Escuela Politécnica Superior (Jaén)
Ingeniería en Informática

**UNIVERSIDAD DE JAÉN**
*Escuela Politécnica Superior (Jaén)*

Proyecto Fin de Carrera

# SISTEMA DE SOPORTE AL CONSENSO EN PROBLEMAS DE TOMA DE DECISIÓN EN GRUPO BASADO EN AGENTES INTELIGENTES

**Alumno: Francisco José Quesada Real**

Tutores: Prof. Dr. D. Luis Martínez López
Prof. D. Iván Palomares Carrascosa

Dpto: Informática

**Junio, 2013**

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Prof. Dr. D. Luis Martínez López y Prof. D. Iván Palomares Carrascosa, tutores del Proyecto Fin de Carrera titulado: Sistema de Soporte al Consenso en Problemas de Toma de Decisión en Grupo Basado en Agentes Inteligentes, que presenta D. Francisco José Quesada Real , autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, junio de 2013

El alumno: Los tutores:

D. Francisco José Quesada Real

Prof. Dr. D. Luis Martínez López
Prof. D. Iván Palomares Carrascosa

*A Luis, por ser mi Pigmalión, e Iván*

*Quisiera hacer extensibles los agradecimientos de esta memoria a todas aquellas personas que desde los inicios de mi formación académica, han contribuido aportando su granito de arena para que este proyecto sea una realidad especialmente:*

*A mis padres, a mi hermana, a mis abuelos y a mi novia por su apoyo y cariño constante, sin los que no habría podido llegar hasta aquí.*

*A Paco y Pedro por su ayuda y aportaciones en temas de consenso y agentes.*

*A Iván y a Luis que han sido los principales artífices del inicio, la elaboración y la finalización de este proyecto. Mil gracias.*

# Índice general

**Índice de figuras** VII

**1. Introducción** **1**

1.1. Introducción al Proyecto . . . . . 1

1.2. Propósito . . . . . 4

1.3. Objetivos . . . . . 4

1.4. Estructura y Planificación del Proyecto . . . . . 5

**2. Sistemas Multi-Agente** **7**

2.1. Introducción . . . . . 7

2.2. Clasificaciones de Agentes . . . . . 8

2.2.1. Clasificación según las principales propiedades de agente . . . . . 8

2.2.2. Clasificación según sus características individuales . . . . . 9

2.2.3. Clasificación según el entorno en el que funcionan . . . . . 10

2.2.4. Clasificación según el modo de organización . . . . . 10

2.3. Ciclo de vida de un agente . . . . . 13
2.3.1. Estados de un agente . . . . . 13
2.3.2. Transiciones entre estados . . . . . 15
2.4. Definición de SMA . . . . . 16
2.5. Arquitecturas de SMA . . . . . 17
2.5.1. Tipos de Arquitecturas de SMA . . . . . 18
2.5.2. Arquitecturas estandarizadas de SMA . . . . . 22
2.6. Comunicación entre agentes . . . . . 25
2.6.1. Actos comunicativos . . . . . 27
2.6.1.1. Información . . . . . 27
2.6.1.2. Realización . . . . . 29
2.6.1.3. Negociación . . . . . 30
2.6.1.4. Intermediación . . . . . 30
2.6.2. Estándares de comunicación . . . . . 31
2.6.2.1. ACL . . . . . 31
2.6.2.2. KQML . . . . . 32
2.6.3. Protocolos de comunicación . . . . . 33
2.7. Modelado de Agentes en SMA . . . . . 42
2.7.1. Medotologías desde el punto de vista del SMA . . . . . 43
2.7.2. Metodologías atendiendo a roles, servicios, interacciones y organizaciones 48

2.7.3. Unificación de notaciones de modelado de SMA: AUML . . . . . 54

2.8. Implementación de un SMA . . . . . 56

2.8.1. Lenguajes de programación . . . . . 56

2.8.2. Plataformas de Sistemas Multi-Agente . . . . . 57

2.9. Problemas y Carencias de los SMA . . . . . 62

**3. Arquitectura Multi-Agente** **65**

3.1. Negociación . . . . . 65

3.1.1. Kasbah. Ejemplo de Plataforma de Negociación . . . . . 68

3.2. Diseño de la plataforma: Arquitectura Multi-Agente . . . . . 71

3.3. Ontologías . . . . . 74

3.3.1. Definición . . . . . 75

3.3.2. Tipos de Ontologías . . . . . 78

3.3.3. Lenguajes de Representación . . . . . 80

3.3.4. Ontologías en JADE . . . . . 82

3.4. Integración con Interfaz Web de Usuario . . . . . 84

**4. Desarrollo del Proyecto** **87**

4.1. Descripción del proyecto . . . . . 87

4.2. Especificación de requerimientos . . . . . 89

4.2.1. Requerimientos funcionales . . . . . 91

4.2.2. Requerimientos no funcionales . . . . . 93

4.3. Análisis del sistema . . . . . 96

4.3.1. Diagrama Frontera . . . . . 97

4.3.2. Principales casos de uso . . . . . 97

4.3.3. Escenarios . . . . . 108

4.4. Diseño del sistema . . . . . 115

4.4.1. Diseño de objetos . . . . . 115

4.4.1.1. Diagrama de clases de diseño . . . . . 116

4.4.1.2. Diagramas de Secuencia . . . . . 118

4.4.2. Diseño de interfaz . . . . . 123

4.4.2.1. Diseño de Pantallas . . . . . 124

4.4.2.2. Caminos de Navegación . . . . . 129

4.4.2.3. Guía de Estilo . . . . . 138

4.4.3. Diseño de los datos . . . . . 141

4.4.3.1. Esquema Conceptual . . . . . 145

4.4.3.2. Esquema Conceptual Modificado . . . . . 147

4.4.4. Diseño de la Ontología . . . . . 149

4.5. Implementación . . . . . 158

4.5.1. Arquitectura del Sistema . . . . . 158

4.5.2. Lenguajes de programación entornos de desarrollo . . . . . 159

4.5.3. Herramientas de Desarrollo . . . . . 160

**5. Conclusiones** **165**

5.1. Conclusiones finales . . . . . 165

5.2. Trabajos futuros . . . . . 167

**A. Procesos de Alcance de Consenso** **171**

A.1. Introducción . . . . . 171

A.2. Problemas de Toma de Decisión en Grupo (TDG) . . . . . 172

A.3. El Consenso en Problemas de TDG . . . . . 175

A.3.1. Procesos de Consenso . . . . . 177

A.3.2. Esquema General de los Procesos de Consenso . . . . . 177

A.4. Modelo de Consenso Propuesto . . . . . 178

A.4.1. Llamada de participación en el Proceso de Consenso . . . . . 180

A.4.2. Expresión de Preferencias de los Expertos . . . . . 181

A.4.3. Cálculo del Grado de Consenso . . . . . 182

A.4.4. Control de Consenso . . . . . 183

A.4.5. Generación de Recomendaciones . . . . . 183

**B. Manual de Instalación** **187**

B.1. Instalación del Servidor . . . . . 187

B.1.1. Instalación de XAMPP . . . . . 188

B.1.2. Configuración Base de datos . . . . . 193

B.1.3. Configuración de Tomcat . . . . . 197

B.1.4. Instalación JADE . . . . . 199

**C. Manual del Usuario "Administrador" 205**

C.1. Acceso . . . . . 205

C.2. Iniciar Problema . . . . . 206

C.3. Ver Problemas . . . . . 208

C.4. Ver Resultados de un Problema . . . . . 210

C.5. Salir . . . . . 212

**D. Manual del Usuario "Experto" 213**

D.1. Acceso . . . . . 213

D.2. Ver Mis Problemas . . . . . 214

D.3. Participar en el Proceso de Negociación . . . . . 216

D.4. Ver Resultados de Mis Problemas . . . . . 217

D.5. Salir . . . . . 219

**Bibliografía 221**

# Índice de figuras

2.1. Agentes reactivos . . . . . 9

2.2. Agentes individuales . . . . . 11

2.3. Agentes cooperativos . . . . . 12

2.4. Agentes negociadores . . . . . 12

2.5. Agentes competitivos . . . . . 13

2.6. Ciclo de vida de agentes según FIPA . . . . . 14

2.7. Esquema básico de la arquitectura de subsunción . . . . . 20

2.8. Diferentes esquemas de arquitectura híbrida . . . . . 21

2.9. Modelo de referencia del estándar FIPA . . . . . 23

2.10. Arquitectura RETSINA . . . . . 24

2.11. Niveles de la Arquitectura RETSINA . . . . . 26

2.12. Elementos de un mensaje FIPA-ACL . . . . . 32

2.13. Elementos de un mensaje KQML . . . . . 33

2.14. Protocolo Subscribe . . . . . 34

2.15. Protocolo Request . . . . . 35

2.16. Protocolo Request-when . . . . . 36

2.17. Protocolo Contract Net . . . . . 37

2.18. Protocolo Iterated Contract Net . . . . . 38

2.19. Protocolo Subasta Inglesa . . . . . 39

2.20. Protocolo Subasta Holandesa . . . . . 40

2.21. Protocolo Brokering . . . . . 41

2.22. Protocolo Recruiting . . . . . 42

2.23. Modelos de MAS-CommonKADS . . . . . 46

2.24. Puntos de vista de un SMA en la metodología INGENIAS . . . . . 48

2.25. Fases de modelado software en Gaia . . . . . 50

2.26. Fases de modelado software en MaSE . . . . . 52

2.27. Fases de modelado software en AALAADIN . . . . . 54

2.28. Arquitectura de JADE . . . . . 59

2.29. Entorno de JADE . . . . . 60

2.30. Entorno de ABLE . . . . . 61

3.1. Estrategias de negociación agente-vendedor en Kasbah . . . . . 70

3.2. Arquitectura del SMA de Negociación . . . . . 73

3.3. Modelo de contenido para ontologías en JADE . . . . . 83

3.4. Funcionamiento de JadeGateway . . . . . 86

4.1. Fases de la Ingeniería del Software . . . . . 89

4.2. Diagrama Frontera . . . . . 97

4.3. Diagrama de Caso de Uso IdentificarUsuario . . . . . 98

4.4. Diagrama de Caso de Uso GestionUsuarios . . . . . 99

4.5. Diagrama de Caso de Uso GestionProblemas . . . . . 102

4.6. Diagrama de Caso de Uso Proceso Consenso . . . . . 106

4.7. Diagrama de Caso de Uso Consultar Datos Problema . . . . . 107

4.8. Diagrama de Clases de Diseño. . . . . 117

4.9. Diagrama de Secuencia Agente Moderador - Agente Experto. . . . . 118

4.10. Diagrama de Secuencia Agente Moderador - Agente Evaluador de Consenso. 120

4.11. Diagrama de Secuencia Agente Moderador - Agente Identificador de Cambios 121

4.12. Diagrama de Secuencia General . . . . . 122

4.13. Pantalla de Acceso . . . . . 124

4.14. Pantalla Menú Principal del Usuario Administrador . . . . . 125

4.15. Pantalla Iniciar Problema . . . . . 125

4.16. Pantalla Ver Problemas . . . . . 126

4.17. Pantalla Parámetros de un Problema . . . . . 126

4.18. Pantalla Menú Principal Usuario Experto . . . . . 127

4.19. Pantalla Ver Mis Problemas . . . . . 127

4.20. Pantalla Ver Datos de mi Problema . . . . . 128

4.21. Pantalla Participar en Proceso de Negociación . . . . . 128

4.22. Pantalla Introducir Valoraciones . . . . . 129

4.23. Storyboard Acceso usuario Administrador . . . . . 131

4.24. Storyboard Iniciar Problema . . . . . 132

4.25. Storyboard Ver Problemas . . . . . 133

4.26. Storyboard Salir del Menú Administrador . . . . . 134

4.27. Storyboard Acceso usuario Experto . . . . . 135

4.28. Storyboard Ver Mis Problemas . . . . . 136

4.29. Storyboard Participar en el Proceso de Negociación . . . . . 137

4.30. Storyboard Salir del Menú Experto . . . . . 138

4.31. Entidad . . . . . 142

4.32. Relación . . . . . 142

4.33. Atributo . . . . . 142

4.34. Entidad Débil . . . . . 143

4.35. Cardinalidad Uno a Uno . . . . . 143

4.36. Cardinalidad Uno a Muchos . . . . . 143

4.37. Cardinalidad Muchos a Muchos . . . . . 144

4.38. Clave de un único atributo . . . . . 144

4.39. Esquema Conceptual . . . . . 146

4.40. Esquema Conceptual Modificado . . . . . 148

4.41. Ontología de la Plataforma de Negociación orientada al Consenso . . . . . 150

4.42. Arquitectura Cliente/Servidor . . . . . 159

4.43. Entorno de programación NetBeans 7.2 . . . . . 161

4.44. Esqueleto Ontología JADE . . . . . 164

4.45. Pantalla principal de PhpMyAdmin . . . . . 164

A.1. Fases del proceso de selección en problemas de TDG . . . . . 173

A.2. Esquema General de "Soft Consensus" para TDG . . . . . 179

A.3. Modelo de Consenso . . . . . 180

B.1. Ventana de elección de idioma de instalación . . . . . 188

B.2. Elección inglés como idioma de instalación . . . . . 189

B.3. Pantalla de inicio de instalación . . . . . 189

B.4. Pantalla de elección del lugar de instalación . . . . . 190

B.5. Pantalla de elección de opciones de instalación . . . . . 191

B.6. Pantalla de instalación completada . . . . . 191

B.7. Permitir acceso de Apache HTTP Server . . . . . 192

B.8. Ventana de finalización del servicio de instalación . . . . . 192

B.9. Pantalla de Bienvenida de XAMPP . . . . . 193

B.10.Pantalla de inicial de phpMyAdmin . . . . . 194

B.11.Seleccionar archivo a importar . . . . . 195

B.12.Finalizar importación . . . . . 195

B.13.Base de Datos que se acaba de importar . . . . . 196

B.14.Panel de control de XAMPP . . . . . 197

B.15.Ventana para instalar servicio tomcat . . . . . 198

B.16.Iniciar tomcat . . . . . 198

B.17.Permitir acceso al firewall . . . . . 199

B.18.Ejecución de tomcat . . . . . 200

B.19.Archivos incluidos en JADE-all-4.3.0.zip . . . . . 200

B.20.Copiar la carpeta JADE en C: . . . . . 201

B.21.Librerías de JADE . . . . . 201

B.22.Creación de un Documento de texto en el directo JADE . . . . . 202

B.23.Cambiar extensión al Documento de texto . . . . . 202

B.24.Dato a añadir en el archivo Runjade.bat . . . . . 203

B.25.Sentencias de ejecución de la plataforma JADE . . . . . 203

B.26.Interfaz de la plataforma JADE . . . . . 204

C.1. Página Principal . . . . . 205

C.2. Acceso como Usuario Administrador . . . . . 206

C.3. Menú Principal Usuario Administrador . . . . . 206

C.4. Acceder a Iniciar Problema . . . . . 207

C.5. Iniciar Problema . . . . . 207

C.6. Menú Principal Usuario Administrador . . . . . 208

C.7. Selección de “Ver Problema” . . . . . 208

C.8. Selección del problema para ver su parámetros . . . . . 209

C.9. Parámetros del problema seleccionado . . . . . 209

C.10.Selección de “Ver Problema” . . . . . 210

C.11.Selección del problema para ver su parámetros . . . . . 211

C.12.Resultados del problema seleccionado . . . . . 211

C.13.Seleccionar “Salir” . . . . . 212

C.14.Página Principal . . . . . 212

D.1. Página Principal . . . . . 213

D.2. Acceso como Usuario Experto . . . . . 214

D.3. Menú Principal Usuario Experto . . . . . 214

D.4. Selección de “Ver Mis Problemas” . . . . . 215

D.5. Selección de un problema en el apartado “Ver Mis Problemas” . . . . . 215

D.6. Ventana Participar en el problema . . . . . 216

D.7. Pantalla Introducir Valoraciones y Seleccionar Estrategia . . . . . 217

D.8. Selección de “Ver Mis Problemas” . . . . . 218

D.9. Selección de un problema en el apartado “Ver Mis Problemas” . . . . . 218

D.10.Resultados de un problema . . . . . 219

D.11.Seleccionar “Salir” . . . . . 220

D.12.Página Principal . . . . . 220

# Capítulo 1

# Introducción

## 1.1. Introducción al Proyecto

La Inteligencia Artificial, IA, se define según la RAE [1] como *el desarrollo y utilización de ordenadores con los que se intentan reproducir los procesos de la inteligencia humana* [15]. Actualmente una de las líneas que atraen mayor atención en IA es el llamado «paradigma de agentes inteligentes». Este paradigma aborda el desarrollo de entidades que puedan actuar de forma autónoma y razonada, pudiendo entender que la IA, en un sentido amplio, trata de construir dichas entidades autónomas e inteligentes [38, 127].

