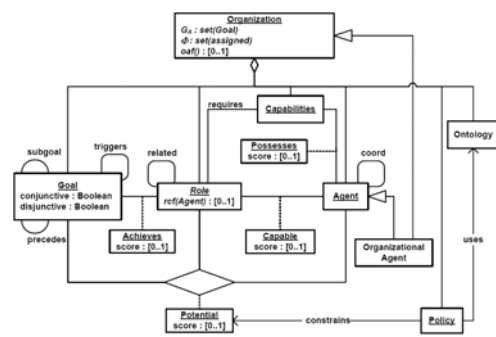
A final element to consider is the automatic code generation. In this issue, the authors developed a CASE tool called AGENTTOOL⁵ (see Fig. 14(c)) that supports the entire life cycle and also verifies the correctness of the agent protocols. Furthermore, it eases MASE deployment and learning.

The first version of MASE ([19]) was an object-oriented methodology, but then, the same authors fixed several weakness and added some improvements in order to add organizational structures detailed in [18]. That extended version of MASE is called O-MASE (Organizational MASE) (Fig. 14(b)). In the *Requirements phase*, the systems goals are represented in a AND/OR goal tree. After, in the *Analysis phase*, an Organizational Model is created, which defines the organization's interactions with external actors, goals and services. Activity and sequence diagrams refine it. Each organization can include sub-organizations to allow abstraction during the design process. Next, a Role Model detail the Organizational Model with protocols and capabilities and the Domain Model presents the ontology. O-MASE does not require to create concurrent task diagrams. In the *High-Level Design phase*, the Agent Class Model and Protocols Models are defined. Finally, in the *Low-Level Design phase*, agent behaviour is detailed using Agent State Models, which are based on finite state automata.

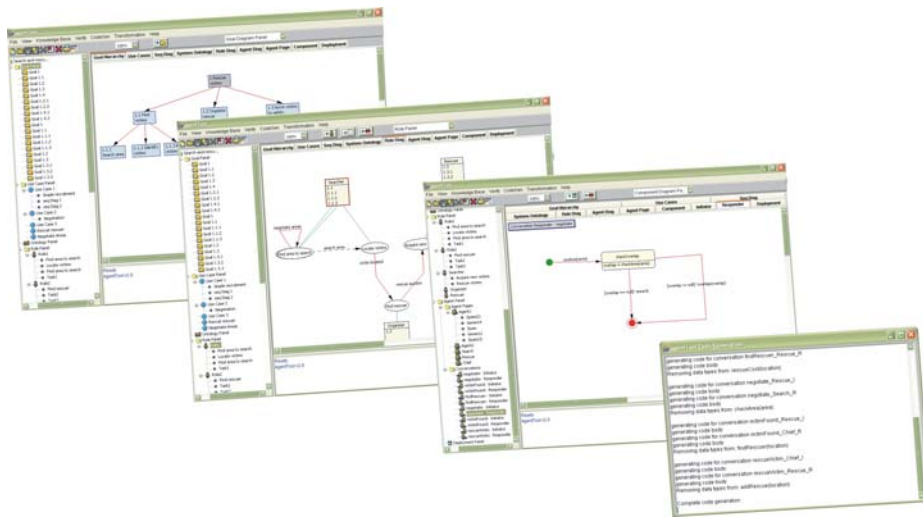
⁵ AGENTTOOL is freely available at <http://macr.cis.ksu.edu/projects/agentTool/agenttool.htm>.



(a) MASE development life-cycle



(b) MASE organizational structures



(c) Screen shots of AGENTTOOL

Fig. 14. MASE overview

TROPOS TROPOS methodology ([6, 32]) is based on two key ideas. First, the notion of agent and all related mentalistic notions (for instance goals and plans) are used in all phases of software development, from early analysis down to the actual implementation. Second, TROPOS covers also the very early phases of requirements analysis, thus allowing for a deeper understanding of the environment where the software must operate, and of the kind of interactions that should occur between software and human agents (founded on *belief*, *desire*, and *intention* (BDI) agent architectures ([31])).

It is based on the *i** model which offers actors, goals and actor dependencies as primitive concepts and for its modelling activities (actor, goal, plans and capabilities), UML ([58]) and AUML (see §2.2) diagrams are used.

The four main development phases of the TROPOS methodology are:

- **Requirements analysis:** is the most important phase. It is split in two main phases which share the same conceptual and methodological approach:
 - *Early Requirements:* the requirements engineer identifies the domain stakeholders and models them as social actors, who depend on one another for goals to be achieved, plans to be performed, and resources to be furnished.
 - *Late Requirements:* the conceptual model is extended including a new actor, which represents the system, and a number of dependencies with other actors of the environment. These dependencies define all the functional and non-functional requirements of the system-to-be.
- **Architectural Design:** defines the system’s global architecture in terms of sub-systems (actors), interconnected through data and control flows (dependencies). It provides also a mapping of the system actors to a set of software agents, each characterized by specific capabilities. In [50], a classification of *Organizational Styles* is proposed to Tropos according two disciplines:
 - *Organization Theory:* describes the structure and design of an organization. For instance: structure-in-5, pyramid style, chain of values, matrix, bidding style.
 - *Strategic Alliances:* model the strategic collaborations of independent organizational stakeholders who have agreed to pursue a set of agreed upon business goals. Some styles are: joint venture, arm’s length, hierarchical contracting, co-optation.

These authors also recommend to incorporate into system architecture some patterns at a micro level like: broker, mediator, wrapper or embassy.

- **Detailed Design:** aims at specifying agent capabilities and interactions.
- **Implementation:** establishes mapping between the implementation platform constructs and the detailed design notions. It provides a detailed implementation of organizational models into JACK agents (agent-oriented extension of Java).

The main two shortcomings [22] of TROPOS are that: *a*) it is **not formal** (although there is some ongoing work on providing a formal semantics for TROPOS), and *b*) it is **too organizational-centred** in the sense that it does not consider that agents can have their own goals and plans, and not just those coming from the organization. Furthermore, TROPOS has no concept representing the normative aspects of an organization. In [53], the *Theory of Commitment Protocols* is applied to TROPOS improving the detection of interactions and dependencies in early phases.

3.4 Rules-Based Methodologies

As it was introduced in §3.2, that kind of methodologies defines a general behaviour as *norms* that affects, agents and their interactions. Examples of such methodologies are E-INSTITUTIONS, EXTENDED GAIA, INGENIAS, OPERA, or MOISE, which it will be analysed in the following.

Electronic Institutions Institutionalized Electronic Organisations (electronic institutions or E-INSTITUTIONS, for shorter) [24] provide a computational analogue of human organizations, oriented to open-systems MAS development. It is populated by a vast amount of heterogeneous (human and software) agents playing different roles and interacting by means of speech acts.