Hoy en día, no existe una única definición para el término de agente [68, 72] aunque una buena aproximación puede ser la que nos dice que *el concepto de "agente" caracteriza a una entidad software con una arquitectura robusta y adaptable que puede funcionar en distintos entornos o plataformas software y es capaz de realizar de forma "inteligente" y autónoma distintos objetivos intercambiando información con el entorno o con otros agentes humanos o software* [54].

Las principales características que definen el comportamiento de un agente son [25]:

- Funcionamiento continuo y autónomo.

[1] Real Academia Española

- Comunicación con el entorno y con otros agentes, a través de un lenguaje o formalismo de comunicación.
- Robustez, entendiéndose ésta como la capacidad para que el agente pueda cumplir sus objetivos y tareas aunque se le presenten adversidades.
- Adaptabilidad, entendida como la capacidad de cumplir objetivos y tareas en diferentes contextos de forma flexible.

Otros atributos importantes que pueden presentar algunos agentes son:

- Razonamiento y Aprendizaje, requisito necesario para que los agentes se comporten de manera inteligente.
- Movilidad, propiedad que permite a los agentes móviles desplazarse por los nodos de una red y ejecutarse en distintas plataformas.

La evolución de este paradigma ha llevado al modelado de los sistemas como organizaciones de múltiples agentes, pasando a denominarse Sistemas Multi-Agente (SMA).

Los SMA se han extendido en la actualidad, estando presentes en numerosas áreas como son: la industria, la medicina, la recuperación de información, el comercio electrónico, las telecomunicaciones, etc [5, 12, 17, 25, 54] .

Algunos ejemplos de SMA en estas áreas son:

- OASIS [25]: Es un sofisticado sistema de control de tráfico aéreo basado en agentes que se está utilizando en el aeropuerto de Sydney. En este sistema se utilizan agentes para representar tanto a los aviones como a los distintos sistemas de control de tráfico aéreo, de manera que cuando un avión entra en el espacio aéreo de Sydney se le asigna un agente con la información y los objetivos del avión.
- WebMate [17]: Es un agente personal que trata de acompañar a un usuario cuando navega por la web, buscando y proporcionándole ayuda inteligente.

- MASFIT [12]: Se trata de un sistema de subastas intermediadas por agentes que permite a los compradores participar en varias subastas que tienen lugar simultáneamente. Actualmente está siendo usado en el Portal de Servicios de Lonjas y del sector Pesquero [9].

- MARVIN [5]: Es un SMA para acceso a información médica, capaz de traducir las consultas automáticamente a diversos idiomas (español, francés, inglés, italiano, alemán) con la ayuda de un diccionario de 20000 términos médicos. Así MARVIN puede hacer la misma consulta a múltiples fuentes de información de todo el mundo.

- MAITE [54]: Uno de los primeros sistemas desarrollados como herramienta de apoyo a la operación y mantenimiento de la red de centrales digitales IBERCOM. El sistema procesa alarmas generadas por el sistema de gestión, localiza el nodo de la red que las está produciendo e interactúa con el sistema de control del nodo para investigar las causas de las alarmas, realizar un diagnóstico e indicar al operador las acciones de reparación adecuadas.

Entendemos como proceso de negociación al proceso de resolución de un conflicto entre dos o más entidades (individuos o grupos) entre las que hay un conflicto de intereses. No obstante, es posible que ambas partes tengan que cooperar para la consecución de dichos intereses. Para poder satisfacer a todas las entidades, hay que llegar a lo que conocemos como *consenso* mediante un proceso de negociación durante el cual los individuos discuten y acercan sus opiniones entre sí para alcanzar un mayor nivel de acuerdo colectivo.

El presente proyecto se basará en la construcción de un Sistema Multi-Agente (colaborativo) para dar soporte a procesos de negociación. En particular, dicha plataforma será instanciada como sistema de soporte al consenso en problemas de Toma de Decisión en Grupo (TDG). Para ello, construiremos una arquitectura Multi-Agente, en la que crearemos los agentes necesarios para la resolución de nuestro problema concreto. Estos agentes serán los encargados de llevar a cabo procesos de consenso según el modelo de consenso implementado, descrito en el Anexo A.

## 1.2. Propósito

Implementar un prototipo de una plataforma Multi-Agente colaborativa, que permita introducir comportamientos autónomos e inteligentes en problemas de negociación para su utilización como un Sistema de Soporte al Consenso (SSC), el cual será aplicado en la resolución de problemas de TDG.

## 1.3. Objetivos

- Realizar una revisión de la tecnología Multi-Agente (arquitecturas, plataformas, estándares, etc).
- Definición de la arquitectura necesaria para la plataforma Multi-Agente.
- Análisis y diseño de los agentes necesarios para llevar a cabo procesos de consenso en TDG.
- Diseño de las ontologías necesarias para que los agentes del sistema puedan comunicarse de forma adecuada dependiendo del problema a resolver mediante la plataforma.
- Revisar el modelo de consenso que será implementado en el SSC.
- Diseñar un modelo automático para el proceso de negociación.
- Instanciar un SSC basado en agentes inteligentes con una interfaz web de comunicación con los usuarios.
- Desarrollar la documentación relativa al desarrollo de la plataforma del sistema Multi-Agente.

## 1.4. Estructura y Planificación del Proyecto

A continuación, se realizará una breve descripción de los capítulos en los que está estructurado este proyecto, así como los contenidos expuestos en los mismos. Como hemos visto, el primer capítulo es una introducción general al proyecto que se ha realizado, donde se recoge la motivación, la definición del propósito y los objetivos que persigue.

En el capítulo 2, abordamos una visión general de los Sistemas Multi-Agente. En primer lugar, definiremos qué es un Agente Software y un Sistema Multi-Agente, y comentaremos la justificación de su utilización. Acto seguido estudiaremos los modelos y arquitecturas Multi-Agente, la forma en que se lleva a cabo la comunicación entre agentes y los métodos y herramientas más utilizados para el modelado de agentes. Finalmente se expondrán las plataformas Multi-Agente más importantes, haciendo especial hincapié en JADE [6], al tratarse de la plataforma de Software Libre más extendida en la actualidad y que utilizaremos en el desarrollo de este proyecto.

El capítulo 3 está dedicado al diseño de la arquitectura Multi-Agente, donde se presentará la plataforma a implementar, el papel que desempeña cada agente dentro de la plataforma, los principales protocolos de comunicación entre agentes usados, así como la ontología necesaria para el paso de mensajes dentro de la plataforma.

El capítulo 4 recoge el proceso completo de desarrollo del proyecto. En primer lugar, se explicará brevemente el proyecto a desarrollar. A continuación, dado que estamos ante un proyecto de Ingeniería del Software, realizaremos todas las fases necesarias para el desarrollo de software (requerimientos, análisis, diseño e implementación) teniéndose en cuenta los matices propios de la Ingeniería del Software (IS) basada en agentes. En el apartado de implementación se recogerán las tecnologías empleadas, así como los lenguajes de programación y el tipo de arquitectura para el desarrollo del proyecto.

Una vez expuesto el desarrollo del proyecto, llegamos al quinto y último capítulo, dedicado

a las conclusiones generales derivadas del desarrollo del mismo.

La sección final de esta memoria contiene un anexo dedicado a los procesos de consenso, donde se expondrá brevemente el estado del arte de la TDG y describiremos el modelo de consenso que instanciaremos para la puesta en marcha de la plataforma Multi-Agente. Además, se incluye una guía de configuración de instalación, el manual del usuario administrador y el manual del usuario experto.

## Capítulo 2

# Sistemas Multi-Agente

En este capítulo de la memoria del proyecto, realizaremos un breve estudio de los Sistemas Multi-Agente (SMA). A lo largo de los siguientes apartados mostraremos una clasificación de agentes bajo distintos puntos de vista, veremos qué arquitecturas existen para la creación de SMA, centrándonos en las de los dos estándares de desarrollo más utilizados: FIPA y RETSINA. A continuación revisaremos cómo se realiza la comunicación entre agentes, describiendo brevemente los estándares ACL y KQML y distintos protocolos de comunicación. Para finalizar, los dos últimos apartados harán referencia a las metodologías de modelado de agentes, donde nos centraremos en AUML, y la implementación de SMA, que contendrá los lenguajes de programación utilizados para implementación de agentes y las principales plataformas de SMA, haciendo especial hincapié en JADE, por ser, como indicamos anteriormente, la plataforma elegida para el desarrollo de nuestro SMA.

## 2.1. Introducción

Los avances de la Informática con la aparición de Internet, han cambiado el concepto de multitud de tecnologías, pasando de estar de modo centralizado a modo distribuido. La Inteligencia Artificial Distribuida intenta construir un modelo del mundo poblado por entidades autónomas e inteligentes que interactúan por medio de la cooperación, la coexistencia o por

la competencia entre ellos. Estas entidades son los llamados Agentes Software.

Un Agente Software es una entidad capaz de percibir su entorno, procesar tales percepciones y responder o actuar en su entorno de manera racional y correcta, tendiendo a maximizar un resultado esperado. Una de las características más atractivas de los Agentes Software es su carácter autónomo, es decir, la capacidad para realizar las tareas que le permitan alcanzar un objetivo sin supervisión humana.

## 2.2. Clasificaciones de Agentes

Podemos clasificar a los agentes desde varias perspectivas: según características generales, según el entorno en el que trabajan, según su comportamiento, según el modo de organización y según la utilidad [142].

### 2.2.1. Clasificación según las principales propiedades de agente

A partir de las principales propiedades de los agentes, podemos realizar una clasificación en:

- **Agente reactivo:** Se caracteriza por ser capaz de interactuar con el entorno que le rodea de forma dinámica, y de responder ante eventos no esperados (comportamiento no determinista).
- **Agente proactivo:** Genera y trata de alcanzar sus propios objetivos, siendo capaz de reconocer diferentes oportunidades o alternativas de actuación y caracterizándose por tomar la iniciativa al tratar con una tarea particular.
- **Agente social:** Es el resultado de añadir al agente proactivo la habilidad de comunicarse y colaborar con otros agentes.

Esta clasificación es una visión tradicional de los tipos de agentes que existen. Con el avance de esta tecnología han ido surgiendo nuevas clasificaciones que se revisarán en los próximos apartados.

### 2.2.2. Clasificación según sus características individuales

Podemos distinguir entre los agentes reactivos y los agentes cognitivos.

- *Agentes reactivos*: Son agentes que realizan tareas sencillas, basando su modelo computacional en un ciclo de percepción/reacción [129]. El agente reactivo recibe una acción (percepción) de su entorno y en base a ella realiza otra acción (reacción) que puede consistir en cambiar su estado interno o modificar su entorno. Los agentes reactivos no realizan procesos de razonamiento y carecen de mecanismos de representación de conocimiento. Un ejemplo gráfico del comportamiento de este tipo de agentes lo podemos encontrar en la figura 2.1

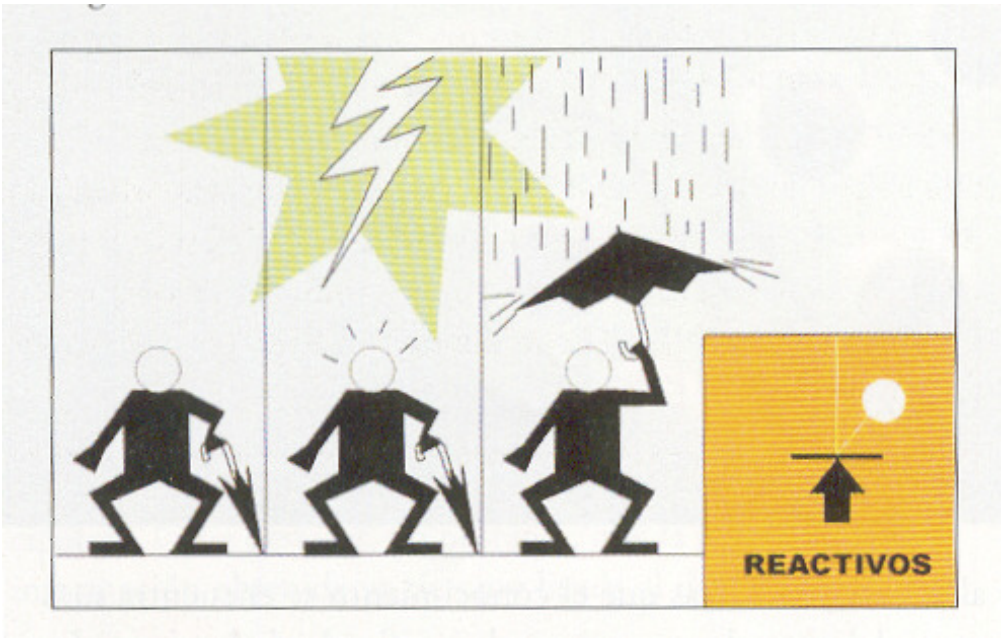


Figura 2.1: Agentes reactivos

- *Agentes cognitivos*: Estos agentes realizan tareas complejas, utilizando una representación simbólica del conocimiento sobre su entorno [44, 145]. Para cumplir su objetivo, deben realizar procesos de razonamiento, planificación y aprendizaje. Su modelo computacional está basado en un ciclo de percepción de eventos, asimilación, razonamiento y actuación [65].

### 2.2.3. Clasificación según el entorno en el que funcionan

Podemos entender el entorno del agente como la infraestructura computacional que le rodea y que le proporciona medios necesarios para que pueda desarrollar su ciclo de vida. Desde este punto de vista podemos considerar dos tipos de agentes:

- Agentes que requieren un entorno especial: necesitan un entorno especial para su funcionamiento. Ejemplo de estos agentes son los agentes móviles, ya que cada plataforma en la que pueden encontrarse proporciona los mecanismos para gestionar su ciclo de vida y para facilitar su desplazamiento entre nodos de una red; y los agentes propios del estándar FIPA [3].