To design such systems requires a theory of organization design, and knowledge of how organizations may change and evolve over the time. Sociological organization theory and social psychology are also important inputs to the design. Moreover, to design open multi-agent systems, political theory may be necessary. Usually, agents may unknown its objectives at the time of design the system [51].

One of the main purposes of e-institutions will be to model the creation, evolution, and consequences of the rules defining institutions and their incorporation into agent organisations. However, it does not consider any explicit topology for the system and only takes into account several hierarchical role relationships (*e.g.* role subsuming functions)[2].

Inside an formal specification of an E-INSTITUTIONS infrastructure is essential to point out the following core notions:

- **Agents:** are the players in an electronic institution, interacting by the exchange of speech acts
- **Roles:** are defined as standardized patterns of behaviour. They can be updated without having to update the actions for every agent on an individual basis.
- **Dialogic framework:** composed of a communication language, a representation language for domain content and an ontology.
- **Scene:** agent group meetings between agents (interactions) with a well-defined communication protocol. No agent interaction within an institution takes place out of the context of a scene. Scene protocols are patterns of multi-role conversation. A scene protocol is specified by a graph where the nodes of the graph represent the different states of the conversation and the arcs connecting the nodes are labelled with illocution schemes that make scene state change. The graph has a single initial state and a set of final states representing the different endings of the conversation.
- **Performative structure:** models the relationships among scenes. It contains a description of how the different roles can legally move from scene to scene. Agents within a performative structure may be possibly participating in different scenes, with different roles, and at the same time.
- **Normative Rules:** obligations to the agents which affect over its possible paths within the performative structure.

The same authors presented two years later, ISLANDER a declarative language for specifying E-INSTITUTIONS [23]. Moreover, in 2004, they proposed AMELI as an agent-based middleware [25].

A comparative of ISLANDER against MOISE+ and AGR can be found in [12] where the main difference is that ISLANDER is focused on an institutional aspects of organizations(normative and dialogical).

Extended GAIA The GAIA methodology explained in §2.2 was revisited three years later by the same authors in order to represent a MAS as an organized *society* of individuals in which each agent plays specific *roles* (or responsibilities) and interacts with other agents according to *protocols* determined by the roles of the involved agents [7, 40, 76]. With that approach, the overall system behaviour is understood in terms of both micro and macro level. The first explains how the agents acts according its roles, and the latter explains the control over those agents.

That revision of GAIA is based on the idea of *human organization* in which the actors which play different roles, can interact with other partners. Interactions between agents are identified and localised in the definition of the role itself, and they help characterize the structure of the organisation and the position of the agent in it [75].

The analysis phase documents all the functional characteristics that the system should express, together with the characteristics of the operational environment in which the MAS is situated (see Fig. 15(a)). The leitmotiv of this phase is the determination of organizations as groups of individuals that exhibits a common behaviour, or that interact loosely with other portions of the system, or that require competencies that are not needed in other parts of the system.

A new model named *environmental model*, documents explicitly the characteristics of the environment in which the MAS is located. First version of GAIA embedded this information into the *roles model*.

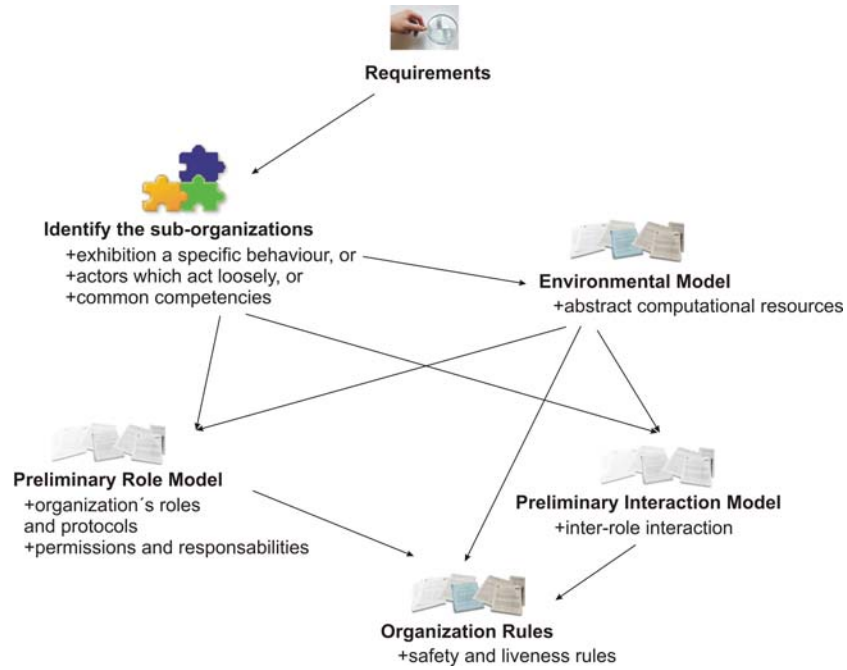
Two *preliminary* documents are also outputs of this phase: *role* and *interaction* models. The *preliminary role model* is intended to identify some characteristics of the system that are likely to remain the same independently of the desired organizational structure followed. And the *preliminary interaction model* is intended to distinguish the dependencies and relationships between roles named protocols.

At the end of the analysis phase, the system designer should identify, from roles and interactions, the basic functionalities and relationships pattern to be deployed among the MAS, in fact, organizations.

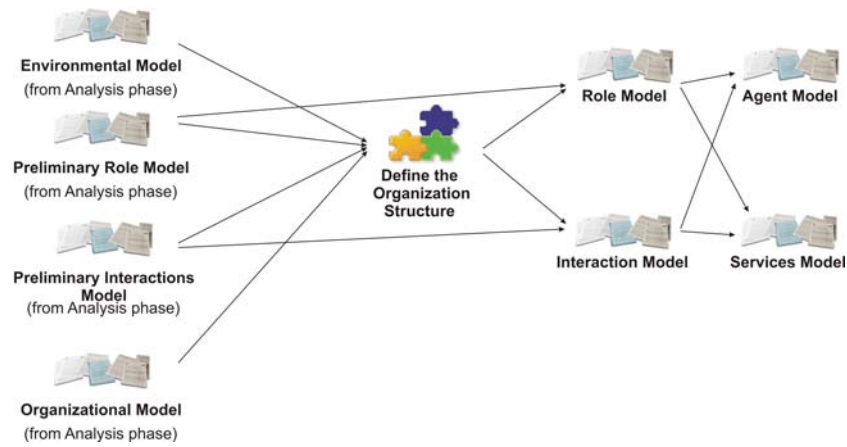
The design stage takes both models and preliminary models from the previous phase in order to define an reliable architecture of agents and complete the unfinished models (see Fig. 15(b)).

First of all, the agent designer should identify the structure of the organization to be implemented. GAIA is independent from the final implementation selected an for this reason it gives some general rules to define that organisation. To tackle this issue could be useful to use on the organisations summarised in §3.1 according to the system requirements. As the first version of GAIA , the *roles model* identifies what the actors should perform, and the *interactions model* explains how the actors should interact in order to achieve the goals, it means, that is an operational description of the MAS organization.

When these models are defined, only agent and services models remain to be defined. *Agent model* defines for each agent which roles should implement, and if it is



(a) Step-by-step analysis procedure



(b) Step-by-step design procedure

Fig. 15. GAIA overview

possible, the number of instances of each that the MAS should support. Finally, *services model* documents the services (derived from the list of protocols, responsibilities, activities, and roles) that may be performed by an agent. The agent designer may identify inputs, outputs, pre-conditions, and post-conditions but does not prescribe details of implementation.

INGENIAS INGENIAS [34, 40] is an improvement of MESSAGE (Methodology for Engineering Systems of Software Agents) where relationships among models and activities identification have been refined to generate MAS specifications, and it incorporates new support tools and development examples.

INGENIAS, based on MESSAGE, present a set of meta-models to analyse all the system using GOPRR language (Graph, Object, Property, Role and Relationship) [49]. These viewpoints are:

- **Organization:** framework where agents, resources, tasks and goals coexist. It is defined by its structure, functionality and social relationships. It proposes a decomposition of the MAS in groups and workflows.
- **Agent:** functionality of each agent: purpose, responsibilities and capabilities. The behaviour consists of: its mental state (information that allows the agent to take decisions), its mental state manager (operations over elements of its mental state and their relationships) and its mental state processor (selection of a decision to execute a task)
- **Tasks/Goals:** decomposition of goals and tasks, the consequences of performing a task and why it should be performed.
- **Interaction:** exchange of information or requests between agents (or agents and human users). Its elements are: actors, interaction specification, context and nature of the interaction.
- **Environment:** entities with which the MAS interacts (resources, other agents or applications).

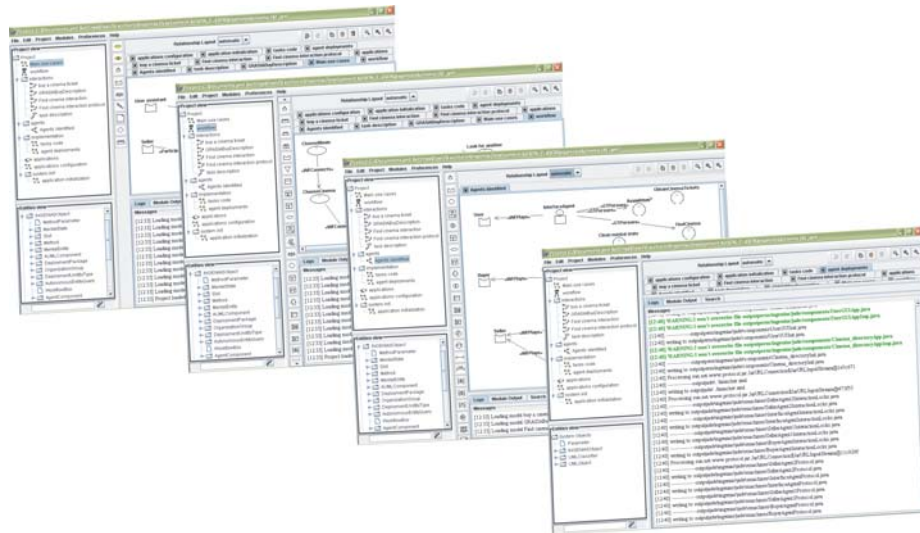
All the models are developed following the steps of the RUP [47] as are depicted in Fig. 16(a).

The authors suggest to start with an organization model, a UML use case model, and an environment model. However, developers have the freedom to select others, taking into account their backgrounds. Following that guideline, developers should determine the kind of product to generate, the phase involved and the work flow stage checking the level of abstraction. INGENIAS executes its activities within the RUP. Therefore, it may be necessary to iterate over different phases until the system is finalized. The process ends when there is a functional system that satisfies user requirements. These are identified with use cases at the beginning.

As regards social norms, they are not explicitly modelled, although they are implicitly in the organizational viewpoint. Also, organizational dynamics are not considered *i.e.* how agents can join or leave the system, how they can form groups dynamically, what their life cycle is, etc.) [2].

		PHASES		
		Inception	Elaboration	Construction
WORKFLOWS	ANALYSIS	- Generate use cases and identify their actors - Sketch a system architecture with an organization model. - Generate environment models to represent results from requirement gathering stage	- Refined use cases and interactions associated to them - Agent models that detail elements of the system architecture. - Workflows and tasks in organization models - Models of tasks and goals to highlight control constraints (main goals, goal decomposition) - Refinements of environment model to include new environment elements	Refinements on existing models to cover use cases
	DESIGN	Generate prototypes perhaps with rapid application development tool such as ZEUS or Agent Tool or code generators from the IDK. Sometimes this prototyping limits to concrete aspects and uses a more conventional technology	- Refinements in workflows - Interaction models that show how tasks are executed. - Models of tasks and goals that reflect dependencies and needs identified in workflows and how system goals are achieved - Agent models to show required mental state patterns	- Generate new models - Social relationships that regulate organizational behavior

(a) Results of analysis and design phases of the INGENIAS development process



(b) Interface of the IDK (Ingenias Development Kit)

Fig. 16. INGENIAS overview

Authors develop an agent-oriented software tool called Ingenias Development Kit (IDK)⁶ (Fig. 16(b)). It allows to edit consistent models (according INGENIAS specification) and generate code with documentation to different platforms such as JADE, Robocode, Servlets or Grasia! Agents.

The main pitfall of INGENIAS is that oriented to too big developments and it can not be reduced to smaller ones. In [33], a recursive application of INGENIAS is presented to manage large-scale MAS.

OperA Organizations per Agents, OPERA [20], is a methodology tailored to OPERA model, but describes generic facilitation and interaction frameworks for agent societies that implement the functionality derived from the coordination model applicable to the problem domain.