- Agentes que se ejecutan en plataformas computacionales existentes: son agentes que se crean mediante los recursos del sistema operativo y siguen el ciclo de vida de cualquier aplicación. Se suelen implementar en lenguajes independientes de la plataforma empleada, como Java [8].

### 2.2.4. Clasificación según el modo de organización

Desde el punto de vista de la organización podemos considerar a los agentes como entidades individuales que tiene un rol dentro de un conjunto o SMA con una finalidad común. Para llevar a cabo su finalidad se requiere una estructura organizativa y unas capacidades del agente. Dependiendo del tipo se clasifican en [142, 145]:

- *Agentes individuales*: Son agentes que no tienen capacidad de cooperación y que realizan sus tareas de forma individual, sin precisar de la colaboración de otros agentes. La Figura 2.2 muestra un ejemplo de actuación de un agente individual.

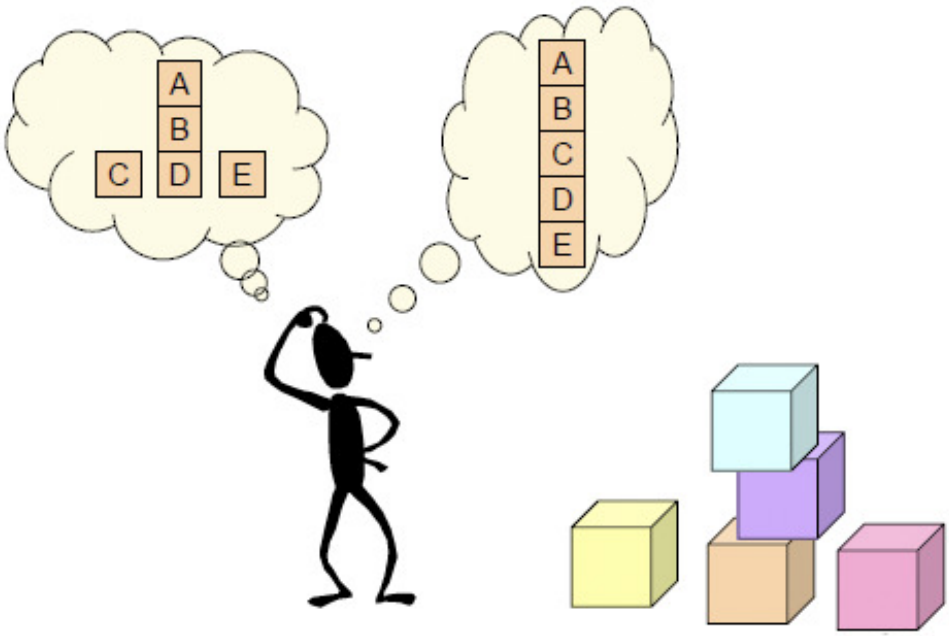


Figura 2.2: Agentes individuales

- *Agentes cooperativos*: Pueden realizar tareas solos o en colaboración con otros agentes, como se muestra en la figura 2.3. Suelen formar parte de organizaciones de agentes. En este ámbito pueden clasificarse según la representación del modelo de cooperación, el rol del agente en la organización y el modo de resolución de conflictos. Un tipo de agentes cooperativos son los agentes negociadores, lo cuales tratan de llegar a un acuerdo para que todos los agentes participantes puedan obtener un recurso que necesitan, tal y como se puede observar en la figura 2.4.

- *Agentes competitivos*: Se trata de agentes que para alcanzar su objetivo normalmente deben competir con otros agentes de su entorno por el uso de un recurso común limitado. La figura 2.5 muestra agentes de este tipo.

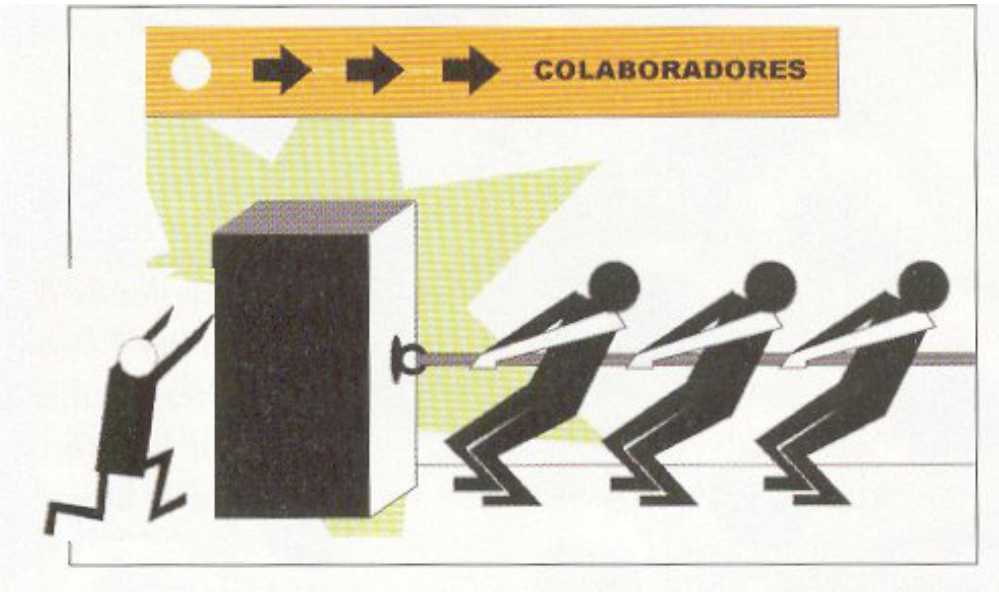


Figura 2.3: Agentes cooperativos

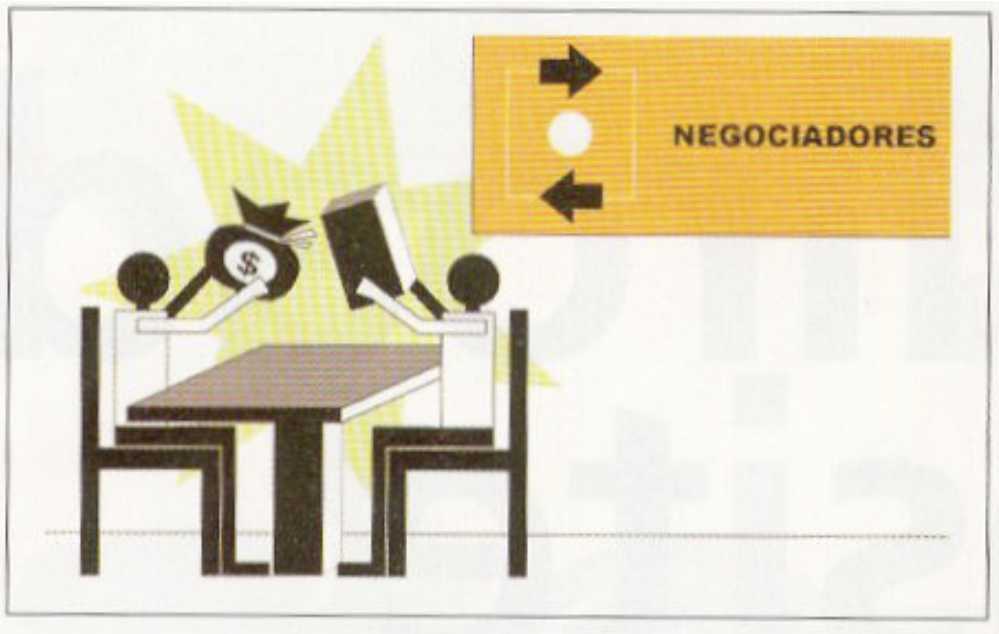


Figura 2.4: Agentes negociadores

Figura 2.5: Agentes competitivos

## 2.3. Ciclo de vida de un agente

Un agente está sujeto a un ciclo de vida en el que se definen los estados en los cuales se puede encontrar el agente, así como los cambios que se pueden realizar entre los diferentes estados, como podemos apreciar en la figura 2.6. En la plataforma que presentamos, los agentes siguen el ciclo propuesto por FIPA, es decir, cumplen con la propuesta del estándar de interoperabilidad entre agentes más aceptada, cuya arquitectura será abordada con mayor detalle posteriormente en este capítulo.

### 2.3.1. Estados de un agente

Un agente puede encontrarse en los siguientes estados:

- **Iniciado**: El objeto Agente está creado, pero todavía no se ha registrado en las páginas blancas (AMS), no tiene nombre ni dirección y tampoco se puede comunicar con otros

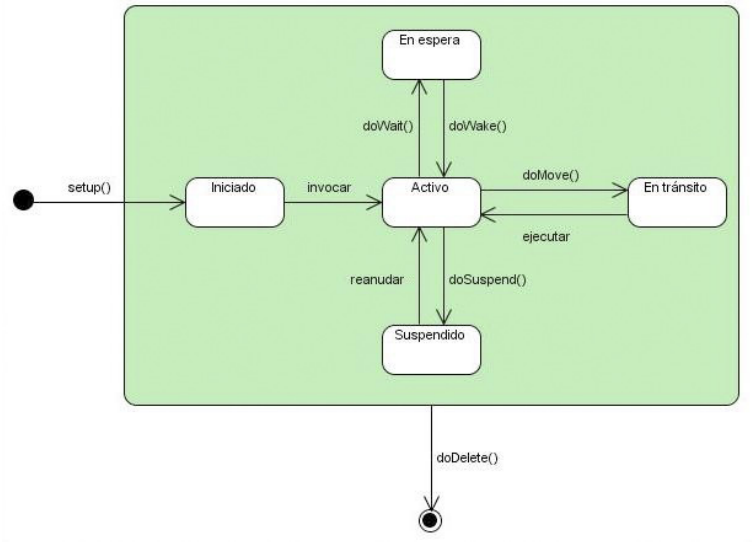


Figura 2.6: Ciclo de vida de agentes según FIPA

agentes.

- **Activo**: El Agente está registrado en el AMS, tiene un nombre, una dirección y puede acceder a todas las opciones de nuestra plataforma de negociación.

- **Suspendido**: El Agente está parado. Su hilo de ejecución está detenido y no ejecuta ningún comportamiento.

- **En espera**: El Agente está bloqueado esperando por algo. Su hilo de ejecución está dormido en un monitor de java y se despertará cuando se cumpla una cierta condición (reciba un mensaje).

- **Desconocido**: El Agente ha sido eliminado. El hilo de ejecución ha terminado y se ha eliminado del registro del AMS.

- **Tránsito**: Un Agente móvil entra en este estado mientras está migrando a una nueva localización. El sistema sigue guardando los mensajes en el buffer hasta que el agente vuelve a estar activo.

### 2.3.2. Transiciones entre estados

Un agente puede cambiar de un estado a otro a través de transiciones. Las diferentes acciones que puede llevar a cabo un agente para ello son:

- *Crear*: Creación o instalación de un nuevo agente.
- *Invocar*: Invocación de un nuevo agente. El agente pasa a estar activo.
- *Suspender*: Pone un agente en estado suspendido. Puede ser iniciado por el agente o por el AMS.
- *Reanudar*: Continúa con la ejecución de un agente que se encuentra en estado suspendido. Sólo puede ser iniciado por el AMS.
- *Esperar*: Pone un agente en estado de espera. Sólo puede ser iniciado por el agente.
- *Despertar*: Continúa con la ejecución de un agente que se encuentra en estado de espera. Sólo puede ser iniciado por el AMS.
- *Mover*: Pone un agente en otro contenedor cambiando su estado al de tránsito. Sólo puede ser iniciado por el agente.
- *Ejecutar*: Continúa con la ejecución de un agente que se encuentra en estado de tránsito. Sólo puede ser iniciado por el AMS.
- *Destruir*: La terminación normal o forzosa de un agente. Sólo puede ser iniciado por el AMS y no puede ser ignorado por el agente.

Estas transiciones de estados quedan reflejadas en la figura 2.6.

## 2.4. Definición de SMA

Una vez vistos los tipos de agentes inteligentes que existen, hemos de preguntarnos, ¿qué es un Sistema Multi-Agente? Al igual que en el caso de los agentes también se han propuesto diferentes definiciones [24]. Desde el punto de vista de la Inteligencia Artificial, un SMA se define como una red de agentes, que trabajan de forma cooperativa para encontrar una respuesta a problemas que no se pueden resolver de forma individual por cada uno de ellos [52].

Desde un punto de vista más general, en [103] se define un SMA como un sistema compuesto por múltiples componentes autónomos que poseen las siguientes propiedades:

- Cada agente tiene la capacidad de resolver el problema de manera parcial y no el problema global.
- No existe un sistema de control centralizado.
- Los datos están distribuidos.
- La ejecución es asíncrona.

Independientemente de cómo se defina un SMA, una de las características de los mismos es su modularidad, que permite que un problema complejo pueda ser resuelto a través de la participación de un conjunto de agentes, que se encargan de resolver aspectos particulares de dicho problema [102].

El motivo del creciente interés que está despertando el área de los sistemas Multi-Agente viene dado principalmente por la capacidad que tienen de realizar las siguientes tareas [108]:

1. Resolver problemas demasiado complejos para un agente centralizado.
2. Proporcionar soluciones a problemas que pueden ser resueltos a través de una sociedad de componentes autónomos que interaccionan [19].

3. Suministrar soluciones que utilicen de forma eficiente fuentes de información distribuidas espacialmente.

4. Ofrecer soluciones a entornos donde el conocimiento experto está distribuido [88].

5. Incrementar el rendimiento de las aplicaciones, aumentando la eficiencia, fiabilidad, extensibilidad, robustez, mantenimiento, rapidez, flexibilidad y reutilización de las mismas.

Aunque los SMA proporcionan una gran cantidad de ventajas, también presentan algunas dificultades [36, 108]:

1. La formulación, descripción y descomposición de problemas así como la obtención de resultados por parte de un grupo de agentes.

2. La comunicación e interacción de agentes.

3. La toma de decisiones por parte de los agentes a fin de obtener soluciones globales.

4. La representación y razonamiento de los agentes sobre acciones, planes y conocimiento de otros agentes con los cuales se coordinan.

5. El tratamiento y resolución de los conflictos entre los agentes que se coordinan.

6. El diseño y la metodología de desarrollo de SMA.

## 2.5. Arquitecturas de SMA

Una vez visto el concepto de Sistema Multi-Agente, sus ventajas y sus inconvenientes, en este apartado presentaremos los tipos de arquitecturas más comunes para la construcción de agentes. También nos centraremos en las dos arquitecturas más utilizadas: FIPA [3] y RETSINA [73, 135].

### 2.5.1. Tipos de Arquitecturas de SMA

La clasificación más común de arquitecturas de SMA diferencia entre arquitecturas deliberativas, reactivas e híbridas.

#### a) Arquitecturas Deliberativas

Las arquitecturas deliberativas son aquellas que utilizan modelos de representación simbólica del conocimiento. Los agentes que siguen esta arquitectura parten de un estado inicial y son capaces de generar planes para alcanzar sus objetivos [106].

Un agente deliberativo ha de disponer de un modelo simbólico del mundo, representado de manera explícita, para realizar un razonamiento lógico a partir de él y tomar decisiones oportunas. La principal ventaja de esta arquitectura es que el conocimiento es más fácil de entender y codificar por el ser humano. Las desventajas que presenta es la dificultad para traducir el mundo real a un modelo simbólico y preciso, y el elevado coste temporal, necesario a veces para trabajar con este modelo [31].

Sin lugar a dudas, la arquitectura deliberativa más estudiada y posiblemente más extendida es la arquitectura BDI (Belief, Desire, Intention) [113, 123, 124].

#### Arquitectura BDI

La arquitectura BDI, cuyos acrónimos en español significan *Creencia*, *Deseo* e *Intención*, combina un modelo filosófico del razonamiento humano fácil de comprender, un número considerable de implementaciones y una semántica lógica abstracta y elegante [39, 40, 84, 97, 110, 115]. Desde su establecimiento en la década de los 80, prácticamente ha permanecido inalterable. Los tres componentes básicos de esta arquitectura son las creencias, los deseos u objetivos, y las intenciones o planes.

Las ***creencias*** representan el conocimiento que los agentes tienen del entorno. En términos computacionales, son la forma de representar el estado del mundo, ya sea mediante el valor de una variable, una base de datos relacional o expresiones simbólicas. Dado que las creencias pueden representar información imperfecta del mundo, su semántica subyacente debe obedecer a una lógica de creencias, aunque la representación computacional no necesite ser puramente lógica o simbólica.

Un ***deseo*** (objetivo) representa un estado final deseado. En términos informáticos, un objetivo para un agente será alcanzar cierto valor en una variable, un registro, o cumplir una expresión simbólica representada en alguna formalización lógica.

Con creencias y objetivos no es suficiente para poder modelar un sistema capaz de operar en entornos dinámicos e inciertos: - *si hemos decidido sobre un plan y el mundo cambia ligeramente, deberíamos cuestionarnos si seguir con el plan o replantearlo* -. La respuesta es que el sistema necesita acometer los planes y sub-objetivos que planifica, pero además debe ser capaz de reconsiderarlos en momentos cruciales. Estos planes acometidos o formas de proceder se llaman ***intenciones*** y representan el tercer componente de la arquitectura BDI. Son un subconjunto de los deseos: aquellos que el agente se ha propuesto alcanzar en un momento dado, y que caracterizan el *estado mental* actual de dicho agente.

Para determinar el grado de persistencia de una intención (su resistencia a ser cambiada ante percepciones de cambio en el mundo), se han definido tres estrategias de dedicación:

- Dedicación total (a ciegas): El agente seguirá con su intención actual hasta que crea haberla alcanzado.

- Dedicación firme: Se mantendrán las intenciones del agente mientras crea que tiene opciones de alcanzarlas.

- Dedicación abierta: El agente seguirá la intención solamente mientras la considere viable.

### b) Arquitecturas Reactivas

Las arquitecturas reactivas implementan la forma de actuar de los agentes como un mecanismo de correspondencia directa *percepción-reacción*, tal como se mostró en la figura 2.1, y se basan en mecanismos de respuesta ante estímulos. A diferencia de las arquitecturas deliberativas, carecen de modelo simbólico y por tanto no emplean ningún mecanismo de razonamiento complejo. Por contra, presentan el inconveniente de que el comportamiento del agente es fijo y este no puede aprender y razonar a partir de la información que recibe. La arquitectura reactiva más conocida es la llamada Arquitectura de Subsunción [116]. Esta arquitectura define una serie de capas conectadas a sensores que transmiten información en tiempo real, de forma que las capas componen una jerarquía de tareas en la que los niveles inferiores tienen menos control sobre los niveles superiores. La figura 2.7 muestra el esquema básico de la arquitectura de subsunción.

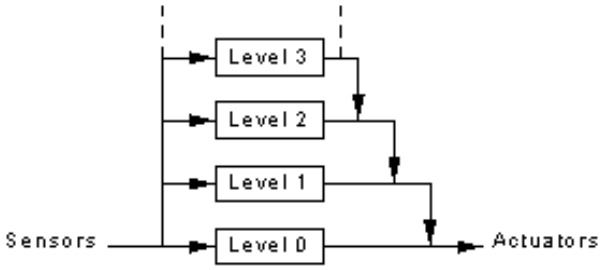


Figura 2.7: Esquema básico de la arquitectura de subsunción

### c) Arquitecturas Híbridas

Para intentar solventar las limitaciones de las arquitecturas deliberativas y reactivas, se han propuesto arquitecturas híbridas, que combinan aspectos de ambos modelos. Los agentes en arquitecturas híbridas presentan dos subsistemas: uno deliberativo, que utiliza un modelo simbólico para generar planes y otro reactivo, centrado en reaccionar ante eventos en el entorno.

La figura 2.8 muestra el esquema básico de las arquitecturas híbridas, las cuales se suelen estructurar por capas. Así, la forma de estructurarse puede ser [69, 114]:

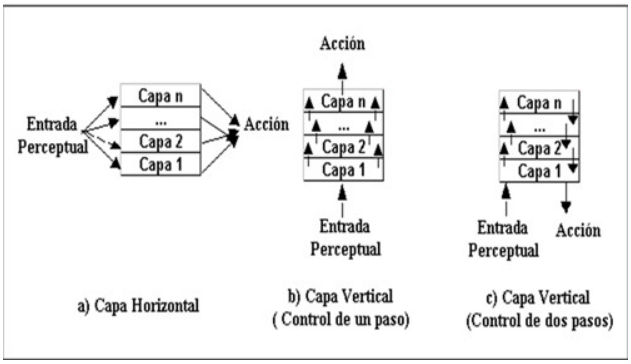


Figura 2.8: Diferentes esquemas de arquitectura híbrida

- *Vertical*: Una única capa tiene acceso a los sensores y actuadores.
- *Horizontal*: Todas las capas tienen acceso a los sensores y actuadores.

Al igual que en las arquitecturas de subsunción, las capas se disponen jerárquicamente, brindando información sobre el entorno a diferentes niveles de abstracción. Los tres niveles básicos presentes en la mayoría de arquitecturas son:

- *Nivel reactivo*: Es el nivel más bajo y en él se toman las decisiones acerca de cómo responder ante los estímulos recibidos en tiempo real. Normalmente se basa en arquitecturas de subsunción.
- *Nivel de conocimiento:* Basado en el conocimiento que el agente posee sobre el medio. En este nivel suele emplearse una representación simbólica del mundo.
- *Nivel social*: Es la capa de más alto nivel y se encarga de aspectos sociales del entorno, incluyendo información de otros agentes, deseos, intenciones, etc.

### 2.5.2. Arquitecturas estandarizadas de SMA

En los primeros años de desarrollo de SMA, cada sistema implementaba una arquitectura específica. Esto presentaba problemas de *interoperabilidad*, que es la facilidad de conexión e integración de un SMA, y de *apertura*, o posibilidad de extensión. Por este motivo es importante disponer de estándares de desarrollo.

En los siguientes apartados veremos la arquitectura FIPA (*Foundation for Intelligent Physical Agents*) [3] que está basada en el estándar FIPA y la arquitectura RETSINA (*Reusable Environment for Task-Structured Intelligent Networked Agents*) [73, 135], capaz de hacer interaccionar agentes heterogéneos.

#### a) Arquitectura FIPA

La arquitectura seguida en FIPA define un modelo para la administración de agentes en un contexto en el que los agentes FIPA pueden existir, operar y ser gestionados, estableciendo así una referencia lógica para la creación, registro, localización, comunicación, migración y operaciones entre agentes. El modelo de referencia FIPA consta de una serie de componentes mostrados en la figura 2.9, y que pasamos a describir a continuación:

- **Plataforma de Agentes:** Es la infraestructura en la que se establecen y utilizan los agentes. El diseño interno de esta plataforma se deja en manos de los desarrolladores y no forma parte del estándar FIPA más allá de lo aquí expuesto. Dado que una misma plataforma de agentes puede encontrarse distribuida en varios ordenadores, los agentes que residen en la misma no tienen por qué encontrarse en el mismo equipo.

- **Agente:** Un agente FIPA se considera un proceso computacional que reside en la plataforma de agentes y normalmente ofrece uno o más servicios, cada uno de los cuales se publica junto a una descripción del mismo.

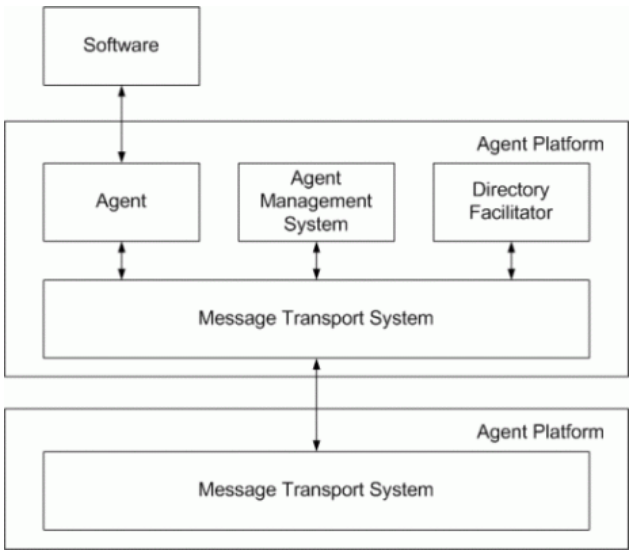


Figura 2.9: Modelo de referencia del estándar FIPA

- **Facilitador de Directorio (DF)**: Este componente opcional de la plataforma de agentes proporciona un servicio de *páginas amarillas* al resto de agentes. El Facilitador de Directorio mantiene una lista completa y precisa de los agentes existentes y proporciona información actualizada sobre los servicios que estos ofrecen. Se trata de un agente en sí mismo y único para cada plataforma de agentes.

- **Sistema de Gestión de Agentes (AMS)**: Es el elemento de gestión principal, que conoce en todo momento el estado de la plataforma y los agentes que pertenecen a ella. Algunos de los servicios que ofrece son la creación, eliminación y control de los cambios de estado entre agentes, supervisión para el registro de nuevos agentes en la plataforma y gestión de los recursos y canales de comunicación. Al igual que el Facilitador de Directorio, es único para cada plataforma, siendo un agente en sí mismo.

- **Sistema de Transporte de Mensajes (MTS)**: Servicio proporcionado por la plataforma de agentes para transportar mensajes FIPA-ACL (FIPA Agent Communication Language) entre agentes, ya sea dentro de la misma plataforma o entre plataformas diferentes.

En nuestro caso, ésta será la arquitectura elegida para el desarrollo del proyecto.

## b) Arquitectura RETSINA

La arquitectura RETSINA (Reusable Task Structure-based Intelligent Network Agents) nace como arquitectura capaz de proveer un marco de desarrollo de comunicación entre agentes, de manera que agentes heterogéneos puedan cooperar entre sí.

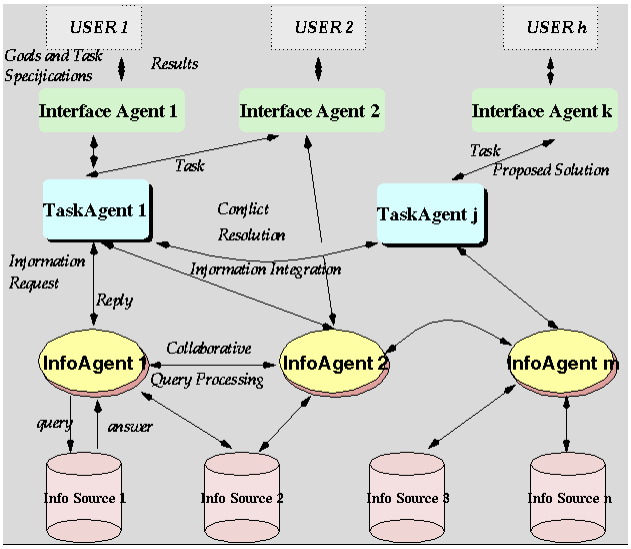


Figura 2.10: Arquitectura RETSINA

RETSINA consta de tres tipos básicos de agentes, como podemos apreciar en la figura 2.10:

- **Agentes de interfaz:** interactivos con los usuarios, reciben los datos de entrada del usuario y muestran dichos datos. Ayudan a emparejar agentes que solicitan servicios con aquéllos que los proveen.
- **Agentes de tarea:** ayudan a los usuarios a realizar tareas, planificando y realizando estos planes coordinando e intercambiando la información con otros agentes software.

- **Agentes de la información:** proporcionan acceso inteligente a una colección heterogénea de fuentes de información.

Cada agente RETSINA tiene cuatro módulos reutilizables para comunicar, planificar, programar, y supervisar la ejecución de tareas y peticiones de otros agentes.

- *Módulo de comunicación y coordinación*: acepta e interpreta mensajes y peticiones de otros agentes.
- *Módulo de planificación*: toma como entrada un conjunto de objetivos y produce un plan que satisfaga los mismos.
- *Módulo de programación*: utiliza la estructura de la tarea creada por el módulo de planificación para pedir las tareas.
- *Módulo de ejecución*: supervisa este proceso y se asegura que las acciones están realizadas.

RETSINA incluye las capacidades y las funciones de los componentes, agentes y módulos del agente, tal y como se muestran en la figura 2.11

## 2.6. Comunicación entre agentes

Para poder resolver un problema de manera distribuida entre un grupo de agentes, es necesaria la coordinación de la actividad de éstos y, en algunos casos, la cooperación entre algunos de ellos. Para que esto sea posible es necesario establecer mecanismos de comunicación entre los agentes.

Cuando se comenzó a utilizar la tecnología Multi-Agente, cada SMA que se desarrollaba hacía uso de sus propias soluciones comunicativas. Dado que este método impedía la comunicación entre agentes de distintos sistemas, aparecieron una serie de iniciativas que han

| **Infraestructura de RETSINA MAS** | **Infraestructura individual del agente en RETSINA** |
|---|---|
| **Interoperación MAS**<br>Retsina-oaa Interoperator | |
| **Capacidad de trazado del agente**<br>Matchmaker | **Capacidad de trazado del agente**<br>Módulo de Matchmaker |
| **Nominación de localización de las trazas**<br>American National Standard | **Nominación de localización de las trazas**<br>Módulo de la American National Standard |
| **Seguridad**<br>Certificado de los servicios de Autoridad/Encriptado | **Servicios SECURITY**<br>Módulo Security/ Claves privadas/públicas |
| **Funcionamientos de los servicios**<br>Supervisión De la Falta | **Funcionamientos de los servicios**<br>Self-Monitory/Cloning |
| **Gerencia de los servicios MAS**<br>Carga/Actividad visualizada/Lanzado | **Servicios de Gerencia**<br>Módulo Del Carga |
| **Infraestructura del Acl**<br>Servidores Públicos de Ontología/Protocolos | **Infraestructura del Acl**<br>Acl Parser / Ontología privada / Protocolo de la máquina |
| **Infraestructura De la Comunicación**<br>Transferencia de Discovery/Message | **Módulos De la Comunicación**<br>Módulo Discovery/ Comunicador RETSINA |
| **Entornos de Funcionamiento**<br>Máquinas, OS, Capa De Transporte De Multicast de la Red: TCP/IP, Radio, Infrarrojo, SSL | |

Figura 2.11: Niveles de la Arquitectura RETSINA

tratado de especificar una serie de estándares de modo que se consiga una interacción real y homogénea entre distintos SMA.

En este apartado vamos a ver los distintos actos comunicativos que se llevan a cabo mediante agentes, los estándares que permiten realizar comunicación de agentes y por último los protocolos de comunicación, todo ello según el estándar FIPA.