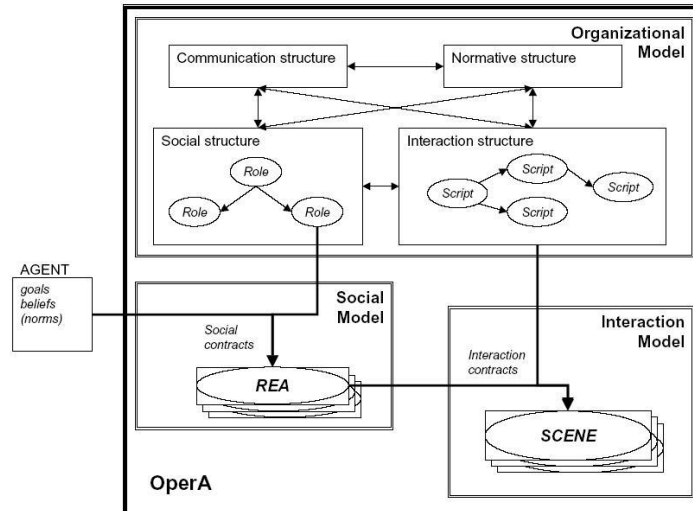
The three components of an OPERA Model are (see Fig. 17(b)):

- **Organizational (OM)**: specifies the organizational characteristics of an agent society in terms of four structures:
 - *Social*: objectives of the society, its roles and what kind of model governs coordination.
 - *Interaction*: interaction moments of a society task that requires the coordinated action of several roles.
 - *Normative*: society norms and regulations, expressed in terms of role and interaction norms.
 - *Communicative*: ontologies for description of domain concepts and communication illocutions.
- **Social (SM)**: fixes the enactment of roles by agents in social contracts that describe the capabilities and responsibilities of the agent within the society. A social contract defines a *role-enacting agent* (REA). An *agent population* is a set of agents enacting society roles at a given moment. The use of contracts allows to link the different models and create specific instances that reflect the needs and structure of the current environment and participants, and on the other hand, verificate the outcome of the system.
- **Interaction (IM)**: describes the concrete interaction scenes between agents based on the interactions scripts specified in the OM. Every interaction commitments are fixed in interaction contracts.

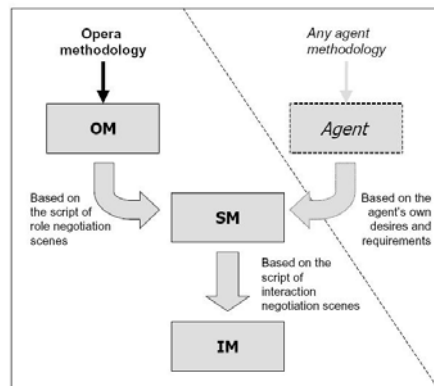
According to the characteristics of the exchange context and the types of goods or services exchanged, three organizational structures are presented in OPERA: market, hierarchy or network. Depending on the coordination model, the design of the agent society will be different. Therefore, it is recommended to identify as preliminary step in the development of the agent society which it will be. In the scope of open society systems, OPERA authors consider that its methodology supports those models because it fulfils the following requirements (see Fig. 17(a)):

- *Internal autonomy requirement*: interaction and structure of the society must be represented independently from the internal design of participating entities.

⁶ Ingenias Development Kit (IDK) is freely available at <http://ingenias.sourceforge.net> with some examples. Its last version is 2.6



(a) OPERA Architecture



(b) OPERA Methodology Steps

Fig. 17. OPERA overview

- *Collaboration autonomy requirement*: activity and interaction in the society must be specified without completely fixing in advance the interaction structures.

Its critical point is that there is not any implementation of the OPERA model which demonstrates the practical possibilities of the model and proves the conceptual choices of OPERA .

MOISE Model of Organization for multi-agent SystEms, MOISE [35], is a methodology structured along three levels:

- **Individual**: definition of the tasks that each agent is responsible of (*roles*, which constrain the action possibilities of each agent according their missions. A role is defined as a set of missions). An agent, playing a given role in the organization, must obey the permitted behaviours specified by the missions building the role.
- **Aggregate**: aggregation of agents in large structures (*groups* which constrain the agents they can cooperate with). It is defined by a set of roles, a set of missions and a set of links. There is two kinds of groups:
 - Internal: links only relate roles of the group (sources and targets).
 - External: in other case. It depends on the policy that is adopted by the designer of the application.
- **Society**: global structuring and interconnection of the agents and structures with each other. (*organizational links*, which constrain the kinds of interaction the agents can have in the systems). To control the execution of the agents in the system, three types of links has been defined:
 - Communication: structure the exchange of information
 - Authority: define the control structure
 - Acquaintance: structure the representation of the other agents and security

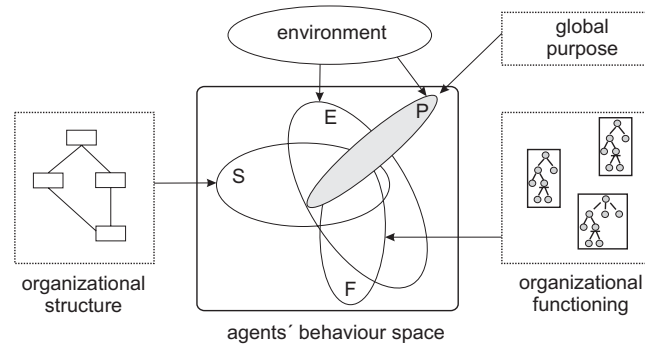
MOISE covers the organizational aspects from two points of view [37]:

- **Functioning**: specification of global plans, policies to allocate tasks to agents, the coordination to execute a plan, and the quality (time consumption, resources usage, . . .) of a plan.
- **Structure**: the roles, the relations among them (*e.g.* communication, authority), roles obligations and permissions, group of roles, etc.

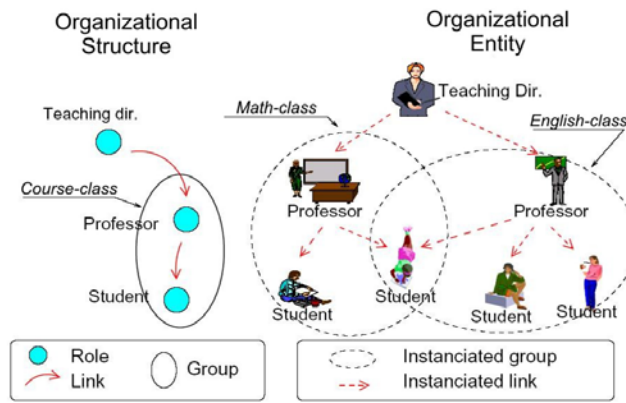
The Fig. 18(a) briefly shows how an organization could explain or constrain the agents behavior in case we consider an organization as having both structural and functional dimensions.

If only the functional dimension is specified, the organization has nothing to “tell” to the agents when no plan can be performed. Otherwise, if only the organizational structure is specified, the agents have to reason for a global plan every time they want to play together.

Thus, in the context of some application domains, we hypothesize that if the organization model specifies both dimensions while maintaining a suitable independence among them, then the MAS that follows such a model can be more effective in leading the group behaviour to its purpose . Another advantage of having both specifications



(a) The organizational effects on a MAS (**P**: all behaviours which draw the MAS's global purposes; **E**: all possible behaviours in the current environment; **S**:organizational structure (roles, groups and links); **F**:functional dimension)



(b) MOISE Example of OS and OE graphs

Fig. 18. MOISEoverview

is that the agents can reason about the others and their organization regarding these two dimensions in order to better interact with them (in the case, for example, of social reasoning).

Two diagrams characterise MOISE model (See example Fig. 18(b)):

- **Organizational Structure (OS)**: a web of roles(nodes), links(arcs) and groups independently of the agents being in the system.
- **Organizational Entity (OE)**: set of agents functioning under an organizational structure. It is an instantiation of an OS.

The constraints defined in the mission imply that the agent will be dependent of others missions for the planning, the execution of action or the use of resources. This implies that an agent has the ability to compute the *dependence relation* that rely the mission of their role to the others agents in the organization considering their roles.

MOISE+ The main shortcoming of MOISE, which motivates its extension, is the lack of the concept of an explicit global plan in the model and the strong dependence among the structure and the functioning [37]. The objective is to create an organization centred model where the first two aspects can be specified almost independently of each other, and can be properly linked by the deontic aspect [38]. The characteristics of each aspect are:

- **Structural:** defines the agents' relations through the notions of roles and links. In this proposal, the original MOISE model is enriched with concepts such as role inheritance, recursive groups, role compatibility, and role cardinality.
- **Functional:** describes how a MAS usually achieves its global goals, i.e., how these goals are decomposed (by plans) and distributed to the agents (by missions). The original MOISE's plans are local to the agents. The MOISE+ contributions here are the inclusion of the concept of global plan, called Social Scheme (SCH), and the definition of preferences between missions.
- **Deontic:** describes the roles' permissions and obligations for missions

Subsequently, it has been emerged variations of MOISE+ [39, 36] as:

- **S-MOISE+:** ensures that the agents follow some of the constraints specified for the organization (cardinality of groups, communication and acquaintance links, role and mission adoption, goal satisfaction), which is interpreted at runtime⁷.
- **J-MOISE+:** helps to program agents with AgentSpeak using the open-source interpreter *Jason*⁸.

A comparison of MOISE+ against AGR is in [12], where it is clarify the equivalence between AGR and a part of the structural dimension of MOISE+.

⁷ S-MOISE+ is freely available at <http://moise.sourceforge.net>

⁸ J-MOISE+ is freely available at <http://jason.sourceforge.net>

3.5 Other methodologies found in the literature

Apart from the described rules-based methodologies, there are some other methodologies or frameworks which we are considered not so important:

CIVIL AGENT SOCIETIES It provides a useful model for designing the infrastructure needed to facilitate the conduct of social interactions achieving the best *quality of service* guarantees (of security, fairness, efficiency...). Its three core elements are: its norms, its institutions and mechanisms for formalizing social interactions as contracts [15].

HARMONIA It is oriented to model regulated electronic organizations from the abstract level where norms usually are defined to the final protocols and procedures that implement those norms [69].

OMNI Organizational Model for Normative Institutions (OMNI) is an extension of OPERA and HARMONIA. It allows the balance of global organizational requirements with the autonomy of individual agents. It is structured in three levels: Abstract, Concrete and Implementation. It is highly modular and it depends on the type of application modules are more or less expansively used [22, 70].

ROADMAP Role Oriented Analysis and Design for Multi-Agent Programming, ROADMAP is variant of GAIA which is oriented to improve engineering open-systems. It has four improvements: formal models of knowledge and the environment, role hierarchies, explicit representation of social structures and relationships, and incorporation of dynamic changes [48].

SODA Societies and infrastructures in the Analysis and Design of Agent-based systems, SODA promotes the separation of individual and social issues, and focuses on the social aspects of agent-oriented software engineering [59].

ADEM It is a comprehensive software development process based on the modified and simplified RUP (Rational Unified Process) and AML (UML-based Agent Modelling Language) ([9, 10]).

4 Comparative Analysis

Several researchers tried to analyse and compare different existing approaches ([1, 65]). Basically, each proposal defines a framework with several slots that are studied and compared in all cases. There are no any existing proposal that covers both organizational and traditional approaches and we may propose another framework. Our framework to evaluate all tools is based on [13, 66, 68]. The proposed framework's evaluation criteria are grouped into five main groups:

- a) **Concepts and properties**, which assess with the main significant agent-oriented concepts and properties which any methodology should implement.
 - *Autonomy*, that express the ability of any agent to solve a problem in a autonomous way.
 - *Communication*, that describes the communication model used, such as message-based or memory-based.
 - *Cooperation issues*, that explains *how* a common-goal is managed by agents.
 - *Adaptability*,
 - *Pro-activity*, if the methodology allows the designer to represent this feature.
- b) **Modelling language criteria** which is addressed with the expressiveness and formalization levels of the defined notation.
 - *Formalisation/preciseness of models* indicates if the models are clearly defined.
 - *Expressiveness* that allows to express the data and the flow of data inside a system.
 - *Abstraction* to create different levels of detail of models.
- c) **Model-related criteria** which evaluate the capabilities of the methodologies models.
 - *Coverage of the life cycle*. The set of phases of the life cycle is covered by the methodology.
 - *Complexity* measures the effort level to learn and use it.
 - *Temporal continuity* that express the changes of the agents across the time.
 - *Human computer-interaction*. Multi agent systems may require to exchange information with (human) users as input and/or output. That interaction should be designed appropriately and represented (typically) as use cases.
- d) **Organizational criteria** which evaluate social relationships among the agent communities.
 - *Open systems* if it allows to represent the incorporation/removal of new agents/resources dynamically.
 - *Topology*. Agent communities relationships could be expressed with different paradigms as summarised in §3.1. Methodologies could be constrained to some of those or be independent.
 - *Social norms* specifies high-level communication patterns among agents or groups of agents.
- e) **Supportive feature criteria** try to give some considerations about support tools.
 - *Software and methodological support* express if any CASE tool exists (e.g. libraries of agents, agent components, architectures or technical support).
 - *Availability of examples* is a useful help during learning or implementing of any methodology.
 - *Usage in projects* is an important factor that express the maturity of a methodology.