### 2.6.1. Actos comunicativos

Los actos comunicativos básicos recogidos en el estándar FIPA, concretamente en FIPA CAL (Communicative Act Library) permiten clasificar las acciones comunicativas en cuatro clases, en función de la actividad social a la que se encuentran destinadas:

- Información.
- Realización.
- Negociación.
- Intermediación.

En los siguientes apartados pasamos a desarrollar las distintas acciones.

#### 2.6.1.1. Información

En estas acciones participan dos agentes, el emisor y el receptor. El emisor puede realizar dos tipos de acciones: ofrecer información (ROL 1) o solicitar información (ROL 2).

Cuando el emisor ofrece información se distinguen las siguientes acciones:

- *inform ($\phi$)*: el emisor informa al receptor que la proposición $\phi$ es verdadera.

- *inform-if ($\phi$)*: es una macro acción en la que el emisor pide al receptor que le informe (con un *inform*) si la proposición $\phi$ es cierta o falsa de acuerdo a sus creencias. Si no las tiene enviará un *refuse*.

- *inform-ref(Ref x $\delta(x)$)*: el contenido de la acción es una expresión referencial. Es una macro acción en la que el emisor pide al receptor que le informe del objeto o conjunto de objetos cuya descripción coincida con la expresión referencial $x\delta(x)$ (si no, *refuse*).

- *confirm($\phi$)*: el emisor informa al receptor que la proposición $\phi$ es cierta, sabiendo que el receptor tiene incertidumbre acerca de la certeza de $\phi$.

- *disconfirm($\phi$)*: el emisor informa al receptor que la proposición $\phi$ es falsa, sabiendo que el receptor tiene incertidumbre acerca de la certeza de $\phi$.

- *not-understood(a,$\phi$)*: el emisor informa al receptor que la acción $a$ ha sido percibida pero no comprendida. Un caso muy frecuente es que el emisor no haya comprendido un mensaje que acaba de recibir. El segundo elemento $\phi$ es una proposición que representa la razón por la que no se ha entendido el mensaje. Obviamente, no está garantizado que esta razón esté representada de forma que el otro agente la entienda.

Cuando el emisor solicita información, podemos distinguir las siguientes acciones:

- *query-if($\phi$)*: el emisor pide al receptor que le comunique (mediante un *inform*) si la proposición $\phi$ es cierta.

- *query-ref(Ref x $\delta(x)$)*: el emisor pide al receptor que le comunique (mediante un *inform*) todo el conjunto de objetos cuya descripción coincida con la expresión referencial $x\,\delta(x)$.

- *suscribe(Ref x $\delta(x)$)*: es una versión persistente de *query-ref*. El agente receptor informará (mediante un *inform*) al emisor del valor de la expresión de referencia $x\,\delta(x)$; y continuará enviando más actos *inform cada vez que dicho valor cambie*.

#### 2.6.1.2. Realización

Las acciones de realización son aquellas en las que el emisor desempeña el ROL 1 si el agente está interesado en la realización de la acción y el ROL 2 si es el agente que realizará la acción. En el primer caso podemos distinguir las siguientes acciones:

- *request(a)*: el emisor requiere que el receptor lleve a cabo alguna acción $a$. El contenido del mensaje es una descripción de la acción que ha de ser llevada a cabo.

- *request-when* $(< j; act >, \phi)$: el emisor quiere que el receptor $j$ lleve a cabo alguna acción *act*, cuando $j$ crea que la proposición $\phi$ es verdadera.

- *request-whenever* $(< j; act > \phi)$: el emisor quiere que el receptor $j$ lleve a cabo alguna acción *act*, tan pronto como $j$ crea que la proposición $\phi$ es verdadera. Además, si en algún momento $\phi$ llega a ser falsa, la acción será repetida por $j$ cada vez que $\phi$ llegue a ser verdadera.

Cuando el emisor desempeña el ROL 2, diferenciamos las siguientes acciones:

- *agree*$(< i; act >, \phi)$: el emisor comunica al receptor $i$ que está de acuerdo en llevar a cabo la acción *act* en un futuro, cuando la proposición $\phi$ sea cierta.

- *refuse*$(< i; act >, \phi)$: el emisor comunica al receptor $i$ que la acción *act* no va a ser llevada a cabo; la proposición $\phi$ es la razón de la negación.

- *cancel (a)*: esta acción permite al agente emisor cancelar o detener al agente receptor de llevar a cabo o continuar llevando a cabo alguna acción $a$ previamente requerida.

- *failure* $(a,\phi)$: esta acción permite al agente emisor comunicar al receptor que la ejecución de la acción $a$ fue intentada, pero el intento fue fallido debido a la razón $\phi$.

#### 2.6.1.3. Negociación

Para las acciones de negociación, el emisor desempeña el ROL 1 cuando es el agente interesado en la realización de la acción, que normalmente no conoce las condiciones bajo las cuales el cooperante realizará la acción solicitada. Por el contrario, realizará el ROL 2 cuando es el agente que coopera en que dicha acción se realice. En el primero de los casos se podrán dar las siguientes acciones:

- *call for proposal* $(< j; act >, Refx\delta(x))$: es una acción de propósito general para iniciar un proceso de negociación haciendo una llamada a la participación para llevar acabo una acción dada.
- *accept-proposal* $(< j; act >, \phi)$: el emisor comunica al receptor $j$ que acepta la propuesta (previamente recibida) de llevar a cabo la acción *act*, y que la llevará a cabo cuando la proposición $\phi$ sea cierta.
- *reject-proposal* $(< j; act >, \phi, y)$: la acción de rechazar una propuesta de llevar a cabo alguna acción durante la negociación. El emisor informa al receptor $j$ que, debido a la proposición $y$, la acción *act* con precondición $\phi$ no será llevada a cabo.

Cuando el emisor desempeña el ROL 2, tenemos sólo una acción:

- *propose* $(< i; act >, \phi)$: el emisor propone llevar a cabo una acción *act*, siempre y cuando la proposición $\phi$, que representa cierta precondición, sea cierta.

#### 2.6.1.4. Intermediación

En las acciones de intermediación, el emisor desempeñará el ROL 1 cuando solicita la intermediación y el ROL 2 cuando corresponde al agente intermediario, que permite hacer de puente entre el primer agente y todos los demás. Al representar el ROL 1 se distinguen las siguientes acciones:

- *propagate (Ref x $\delta(x)$, $< i; cact >, \phi$)*: el emisor quiere que el agente receptor $i$ actúe de intermediario; de modo que reciba el mensaje ACL *cact*, y además que propague dicho mensaje a los agentes denotados por la expresión referencial $x\ \delta(x)$, siempre y cuando se verifique la restricción $\phi$. Además, se pretende que los posibles receptores del envío propaguen a su vez el mensaje.

- *proxy (Ref x $\delta(x)$, $< j; cact >, \phi$)*: idéntica a la anterior con la diferencia de que el mensaje *cact* no va dirigido a $j$ (el intermediario), y además los posibles receptores del envío no han de propagar a su vez el mensaje.

### 2.6.2. Estándares de comunicación

Para que sea posible entender los mensajes que se envían los distintos agentes entre sí, es necesaria la existencia de estándares que rijan la estructura y contenido de los mensajes. Si los mensajes son demasiado complejos al tener muchos atributos, la mejor manera de comunicación es utilizando una ontología del dominio del SMA (abordaremos las ontologías con mayor detalle en el capítulo 3).

En los siguientes apartados vamos a ver el estándar que usa FIPA como estructura de mensaje ACL [117, 119] , y el estándar KQML [67, 130, 141].

#### 2.6.2.1. ACL

Dentro de la recomendación FIPA encontramos la definición del lenguaje de Comunicación entre Agentes, ACL (Agent Communication Language). ACL define la arquitectura que tienen los mensajes que envían los agentes. Dentro de un mensaje FIPA ACL [22, 86, 118] puede haber uno o más elementos de mensaje (Figura 2.12), siendo obligatorio en todos los mensajes el tipo de performativa (*performative*). Este atributo denota el tipo de acto comunicativo del mensaje.

| Elemento de mensaje | Categoría de elementos |
|---|---|
| Performative | Tipo de acto comunicativo |
| Sender | Participante de la comunicación |
| Receiver | Participante de la comunicación |
| Reply-to | Participante de la comunicación |
| Content | Contenido del mensaje |
| Language | Descripción del contenido |
| Encoding | Descripción del contenido |
| Ontology | Descripción del contenido |
| Protocol | Control de la conversación |
| Conversation-id | Control de la conversación |
| Reply-with | Control de la conversación |
| in-reply-to | Control de la conversación |
| Reply-by | Control de la conversación |

Figura 2.12: Elementos de un mensaje FIPA-ACL

Dependiendo del tipo de acto comunicativo, el contenido del mensaje será distinto, ya que cada uno incluye información específica de dicho acto. Esto se incluirá en el elemento *content* de ACL. En la figura 2.12 se muestran los distintos elementos que pueden estar presentes en un mensaje ACL.

#### 2.6.2.2. KQML

Es un lenguaje de comunicación y protocolo, orientado a mensajes, para el intercambio de información. Es independiente de protocolos de transporte (TCP/IP, HTTP,...), sintaxis del contexto, ontologías y protocolos de alto nivel (contract net,...).

Los mensajes KQML comunican una actitud sobre el contenido que llevan (aserción, solicitud, pregunta). A las primitivas del lenguaje las denominamos performativas pudiendo ser de distinto tipo (inform, query,...). Cada mensaje representa un acto de habla y puede

contener los elementos que hay en la figura 2.13.

| Elemento de mensaje | Descripción |
|---|---|
| :sender | Nombre del emisor del mensaje |
| :receiver | Nombre del receptor del mensaje |
| :in-reply-to | Etiqueta para la respuesta a un mensaje anterior |
| :reply-with | Si el emisor espera respuesta, la etiqueta para dicha respuesta |
| :language | Nombre del lenguaje de representación del parámetro :content |
| :ontology | Nombre de la ontología usada en el parámetro :content |
| :content | Información sobre la que la performativa expresa una actitud |

Figura 2.13: Elementos de un mensaje KQML

### 2.6.3. Protocolos de comunicación

Definimos un protocolo de comunicación como un conjunto de reglas estándar que dominan la sintaxis, semántica y sincronización de la comunicación y que controlan o permiten la conexión, comunicación y transferencia de datos entre dos puntos finales.

En este apartado veremos los distintos protocolos que existen para poder comunicar agentes dentro de un SMA. En concreto se describirán los protocolos: subscribe, request, request-when, contract-net, iterated contract-net, protocolos de subastas, brokering y recruiting, los cuales cumplen con el estándar FIPA.

#### a) Subscribe

El agente que inicia el protocolo indica mediante el acto comunicativo *subscribe* que desea ser notificado siempre que se cumpla cierta condición. El participante podrá indicar que la rechaza o mandar directamente el primer mensaje informativo (previamente el participante puede enviar de forma optativa el mensaje *agree*). En este último caso, podrá seguir

mandando todo en una secuencia de mensajes de la clase *inform*. En cualquier momento la suscripción podrá terminar cuando el agente participante experimente un fallo, hecho que comunicará al iniciador con un mensaje *failure*. Esta comunicación se describe en la figura 2.14.

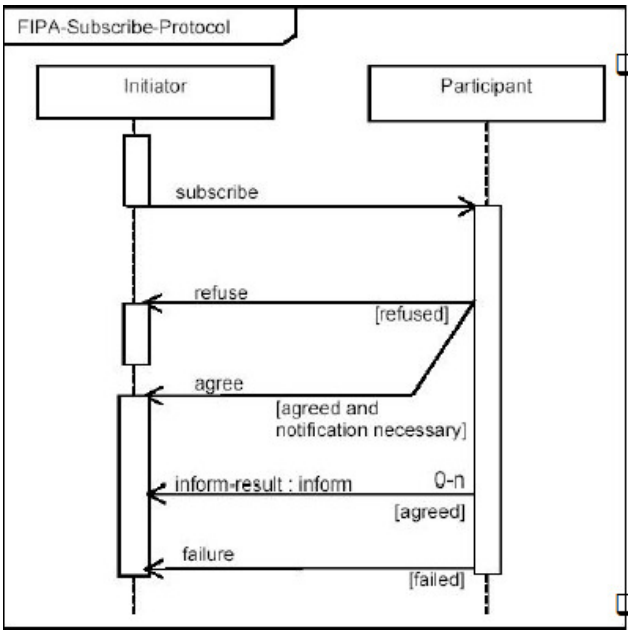


Figura 2.14: Protocolo Subscribe

### b) Request

En el protocolo *Request*, descrito en la figura 2.15, el agente iniciador del protocolo pide al otro participante que realice una acción (la semántica del acto de habla *request* se refiere a acciones). En primer lugar, el participante indica mediante un mensaje si está dispuesto a atender la petición y, en caso de que sea así, posteriormente manda otro mensaje informando del éxito de la acción. Sintácticamente, la construcción [*agreed*] es una condición que indica que sólo se manda el mensaje complejo correspondiente, si anteriormente se ha contestado con el mensaje *agree*.

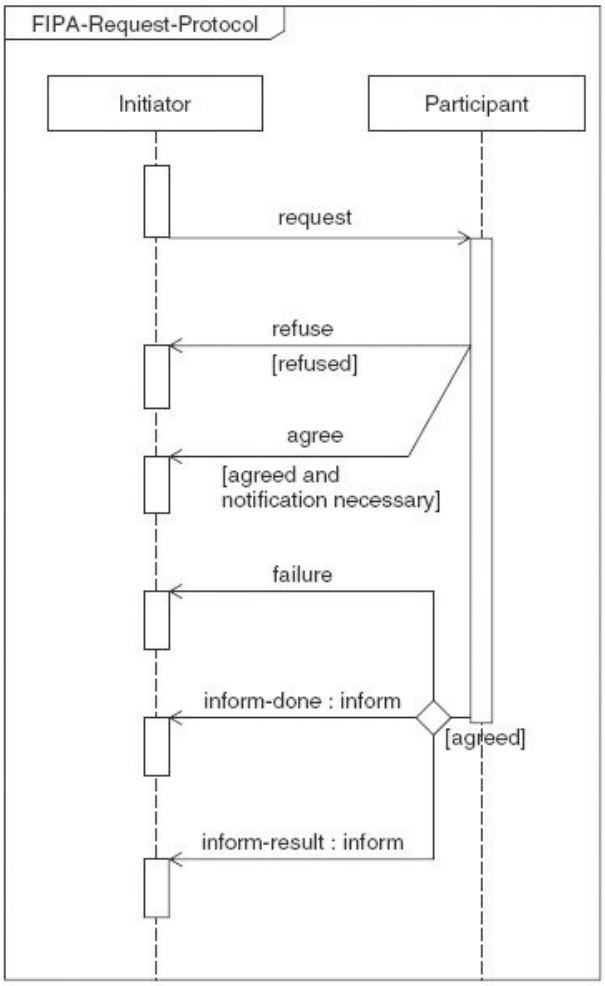


Figura 2.15: Protocolo Request

### c) Request-when

Como se puede observar en la figura 2.16, este protocolo es similar al *Request*, puesto que el agente que inicia el protocolo pide al otro participante que realice una acción. Sin embargo, al aplicar el acto de habla *request-when*, el participante sólo deberá realizar la acción en cuanto se cumpla cierta precondición. Si el participante indica mediante un mensaje que está dispuesto a atender la petición, se supone que se compromete a realizar la acción nada más cumplirse la precondición. Por eso, eventualmente realizará la acción e informará de su

resultado, o bien se retirará su anterior compromiso mediante un mensaje de clase *refuse*.