Table 2. Summary of studied methodologies with basic details

	MAS-COM-MONKADS	PROMETHE-US	GAIA	PASSI	AUML	AGR	MASE	TROPOS	E-INSTITUTIONS	EXTENDED GAIA	INGENIAS	OPERA	MOISE	MOISE+
<i>Concepts and properties</i>														
Autonomy	+	++	+	++	-	+	+	+	+	+	++	+	++	++
Communication	+	++	++	++	++	+	+	+	+	++	++	+	++	++
Cooperation	+	+	+	+	-	+	+	≈	+	+	++	-	+	+
Adaptability	+	<i>n.d.</i>	-	-	-	+	-	≈	+	-	+	-	≈	≈
Pro-activity	+	++	++	-	--	+	--	-	+	++	++	-	≈	≈
<i>Modeling language</i>														
Formalisation	-	++	++	≈	+	-	-	+	+	++	++	≈	+	+
Expressiveness	-	≈	+	≈	+	-	-	+	+	≈	+	≈	+	+
Abstraction	-	+	+	+	+	-	-	-	+	+	+	+	+	+
<i>Model-I-related</i>														
Coverage	A/D	A/D/I	A/D	A/D/I	A/D	A/D	A/D/I	A/D/I	D	A/D	A/D/I	A/D	A/D	A/D
Complexity	≈	++	-	<i>n.d.</i>	-	-	-	--	≈	<i>n.d.</i>	++	≈	+	+
Temporal	<i>n.d.</i>	≈	-	-	++	--	-	-	++	≈	+	-	+	+
Human-computer	++	-	-	--	--	--	--	--	--	--	-	--	--	--
<i>Organizational criteria</i>														
Open systems	+	-	--	+	-	-	+	+	++	+	++	-	+	+
Topology	+	-	+	-	--	-	-	+	-	-	++	+	+	+
Social norms	≈	-	-	-	≈	-	-	--	+	++	≈	+	+	++
<i>Supportive feature</i>														
Software	-	++	--	++	--	--	++	++	+	--	++	--	-	-
Examples	-	+	+	+	-	-	+	++	+	+	+	--	-	-
Projects	-	++	+	+	-	≈	+	++	+	+	++	-	-	-

Notation: ++: Very high or totally agree; +: High or agree; ≈: Medium or don't specified explicitly by the authors; -: Low or disagree;

--: Very low or totally disagree; *n.d.*: Not available; A/D/I: Analysis/Design/Implementation

Appendix 1: Studied Approaches

Table 3 summarises all studied methodologies in this report and depicts the main information related to all of them such as the authors, the kind of methodology or the main references where the reader can look for more information.

Table 3. Summary of studied methodologies with basic details

Methodology	Authors	Year	Classification*	References
AGR	Ferber and Gutknecht	2004	NRBO	[26, 12]
AUML	Odell and Huget	2000	OO	[57, 3, 27, 43, 44, 42]
E-INSTITUTIONS	Esteva <i>et al.</i>	2001	RBO	[24, 23, 51, 25, 12, 2]
EXTENDED GAIA	Zambonelli <i>et al.</i>	2003	RBO	[75, 76, 8, 7, 40]
GAIA	Wooldridge <i>et al.</i>	2000	AO	[74, 54, 55, 4]
INGENIAS	Gómez-Sanz <i>et al.</i>	2002	RBO	[49, 34, 33, 5, 63, 2]
MAS-COMMONKADS	Iglesias <i>et al.</i>	1999	KE	[45, 34, 68, 28, 40]
MASE	Wood and DeLoach	2000	RBO	[72, 19, 16, 17, 40, 18]
MOISE	Hannoun <i>et al.</i>	2000	RBO	[35, 37]
MOISE+	Hübner <i>et al.</i>	2002	RBO	[37, 38, 12, 39, 36]
OPERA	Dignum	2004	RBO	[20]
PASSI	Cossentino	2002	AO	[11, 40]
PROMETHEUS	Padgham and Winikoff	2002	AO	[61, 60, 62, 40]
TROPOS	Bresciani <i>et al.</i>	2004	NRBO	[57, 6, 32, 22, 53, 50]

*RBO = Rule-Based Organizational; NRBO = Non-Rules-Based Organizational;
OO = Object-Oriented; KE = Knowledge-Engineering Based

References

1. F. Alonso, S. Frutos, L. Martínez, and C. Montes. Towards a Natural Agent Paradigm Development Methodology. In G. Lindemann, J. Denzinger, I. J. Timm, and R. Unland, editors, *Multiagent System Technologies. Proc. of Second German Conference, MATES 2004*, volume 3187 of *LNCS*, pages 155–168. Springer Berlin / Heidelberg, 2004.
2. E. Argente, V. Julian, and V. Botti. Multi-Agent System Development Based on Organizations. *Electronic Notes in Theoretical Computer Science*, 150:55–71, 2006.
3. B. Bauer. UML Class Diagrams Revisited in the Context of Agent-Based Systems. In *Revised Papers and Invited Contributions from the Second International Workshop on Agent-Oriented Software Engineering II*, volume 2222 of *LNCS*, pages 101 – 118. Springer-Verlag / Heidelberg, 2001.
4. F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. Willey, 2007.
5. C. Bernon, M. Cossentino, and J. Pavón. Agent-oriented software engineering. *Knowl. Eng. Rev.*, 20(2):99–116, 2005.
6. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, May 2004.
7. L. Cernuzzi, T. Juan, L. Sterling, and F. Zambonelli. *Methodologies and Software Engineering for Agent Systems*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, chapter The Gaia Methodology: Basic Concepts and Extensions, pages 69–88. Springer US, 2004.
8. L. Cernuzzi and F. Zambonelli. Experiencing AUML in the GAIA Methodology. In *ICEIS (3)*, pages 283–288, 2004.
9. R. Cervenka and I. Trencansky. *The Agent Modeling Language - AML*. Whitestein Series in Software Agent Technologies and Autonomic Computing. Springer / Birkhäuser, 2007.
10. R. Cervenka, I. Trencansky, M. Calisti, and D. Greenwood. AML: Agent Modeling Language Toward Industry-Grade Agent-Based Modeling. In *5th International Workshop, AOSE 2004. Revised Selected Papers*, volume 3382 of *LNCS*, pages 31–46. Springer Berlin / Heidelberg, 2004.
11. A. Chella, M. Cossentino, L. Sabatucci, and V. Seidita. Agile PASSI: An Agile Process for Designing Agents. *International Journal of Computer Systems Science & Engineering. Special issue on "Software Engineering for Multi-Agent Systems"*, 21(2), 2006.
12. L. R. Coutinho, J. S. Sichman, and O. Boissier. Modeling Organization in MAS: A Comparison of Models. In *1st. Workshop on Software Engineering for Agent-Oriented Systems, SEAS'05*, Uberlândia, 2005.
13. K. H. Dam and M. Winikoff. Comparing Agent-Oriented Methodologies. In P. Giorgini, B. Henderson-Sellers, and M. Winikoff, editors, *Agent Oriented Information Systems. 5th International Bi-Conference Workshop, AOIS 2003. Revised Selected Papers.*, volume 3030 of *LNAI*, pages 78–93. Springer Berlin / Heidelberg, 2004.
14. M. Dastani, V. Dignum, and F. Dignum. Organizations and Normative Agents. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *EurAsia-ICT 2002: Information and Communication Technology: First EurAsian Conference Shiraz. Proceedings*, volume 2510 of *LNCS*, pages 982–989. Springer Berlin / Heidelberg, 2002.
15. C. Dellarocas and M. Klein. Civil agent societies: Tools for inventing open agent-mediated electronic marketplaces. In *Agent Mediated Electronic Commerce (IJCAI Workshop)*, pages 24–39, 1999.
16. S. A. DeLoach. Analysis and Design using MaSE and agentTool. In *12th Midwest Artificial Intelligence and Cognitive Science Conference, MAICS 01*, Oxford, Ohio, 2001.

17. S. A. DeLoach. *Methodologies and Software Engineering for Agent Systems*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, chapter The MaSE Methodology, pages 107–125. Springer US, 2004.
18. S. A. DeLoach. Engineering Organization-Based Multiagent Systems. In A. García, editor, *Software Engineering for Multi-Agent Systems IV: 4th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, SELMAS 2005*, volume 3914 of *LNCS*, pages 109–125. Springer Berlin / Heidelberg, 2006.
19. S. A. DeLoach, M. F. Wood, and C. H. Sparkman. Multiagent Systems Engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001.
20. V. Dignum. *A Model for Organizational Interaction: Based on Agents, Founded in Logic*. PhD thesis, Universiteit Utrecht, 2004.
21. V. Dignum, J. J. Meyer, H. Weigand, and F. Dignum. An Organizational-Oriented Model for Agent Societies. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems, AAMAS 02*, volume Part 2, pages 694–5. ACM Press, 2002.
22. V. Dignum, J. Vázquez-Salceda, and F. Dignum. OMNI: Introducing Social Structure, Norms and Ontologies into Agent Organizations. In *PROMAS*, pages 181–198, 2004.
23. M. Esteva, J. A. Padget, and C. Sierra. Formalizing a Language for Institutions and Norms. In *ATAL '01: Revised Papers from the 8th International Workshop on Intelligent Agents VIII*, pages 348–366, London, UK, 2002. Springer-Verlag.
24. M. Esteva, J. Rodríguez-Aguilar, C. Sierra, P. García, and J. Arcos. On the Formal Specification of Electronic Institutions. In F. Dignum and C. Sierra, editors, *Agent mediated electronic commerce*, volume 1991 of *LNAI*, pages 126–147. Springer Berlin / Heidelberg, 2001.
25. M. Esteva, B. Rosell, J. A. Rodríguez-Aguilar, and J. L. Arcos. AMELI: An Agent-Based Middleware for Electronic Institutions. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 236–243, Washington, DC, USA, 2004. IEEE Computer Society.
26. J. Ferber, O. Gutknecht, and F. Michel. From Agents to Organizations: an Organizational View of Multi-Agent Systems. In P. Giorgini, J. Müller, and J. Odell, editors, *Agent-Oriented Software Engineering (AOSE) IV*, volume 2935 of *LNAI*, pages 214–230. Springer, 2004.
27. FIPA. FIPA Communicative Act Library Specification. Technical Report SC00037J, Foundation for Intelligent Physical Agents, 2002. Available at <http://www.fipa.org/specs/fipa00037/>.
28. F. Gallego, F. Llorens, and R. Rizo. Breve análisis de algunas metodologías de diseño de SMA. Technical report, Universidad de Alicante, 2004. Available at <http://www.dccia.ua.es/dccia/inf/asignaturas/AI/docs/SMA.pdf>.
29. Gartner. *Hype Cycle for Human-Computer Interaction*. Gartner Inc., 2004.
30. Gartner. *Hype Cycle for Human-Computer Interaction*. Gartner Inc., 2006.
31. M. Georgeff, B. Pell, M. Pollack, M. Tambe, and M. Wooldridge. The Belief-Desire-Intention Model of Agency. In J. Müller, M. P. Singh, and A. S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages, ATAL 98*, number 1555 in *LNCS*, pages 1–10. Springer Verlag / Heidelberg, 1999.
32. P. Giorgini, M. Kolp, J. Mylopoulos, and M. Pistore. *Methodologies and Software Engineering for Agent Systems*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, chapter The Tropos Methodology: An Overview, pages 89–106. Springer US, 2004.
33. A. Giret and V. J. Botti. Towards a Recursive Agent Oriented Methodology for Large-Scale MAS. In *AOSE*, pages 25–35, 2003.
34. J. Gómez Sanz. Metodologías para el desarrollo de sistemas multi-agente. *Revista Iberoamericana de Inteligencia Artificial*, 18(3):51–63, 2003.

35. M. Hannoun, O. Boissier, J. S. Sichman, and C. Sayettat. MOISE : An Organizational Model for Multi-agent Systems. In M. C. Monard and J. S. Sichman, editors, *Advances in Artificial Intelligence: Proceedings of International Joint Conference, 7th Ibero-American Conference on AI, 15th Brazilian Symposium on AI, IBERAMIA-SBIA 2000*, volume 1952 of *LNCS*, pages 156–165. Springer Berlin / Heidelberg, 2000.
36. J. Hübner, O. Boissier, and J. Sichman. Programming MAS reorganisation with Moise+. Talk in Dagstuhl Seminar - Foundations and Practice of Programming Multi-Agent Systems, 2006, 2006.
37. J. F. Hübner, J. S. Sichman, and O. Boissier. A Model for the Structural, Functional, and Deontic Specification of Organizations in Multiagent Systems. In G. Bittencourt and G. L. Ramalho, editors, *Proceedings of the 16th Brazilian Symposium on Artificial Intelligence, SBIA 2002*, volume 2507 of *LNAI*, pages 118–128. Springer Verlag, 2002.
38. J. F. Hübner, J. S. Sichman, and O. Boissier. MOISE+: Towards a Structural, Functional, and Deontic Model for MAS Organization. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2002)*, pages 501–502, Bologna, Italy, 2002. ACM Press.
39. J. F. Hübner, J. S. Sichman, and O. Boissier. S-MOISE+: A Middleware for developing Organised Multi-Agent Systems. In *Proc. of International Workshop on Organizations in Multi-Agent Systems at AAMAS 2005*, 2005.
40. B. Henderson-Sellers and P. e. Giorgini, editors. *Agent-Oriented Methodologies*. Idea Group, 2005.
41. B. Horling and V. Lesser. A Survey of Multi-Agent Organizational Paradigms. *The Knowledge Engineering Review*, 19(4):281 – 316, 2005.
42. M.-P. Huget. Agent UML Notation for Multiagent System Design. *IEEE Internet Computing*, 8(4):63–71, July/August 2004.
43. M.-P. Huget and J. Odell. Representing Agent Interaction Protocols with Agent UML. In *5th International Workshop, AOSE 2004. Revised Selected Papers*, volume 3382 of *LNCS*, pages 16–30. Springer Berlin / Heidelberg, 2004.
44. M.-P. Huget, J. Odell, and B. Bauer. *Methodologies and Software Engineering for Agent Systems*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, chapter The AUML Approach, pages 237–257. Springer US, 2004.
45. C. A. Iglesias, M. Garijo, and J. C. González. A Survey of Agent-Oriented Methodologies. In J. Müller, M. P. Singh, and A. S. Rao, editors, *5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL'98)*, volume 1555, pages 317–330, Paris, France, 1999. Springer Verlag.
46. J. Iivari, R. Hirschheim, and H. Klein. Beyond methodologies: keeping up with information systems development approaches through dynamic classification. In *Proceedings of the 32nd Annual Hawaii International Conference on System Sciences, 1999. HICSS-32.*, pages 10–20, Maui, HI, USA, 1999. IEEE Press.
47. I. Jacobson, G. Booch, and J. Rumbaugh. *The unified software development process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
48. T. Juan, A. Pearce, and L. Sterling. ROADMAP: extending the gaia methodology for complex open systems. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 3–10, New York, NY, USA, 2002. ACM Press.
49. S. Kelly, K. Lyytinen, and M. Rossi. MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. *LNCS*, 1080, 1996.
50. M. Kolp, P. Giorgini, and J. Mylopoulos. Multi-Agent Architectures as Organizational Structures. *Autonomous Agents and Multi-Agent Systems*, 13(1):3–25, 2006.
51. M. Luck, P. McBurney, and C. Preist. *Agent Technology: Enabling Next Generation Computing. A Roadmap for Agent Based Computing*. AgentLink, 2003.