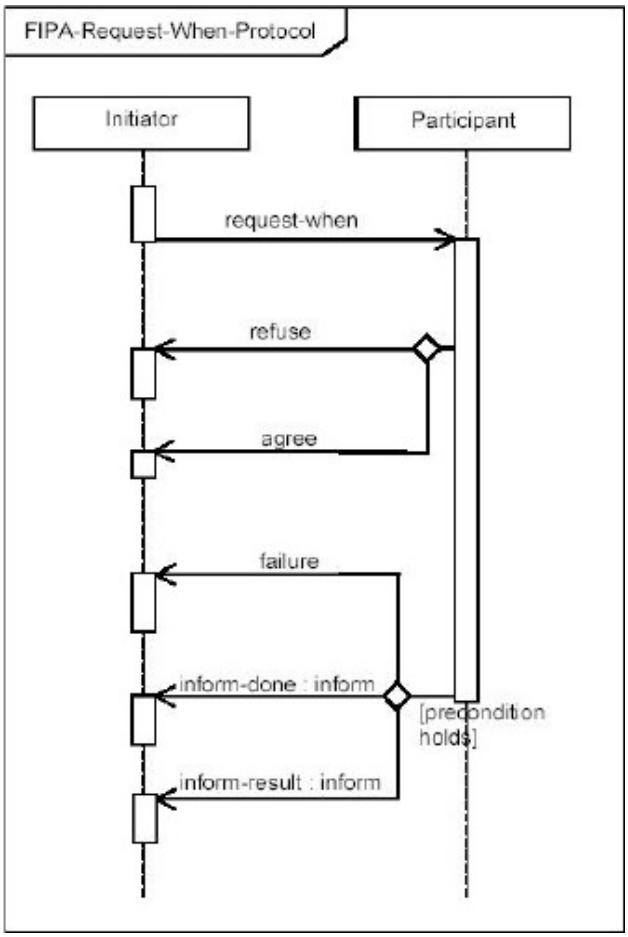


Figura 2.16: Protocolo Request-when

### d) Contract Net

En este protocolo, el agente iniciador especifica las características de unas tareas que desea ver realizadas mediante el acto de habla *call for propose (cfp)*, al cual los participantes tendrán que notificar su disposición antes de un *deadline*. Posteriormente, el agente inicial delega las tareas en uno o más agentes que han contestado positivamente, los cuales finalmente informan del resultado de las tareas delegadas en ellos.

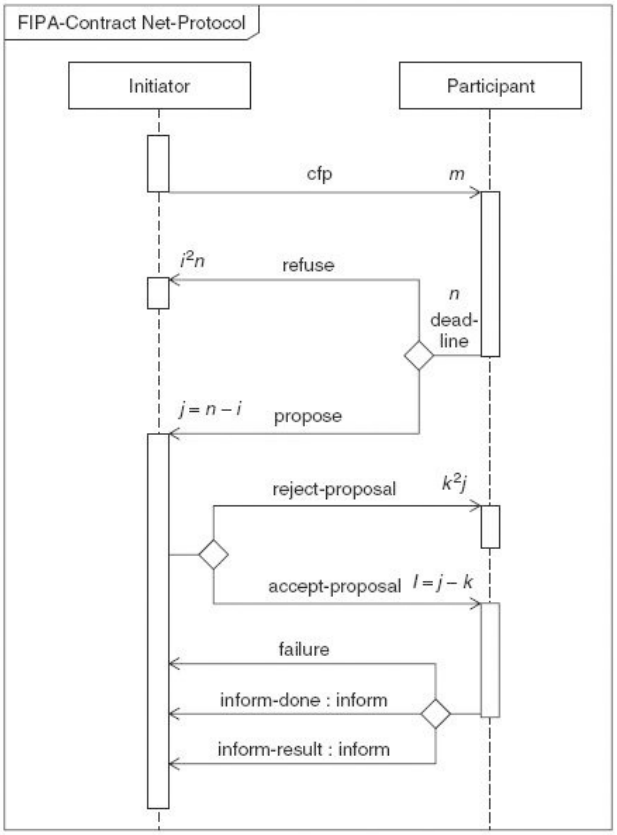


Figura 2.17: Protocolo Contract Net

### e) Iterated Contract Net

Amplía el protocolo *Contract Net*, pues permite varias rondas de contrataciones. Al igual que en el *Contract Net* simple, un agente inicia el protocolo especificando las características de unas tareas que desea ver realizadas mediante el acto de habla *cfp*, al cual los participantes tendrán que notificar su disposición antes de un plazo límite. Sin embargo, aparte de rechazar o aceptar las ofertas, también puede optar por volver a anunciar la tarea (posiblemente con características modificadas). Con tal fin, primeramente rechaza la oferta anterior, y seguidamente envía un nuevo *cfp*. En este caso han de realizarse las dos acciones expresándose mediante un paralelismo *and*, denotado por una línea gruesa (sin rombo). El orden de los mensajes se indica en el *participant deadline*, añadiendo números a las etiquetas de las flechas

correspondientes. Esta descripción queda reflejada en la figura 2.18

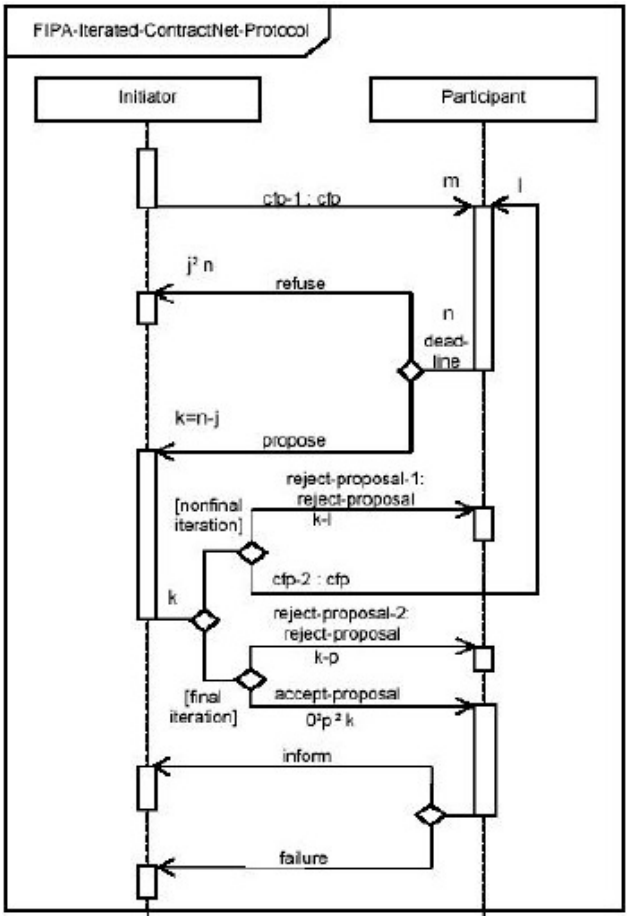


Figura 2.18: Protocolo Iterated Contract Net

### f) Protocolos de Subastas

En este caso, existen dos protocolos que permiten modelar los patrones de interacción en subastas inglesas y holandesas. Ambos protocolos necesitan la figura de un iniciador que es quien da comienzo a la subasta, invitando a los participantes (*call for propose*). Los participantes por su parte mandan al iniciador la cantidad de la puja (*propose*). Posteriormente el iniciador informa de la cantidad por la que va la subasta. Esto se repetirá hasta que haya un ganador de la subasta, el cual estará regulado por un máximo tiempo de espera sin recibir

más pujas o por un umbral de cantidad.

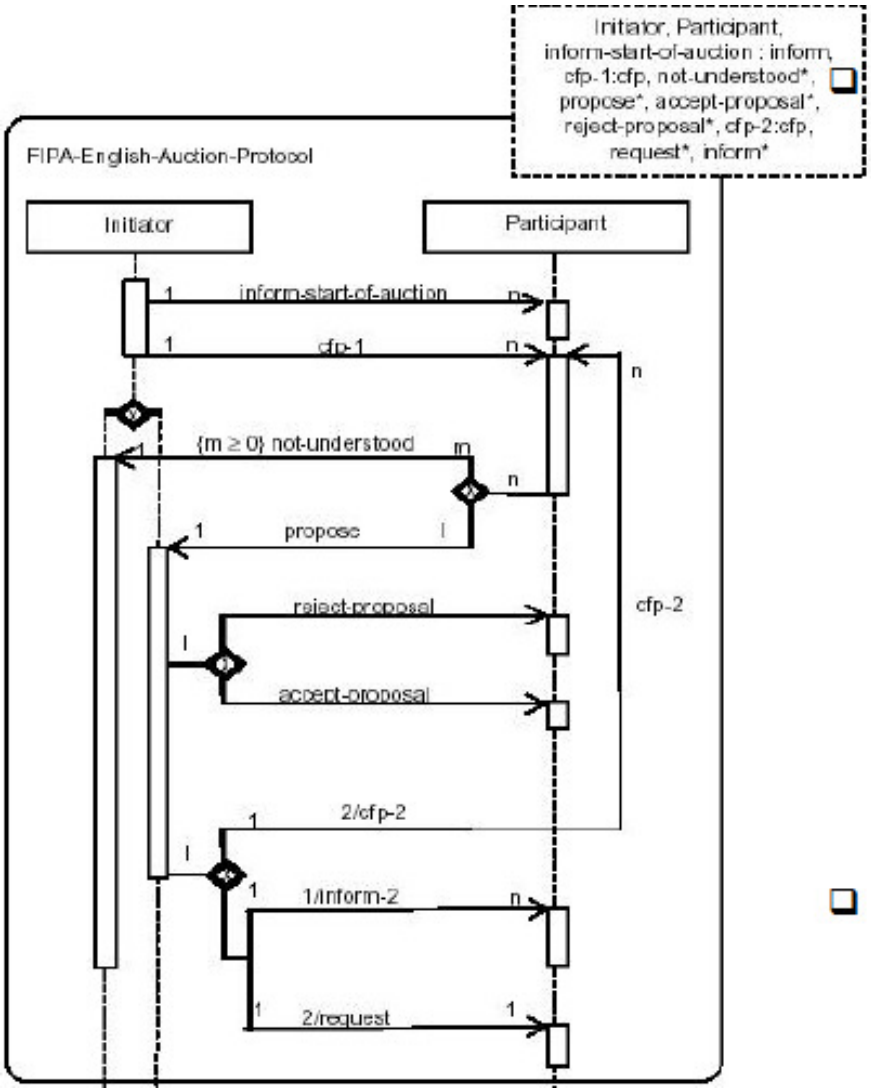


Figura 2.19: Protocolo Subasta Inglesa

La principal diferencia entre la subasta a la inglesa y a la holandesa estriba en que la primera, como podemos apreciar en la figura 2.19, se parte de un valor inicial y con las sucesivas pujas se va incrementando la cantidad de la puja. En cambio en la subasta a la holandesa, cuyo protocolo se describe en la figura 2.20, se parte de un valor de subasta inicial y se va decrementando el valor hasta que algún participante acepta el valor propuesto por

el iniciador(subastador).

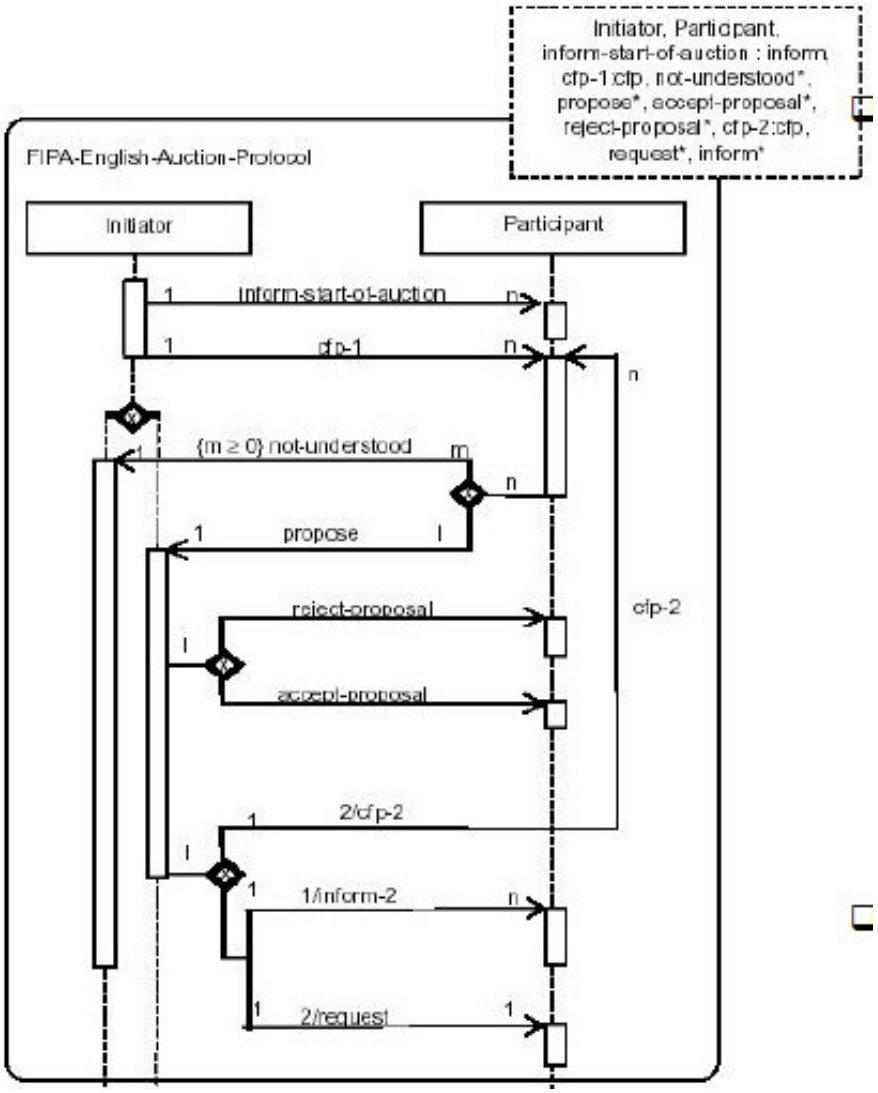


Figura 2.20: Protocolo Subasta Holandesa

### g) Brokering

El agente que inicia la interacción utiliza un intermediario (*broker*) para que traslade un acto comunicativo a un agente competente. El *broker* entonces determina un conjunto de agentes apropiados a los que envía el acto comunicativo (mediante el subprotocolo),

y comunica los resultados de este subprotocolo al agente iniciador. En este contexto, la interacción entre este último y el intermediario no es relevante, por lo que se modela mediante un subprotocolo. En la figura 2.21 queda descrito el funcionamiento del protocolo Brokering.