52. M. Luck, P. McBurney, O. Shehory, and S. Willmott. *Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing)*. AgentLink, 2005.
53. A. U. Mallya and M. P. Singh. Incorporating Commitment Protocols into Tropos. In J. P. Müller and F. Zambonelli, editors, *AOSE*, volume 3950 of *LNCS*, pages 69–80. Springer Verlag / Heidelberg, 2005.
54. P. Moraitis, E. Petraki, and N. I. Spanoudakis. Engineering JADE Agents with the Gaia Methodology. In R. K. et. al., editor, *Agent Technology Workshops 2002*, volume 2592 of *LNAI*, pages 77–91. Springer Verlag, 2002.
55. P. Moraitis and N. I. Spanoudakis. Combining gaia and jade for multi-agent systems development. In *17th European Meeting on Cybernetics and Systems Research (EMCSR 2004)*, 2004.
56. J. Odell. Objects and Agents Compared. *Journal of Object Technology*, 1:41–53, 2002.
57. J. Odell, H. V. Dyke, and B. Bauer. Extending UML for Agents. In *Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, AOIS 2000*, pages 3–17, 2000.
58. OMG. Unified Modeling Language. Resource page. Available at www.uml.org.
59. A. Omicini. SODA: Societies and Infrastructures in the Analysis and Design of Agent-based Systems. In *AOSE*, pages 185–193, 2000.
60. L. Padgham and M. Winikoff. Prometheus: A Methodology for Developing Intelligent Agents. In *First international joint conference on Autonomous agents and multiagent systems, AAMAS 2002*, pages 37–38. ACM Press, 2002.
61. L. Padgham and M. Winikoff. Prometheus: A Pragmatic Methodology for Engineering Intelligent Agents. In J. Debenham, B. Henderson-Sellers, N. Jennings, and J. Odell, editors, *OOPSLA 2002 Workshop on Agent-Oriented Methodologies*, pages 97–108, 2002.
62. L. Padgham and M. Winikoff. *Developing Intelligent Agent Systems: A Practical Guide*. Wiley Series in Agent Technology. John Wiley and Sons, 2004.
63. J. Pavón, J. J. Gómez-Sanz, and R. Fuentes. Model Driven Development of Multi-Agent Systems. In *ECMDA-FA*, pages 284–298, 2006.
64. J. S. Sichman, V. Dignum, and C. Castelfranchi. Agents Organizations: A Concise Overview. *Brazilian Computer Society, Special Issue on Agents Organizations*, 11(1):1–8, July 2005.
65. A. Sturm and O. Shehory. *Methodologies and Software Engineering for Agent Systems*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, chapter A Comparative Evaluation of Agent-Oriented Methodologies, pages 127–149. Springer US, 2004.
66. J. Sudeikat, L. Braubach, A. Pokahr, and W. Lamersdorf. Evaluation of Agent - Oriented Software Methodologies - Examination of the Gap Between Modeling and Platform. In P. Giorgini, J. P. Müller, and J. Odell, editors, *Agent-Oriented Software Engineering V, Fifth International Workshop AOSE 2004*, volume 3382 of *LNCS*, pages 126–141. Springer Berlin / Heidelberg, 2004.
67. J. Thangarajah, L. Padgham, and M. Winikoff. Prometheus Design Tool. In *Proceedings of the fourth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS05*, pages 127–128. ACM Press, 2005.
68. Q.-N. N. Tran, G. Low, and M.-A. Williams. A preliminary comparative feature analysis of multi-agent systems development methodologies. In P. Bresciani, P. Giorgini, B. Henderson-Sellers, G. Low, and M. Winikoff, editors, *Agent-Oriented Information Systems II. 6th International Bi-Conference Workshop, AOIS 2004. Revised Selected Papers*, volume 3508 of *LNCS*, pages 386–398. Springer Berlin / Heidelberg, 2005.
69. J. Vázquez-Salceda and V. Dignum. Modelling Electronic Organizations. In J. G. Carbonell and J. Siekmann, editors, *Multi-Agent Systems and Applications III: 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003. Proceedings*, volume 2691 of *LNAI*, pages 1070–80, 2003.

70. J. Vázquez-Salceda, V. Dignum, and F. Dignum. Organizing Multiagent Systems. *Autonomous Agents and Multi-Agent Systems*, 11(3):307–360, 2005.
71. W3C. Extensible Markup Language. Resource page. Available at <http://www.w3.org/XML/>.
72. M. F. Wood and S. DeLoach. An Overview of the Multiagent Systems Engineering Methodology. In *AOSE*, pages 207–222, 2000.
73. M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley and Sons, 2002.
74. M. Wooldridge and P. Ciancarini. Agent-Oriented Software Engineering: The State of the Art. In *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000*, volume 1957 of *LNCS*, pages 55–82. Springer, 2000.
75. F. Zambonelli, N. Jennings, and M. Wooldridge. Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):303–328, 2001.
76. F. Zambonelli, N. Jennings, and M. Wooldridge. Developing Multiagent Systems: The Gaia Methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370, 2003.