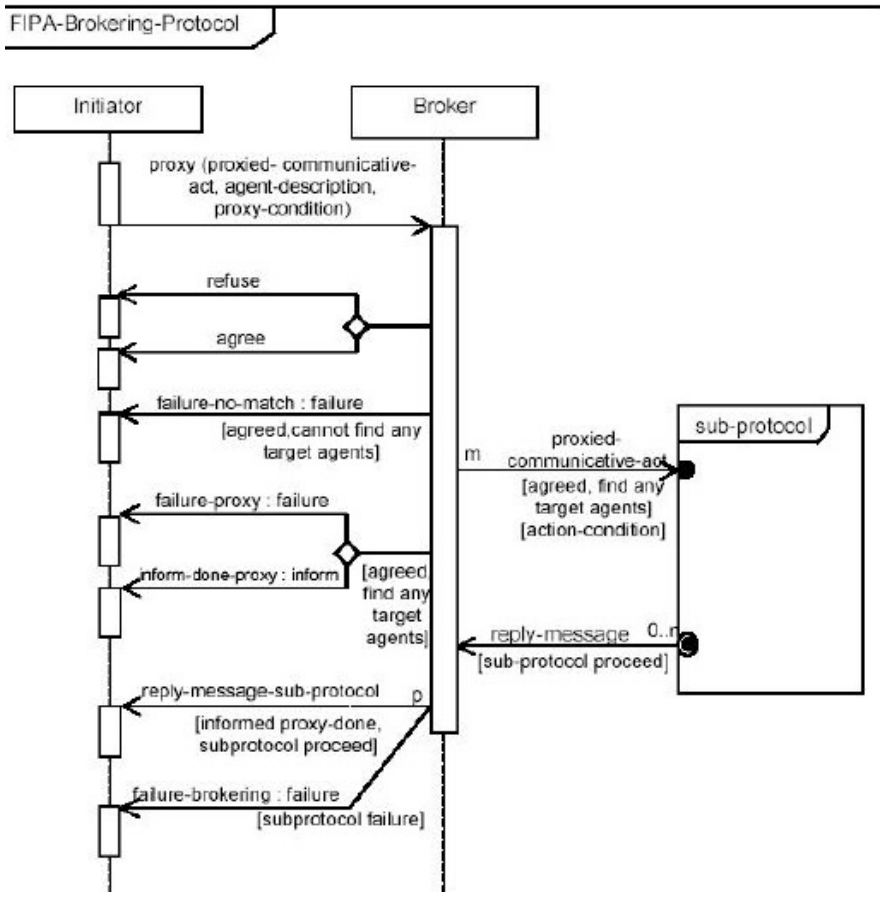


Figura 2.21: Protocolo Brokering

### h) Recruiting

Este protocolo, descrito en la figura 2.22 se diferencia del protocolo de *Brokering* en que el receptor (*destinator*) del resultado de la interacción puede no ser el agente que inició el protocolo de interacción (*initiator*).

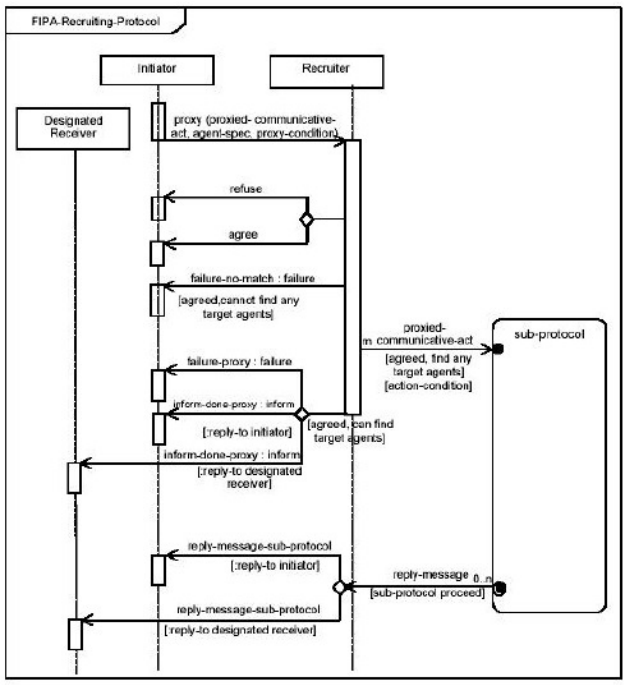


Figura 2.22: Protocolo Recruiting

## 2.7. Modelado de Agentes en SMA

Una vez visto los tipos de agentes, las arquitecturas, la comunicación y la coordinación de Sistemas Multi-Agente, en este apartado pasamos a ver distintas metodologías y los pasos a seguir para llevar a cabo el modelado de agentes inteligentes.

Antes de continuar tenemos que hacernos una pregunta, ¿cuándo necesitamos realmente un SMA para resolver un determinado problema?

Para poder dar respuesta a esta pregunta tendremos que preguntarnos antes:

- ¿Se trata de un sistema distribuido abierto?¿Pueden incorporarse nuevos tipos de entidades al sistema de manera dinámica?¿Pueden cambiar las entidades existentes?
- ¿Es necesario considerar una evolución del comportamiento independiente para cada

uno de los componentes del sistema o para una parte significativa?

- ¿Hay incertidumbre?¿Es posible para una entidad del sistema conocer su contexto suficientemente para poder decidir con certeza el efecto de las acciones que puede realizar?

- ¿Hay personalización?¿Un mismo servicio se puede ofrecer simultáneamente de manera distinta según las características de cada usuario?

- ¿Hace falta definir una organización de entidades que interactúen para resolver conjuntamente problemas globales?

Si la respuesta a varias de estas preguntas es afirmativa, entonces será conveniente concebir el sistema como una sociedad de agentes.

Las siguientes secciones tratan de explicar metodologías de modelado de Sistemas Multi-Agente, dividiéndolas desde el punto de vista del SMA o desde los roles, servicios e interacciones que presentan los agentes. Por último, veremos qué lenguaje se utiliza actualmente como unificación de notaciones de las distintas metodologías.

### 2.7.1. Medotologías desde el punto de vista del SMA

A la hora de modelar un SMA casi todas las metodologías proponen hacerlo desde varios puntos de vista complementarios, de modo que se pueda gestionar mejor su complejidad [82]. En este apartado se verán las metodologías que tienen en cuenta al SMA, para posteriormente modelar los distintos agentes y las relaciones entre éstos.

#### a) AAII/BDI

Esta metodología considera varios puntos de vista:

- *Punto de vista externo*: Su propósito es identificar agentes e interacciones, obteniéndose un modelo de agentes, donde se identifica una jerarquía de clases de agentes, y un modelo de interacciones, donde podemos identificar el conjunto de relaciones entre agentes.

- *Punto de vista interno*: Describe el comportamiento de cada uno de los agentes. Se basa fundamentalmente en el modelo BDI, descrito anteriormente en este capítulo, y se obtienen el modelo de creencias (información sobre el entorno que conoce el agente, estado interno del agente y acciones que es capaz de realizar), modelo de objetivos (el agente persigue la satisfacción de sus objetivos, que definen el estado ideal del mundo que espera el agente) y el modelo de planes (conjunto de planes que definen la estructura de control del agente).

El desarrollo del SMA está guiado por la elaboración y el refinamiento de los puntos de vista. Así se realizará una iteración entre punto de vista externo e interno, hasta que el punto de vista interno tenga un nivel de detalle suficiente para realizar la implementación.

Los pasos para elaborar los modelos de agentes e interacciones son:

1. Identificar los roles del dominio de la aplicación.
2. Para cada rol, identificar las responsabilidades y servicios que proporciona y necesita cada rol.
3. Ver las interacciones asociadas requeridas y contenido de información para cada servicio.
4. Refinar la jerarquía de agentes

Este proceso permite asignar funcionalidad a los agentes y determinar sus relaciones.

La metodología para el desarrollo de los modelos del punto de vista interno (creencia, objetivos y planes) parte de los servicios proporcionados por el agente, las interacciones y

eventos relacionados con él. Este proceso se compone de dos pasos:

1. Analizar los medios para alcanzar los objetivos.
2. Construir las creencias del sistema mediante el análisis del contexto.

Es interesante observar que se hace más énfasis en los objetivos que en los comportamientos de los agentes.

### b) Ingeniería de las Vocales

Esta metodología da una correspondencia a los puntos de vista que tenemos que tener en cuenta para realizar el modelado con las vocales [95].

Los distintos puntos de vista son: Agente, Entorno, Interacciones, Organización y Usuario.

Su propósito es considerar bibliotecas de componentes que den soluciones para cada uno de estos aspectos. Dependiendo de la clase de sistema que se quiera realizar, variará el orden de las vocales, de modo que se profundizará más en unos puntos de vista que en otros.

La plataforma Volcano [95] orientada a componentes, permite aplicar las ideas de la Ingeniería de las Vocales.

### c) MAS-CommonKADS

CommonKADS [105] es una metodología para el desarrollo de sistemas de gestión de conocimiento que ya tenía en cuenta la idea de considerar distintos puntos de vista. En este caso se consideraban seis modelos, los cuales están representados en la figura 2.23:

1. Organización en la que trabaja el sistema de gestión de conocimiento.
2. Tareas.

3. Agente (el sistema experto).

4. Comunicaciones.

5. Experiencia (conocimiento del dominio y conocimiento de resolución del sistema).

6. Diseño (arquitectura del sistema de gestión del conocimiento).

Dado que los sistemas expertos [46, 71, 87] son centralizados al intentar aplicar esta metodología a SMA se deben considerar aspectos de interacción de agentes y distribución. Así, surgió MAS-CommonKADS[132].

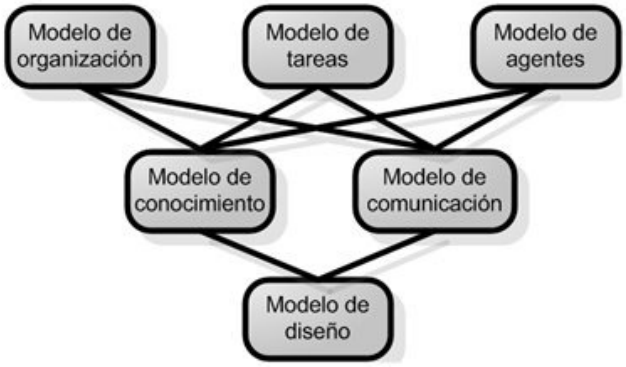


Figura 2.23: Modelos de MAS-CommonKADS

Las características más importantes que se añaden a CommonKADS son:

- Utilización de una notación orientada a objetos para estructurar el SMA.
- Definición de casos de uso para la captura de requisitos.
- Técnicas de ingeniería de protocolos para especificar las interacciones entre agentes.

### d) MESSAGE e INGENIAS

MESSAGE propone integrar varias metodologías e investigaciones realizadas en el área de los agentes inteligentes. Incorpora cinco puntos de vista similares a la metodología de la

Ingeniería de las Vocales [21].

Como novedad permite describir los elementos que modelan un SMA desde cada punto de vista usando meta-modelos [57].

Una extensión de estos meta-modelos y su validación mediante la experimentación y el desarrollo de varios casos de uso es la metodología INGENIAS [112].

Los puntos de vista que considera INGENIAS para el modelado de un SMA están recogidos en la figura 2.24 y son:

- *Agente*: Describe responsabilidades con tareas y roles. Considera el control del agente definiendo sus objetivos y estados mentales que requieren durante la ejecución.

- *Organización*: Este punto de vista describe el marco en el que existen agentes, recursos, tareas y el propósito del sistema. Para definir la organización hay que considerar la funcionalidad, las relaciones sociales y la estructura, la cual determina la arquitectura del sistema. La arquitectura se compone de grupos y flujos de trabajo, los cuales relacionan tareas, recursos asociados a las tareas y sus responsables.

- *Entorno*: Define los sensores y los actuadores de los agentes. Identifica los recursos, los agentes y las aplicaciones con las que tienen que interactuar los agentes.

- *Tareas y objetivos*: Su principal propósito es justificar la ejecución de tareas en función de los objetivos. Para ello se basa en el principio de racionalidad [26], el cual proporciona la descomposición en tareas y objetivos.

- *Interacciones*: Describen cómo se produce la coordinación entre los agentes. Muestran la motivación de los participantes de la interacción, requiriendo para ello información del estado mental que requieren los agentes en su ejecución y tareas que ejecutarán.

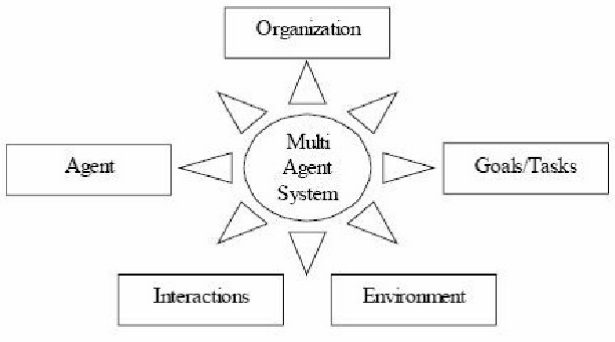


Figura 2.24: Puntos de vista de un SMA en la metodología INGENIAS

**e) MASSIVE y ODAC**

MASSIVE [75] considera siete puntos de vista los cuales son: entorno, tareas, roles, interacciones, sociedad, arquitectura y sistema.

ODAC [92] considera los cinco puntos de vista del estándar Open Distributed Processing (ODP): empresa, información, computacional, tecnología e ingeniería.

### 2.7.2. Metodologías atendiendo a roles, servicios, interacciones y organizaciones

Como se ha visto anteriormente, los roles, servicios e interacciones pueden servir de guía para la construcción de un SMA. El concepto de rol aparece de forma natural cuando se considera el aspecto organizativo del sistema. Los roles identifican la funcionalidad, mediante servicios, y las características de las partes en las interacciones. Los agentes que representan roles tienen capacidades para realizar dichos servicios.

Si en el anterior apartado se tenía en cuenta el SMA como un todo desde el que se iban a modelar los agentes, en este apartado veremos metodologías que realizan el modelado de manera inversa, ya que modelarán a los agentes y las relaciones entre estos analizando los

roles que deben desempeñar, los servicios que debe de ofrecer el SMA, las interacciones que tiene que haber entre los distintos agentes y las posibles organizaciones que se den dentro del Sistema Multi-Agente.

### a) Gaia

Esta metodología, propuesta por Wooldridge [139], considera un sistema basado en agentes como una sociedad u organización, la cual consta de una colección de roles que tienen ciertas relaciones entre ellos y que toman parte en patrones de interacción institucionalizados con otros roles.

Cada rol está compuesto por cuatro atributos definidos en un modelo de roles:

- *Responsabilidades*: determinan su funcionalidad descrita como propiedades de disponibilidad y seguridad.
- *Permisos*: definen derechos de los roles a la hora de utilizar recursos.
- *Actividades*: aquellas funciones que pueden realizarse sin interacturar con otros roles.
- *Protocolos*: son modos de interacción con otros roles. Quedan definidos en un modelo de interacciones.

La metodología Gaia pretende ayudar al analista a ir sistemáticamente desde unos requisitos iniciales a un diseño suficientemente detallado para pasar a la implementación. Aunque está orientada principalmente a las fases de análisis y diseño, tal y como se puede ver en la figura 2.25.

- **Etapa de análisis:**

  1. *Identificar los roles del sistema.* Los roles pueden ser individuos, departamentos u organizaciones y tras este paso obtenemos como resultado una lista de roles descrita de manera informal.

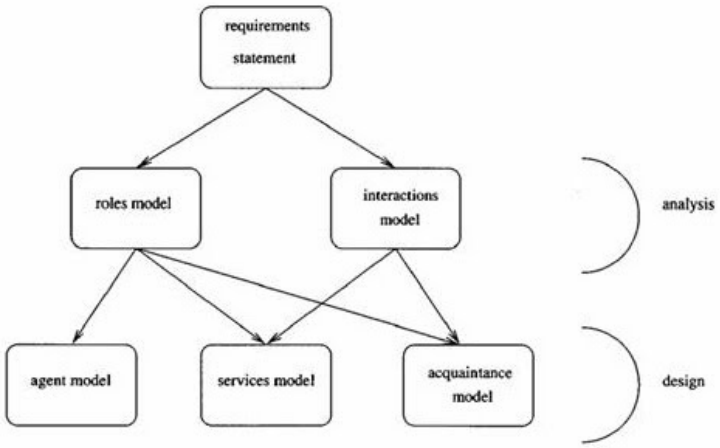


Figura 2.25: Fases de modelado software en Gaia

2. *Identificar y documentar los protocolos asociados a cada rol.* Esto permite definir el modelo de interacciones.

3. *Elaboración de un modelo de roles.* Con la información obtenida en los pasos anteriores podremos construir un modelo donde se incluyan los permisos, responsabilidades, protocolos y actividades en las que participa cada rol.

4. *Realizar iteración de pasos 1 a 3.* Se estará iterando hasta que tengamos en el modelo el nivel de detalle deseado.

- **Etapa de diseño:** En esta etapa se transformarán los modelos abstractos creados en la etapa de análisis, en modelos con una descripción a bajo nivel suficiente de abstracción como para poder ser implementados. Para llevar a cabo dicha tarea, se generan 3 modelos:

  - *Modelo de agentes:* identifica los tipos de agentes (se crearán por agregación de roles), número de instancias de cada tipo que se creará en tiempo de ejecución y rol.

  - *Modelo de servicios:* define los servicios del agente asociados a cada rol y las propiedades de los mismos (entradas, salidas, pre-condiciones, post-condiciones).

- *Modelo de conocimiento:* se trata de un grafo dirigido que describe simplemente los enlaces de comunicación entre agentes.

Gaia no se preocupa de cómo definir un modelo computacional del sistema de agentes, en este sentido da libertad al programador para determinar qué implementación será más apropiada. Simplemente describe cómo coopera la sociedad de agentes para logar los objetivos globales del sistema y cómo participa cada agente individual.

El hecho de que esta metodología no incluya una forma de implementar los servicios que documenta, hace que se encuentre bastante limitada y que adquiera relevancia en ingeniería de agentes software cuando se combina con otras metodologías, centrándose principalmente en el análisis de roles e interacciones y en el estudio de organizaciones [143].

### b) MaSE

La metodología MaSE [51](Multi-agent System Software Engineering) trata todo el ciclo de desarrollo desde la descripción hasta la implementación. A diferencia de Gaia, se concibe como una abstracción del paradigma orientado a objetos donde los agentes son especializaciones de objetos. Esta especialización consiste en la coordinación de los agentes a través de conversaciones y en la naturaleza proactiva del agente.

Una vez obtenidos los requerimientos del sistema, MaSE hace especial hincapié en las etapas de análisis y diseño, las cuales están compuestas por varios pasos, descritos en la figura 2.26.

- **Etapa de análisis:**

1. *Captura de objetivos.* En esta fase el analista tiene que identificar los objetivos, a partir de los documentos de requerimientos, y estructurarlos por importancia en un diagrama de jerarquía de objetivos.

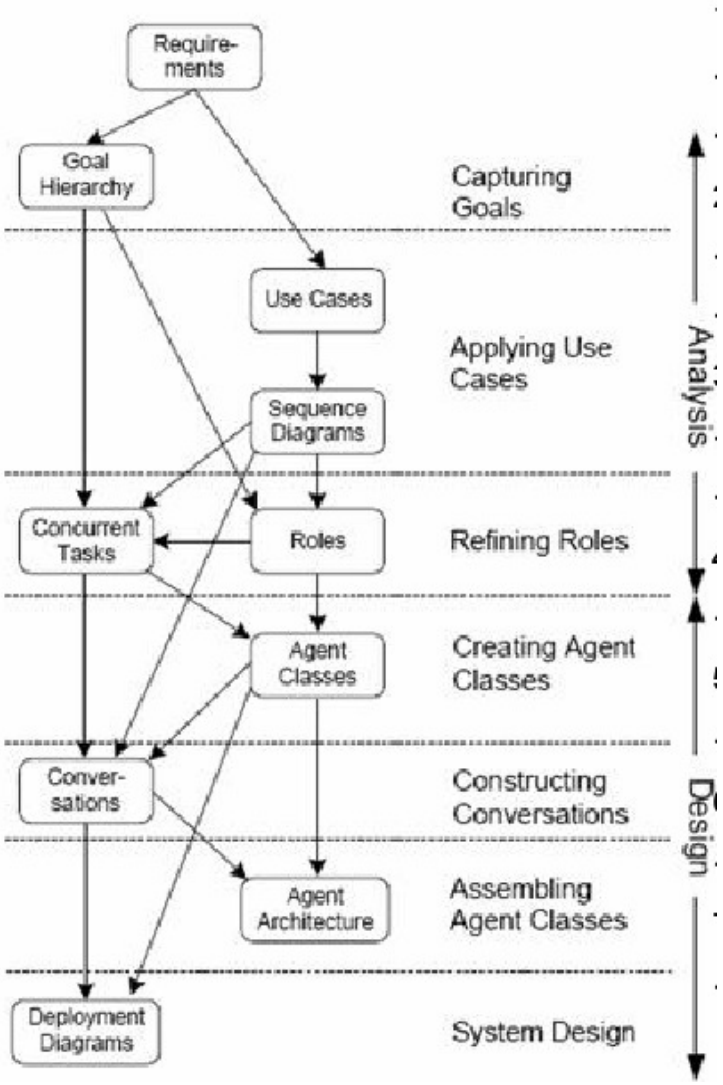


Figura 2.26: Fases de modelado software en MaSE

2. *Aplicación de casos de uso.* En este paso se convierten los objetivos en roles y tareas asociadas mediante el análisis de los casos de uso y la definición de diagramas de secuencia (similiar a UML).

3. *Refinamiento de roles.* La tercera fase en MaSE es para asegurar que hemos identificado todos los roles necesarios y desarrollar las tareas que definen el comportamiento de los mismos y los patrones de comunicación. Una vez que todos los objetivos son tenidos en cuenta asignándole un rol a cada uno o estando agrupados varios objetivos en un rol, los roles quedan captados en el modelo de roles.

- **Etapa de diseño:**

1. *Creación de clases de agentes.* Las clases de agentes son identificadas a partir de los roles y descritas en el diagrama de clases de agentes. Se suele crear una clase por rol, aunque pueden crearse varias clases para un mismo rol o una clase para varios roles. Es recomendable agrupar roles que tengan en común gran cantidad de pasos de mensajes por motivos de optimización del sistema. Una vez creado el diagrama de clases, la organización del sistema queda definida.

2. *Construcción de conversaciones.* Una conversación en MaSe la podemos definir como un protocolo de coordinación entre dos agentes descritos por un autómata iniciador y otro receptor y representados mediante dos diagramas de clases de comunicación: uno para el emisor y otro para el receptor. Por este motivo la mayoría de las implementaciones realizadas en MaSE se limitan a la definición de agentes reactivos (véase apartado 2.2).

3. *Ensamblado de clases de agentes.* En este paso se crea el interior del agente, usando un lenguaje de modelado arquitectónico que combina la naturaleza abstracta de los lenguajes tradicionales de descripción arquitectónica con el lenguaje de restricción de objetos, permitiendo al diseñador especificar detalles de bajo nivel.

4. *Diseño del sistema.* En esta fase se decide la configuración final del sistema a ser implementado, definiendo la arquitectura global del sistema mediante diagramas

de despliegue y tomando las decisiones de implementación que no se habían tomado antes, como el lenguaje de programación a usar o el "framework" para las comunicaciones.

### c) AALAADIN

Ferber crea AALAADIN, basándose en que los roles y los servicios ayudan a estructurar la funcionalidad asociada a un agente o grupo de agentes, contribuyendo a la comprensión y gestión de sistemas complejos desde una perspectiva organizacional [20]. No es una metodología de especificación de agentes, tal y como hemos visto con Gaia y MaSE, sino más bien un meta-modelo genérico de SMA utilizando tres conceptos básicos: Agentes, Grupos y Roles (AGR), tal y como podemos observar en la figura 2.27.

Este modelo es la base de la herramienta para el desarrollo de agentes MadKit [10]

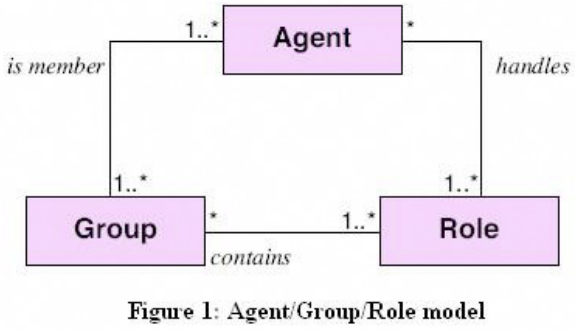


Figura 2.27: Fases de modelado software en AALAADIN

## 2.7.3. Unificación de notaciones de modelado de SMA: AUML

Muchas metodologías de modelado de agentes parten del paradigma orientado a objetos. Así, con el propósito de disponer de un lenguaje de modelado estándar para el desarrollo de la tecnología y las herramientas de SMA, nace AUML. Se trata de un lenguaje de descripción de

agentes en FIPA como extensión de UML que se usa para especificar protocolos de interacción de agentes [30].

Mediante los diagramas de protocolo en AUML se extienden los diagramas de secuencia de UML con mecanismos para definir roles de los agentes, las líneas de vida de los agentes (hilos de interacción, que se pueden dividir en varias líneas de vida con paralelismo y decisiones, y mezclarse en un punto futuro, protocolos anidados) patrones de interacción que pueden reutilizarse con guardas y restricciones y una semántica extendida para los mensajes UML.

Estos esquemas se han utilizado para especificar los protocolos de interacción FIPA (véase apartado 2.4.3).

Otra extensión es la propuesta por Parunak [111]. En este caso, se consideran los conceptos de roles, grupos, dependencias y actos del habla, que están relacionados con la organización, la cual, resulta irrelevante dentro de un SMA.

Para ello plantea utilizar tres artefactos:

- *Grupo*: como conjunto de agentes que están asociados por un interés, propósito o tarea comunes. Esto se representa mediante un diagrama de clases con líneas de partición para organizar los roles en el grupo.
- *Rol*: como una representación de una función agente, permitiendo que un agente pueda desempeñar varios roles. El modo de representación lo tenemos mediante el diagrama de clases, en el que quedará reflejada la organización, los agentes y sus roles.
- *Entorno*: se encarga de la distribución en información entre agentes.

La notación propuesta ofrece la posibilidad de definir clases de agentes e instancias. Una clase de agentes estará definida por:

1. Un nombre de clase y, opcionalmente, una lista de roles.

2. Una descripción de estado, similar a una descripción de atributos en diagramas de clase, pero que permiten describir el estado mediante fórmulas bien formadas.

3. Acciones.

4. Métodos, como en las clases de UML.

5. Capacidades del agente.

6. Actos comunicativos, los cuales definen la interfaz principal del agente.

Actualmente se están investigando otros aspectos del modelado de agentes, aunque debido a la falta de financiación, el proyecto no avanza con la rapidez que requiere la tecnología [1].

## 2.8. Implementación de un SMA

A la hora de abordar la implementación de un Sistema Multi-Agente, podemos preguntarnos: ¿Con qué lenguajes de programación se puede implementar un SMA? ¿Existe alguna plataforma que nos facilite los agentes básicos para el funcionamiento de un SMA? ¿Qué plataforma es la más extendida?

A lo largo de este apartado pretendemos resolver estas preguntas, analizando en primer lugar qué lenguajes de programación son más apropiados para la implementación de SMA y describiendo posteriormente las plataformas de Sistemas Multi-Agente más extendidas.

### 2.8.1. Lenguajes de programación

La mayoría de los SMA están programados en Java, esencialmente por su buena adecuación a Internet, ya que con Java tenemos muchas bibliotecas de clases para interfaz de usuario, comunicaciones, etc. De esta manera resulta relativamente cómodo añadir las clases

y métodos específicamente requeridos para la implementación genérica de un SMA. Siguiendo esta filosofía se han implementado las plataformas JADE y ABLE, que estudiaremos a continuación. Además, Java facilita la integración con otros paradigmas de programación (programación lógica, sistemas basados en reglas de producción,...) lo que permite implementar cada agente empleando alguno de estos paradigmas y posteriormente integrarlos en una plataforma común.

No obstante, se pueden implementar SMA en cualquier lenguaje que sea orientado a objetos, ya que aunque no va a ofrecer las ventajas que ofrece Java, puede que para el problema en cuestión necesitemos un sistema en el que, por ejemplo premie la velocidad. En tal caso, se podría implementar en C++, ya que al no necesitar máquina virtual, el uso de recursos es menor y por tanto tendrá un mayor aprovechamiento computacional [28, 125].

### 2.8.2. Plataformas de Sistemas Multi-Agente

Desde que surgió la tecnología de SMA se han desarrollado muchas plataformas con la finalidad de permitir la interconexión y ejecución de los agentes de un SMA. A partir del nacimiento de FIPA la mayoría de las plataformas basadas en arquitecturas de agente propias están evolucionando para admitir también el estándar FIPA. Actualmente las plataformas más utilizadas son JADE y ABLE, las cuales se detallarán en los siguientes apartados.

#### a) JADE

JADE [6] es la plataforma de desarrollo de agentes y SMA basada en el estándar FIPA que más se usa en la actualidad. Por este motivo, vamos a describir en qué consiste y sus principales características.

La plataforma JADE (*Java Agent DEvelopment Framework*) es un entorno software, implementado en Java, para simplificar la implementación de SMA a través de un *middleware* acorde con las especificaciones FIPA y dotado de un conjunto de herramientas gráficas para

ayudar al desarrollador con tareas de depuración y puesta en marcha de la aplicación. Algunas de las funcionalidades más atractivas que JADE proporciona al programador son:

- Un sistema en el que cada agente se ejecuta como un hilo separado de los demás, capaz de ejecutarse en máquinas remotas y de comunicarse con otros agentes de forma totalmente transparente.
- Transporte eficiente de mensajes asíncronos, con una API transparente.
- Implementación de servicios de *Páginas amarillas* y *Páginas blancas*.
- Administración sencilla y eficiente del ciclo de vida de los agentes. Los agentes reciben un identificador único (AID, *Agent Identifier*) y una dirección automáticamente durante su creación. Se proporcionan APIs para crear, suspender, reanudar, bloquear, despertar, migrar, clonar y destruir agentes.
- Soporte para movilidad de agentes, entre procesos y entre distintas máquinas, de forma transparente.
- Herramientas para interceptar y analizar mensajes de comunicación entre agentes.
- Soporte para ontologías y lenguajes de contenido.
- Integración con tecnologías Web como JSP, servlets, applets y servicios Web.
- Soporte para la plataforma J2ME para dispositivos móviles.

La arquitectura de JADE está basada en el estándar FIPA. Proporciona varios agentes ya implementados, que pueden ser utilizados para distintos propósitos, y que están ligados a la plataforma, tal y como se puede apreciar en la figura 2.28:

- *Facilitador de directorio* (veáse apartado 2.3.3).
- *Sistema de Gestión de Agentes* (véase apartado 2.3.3).

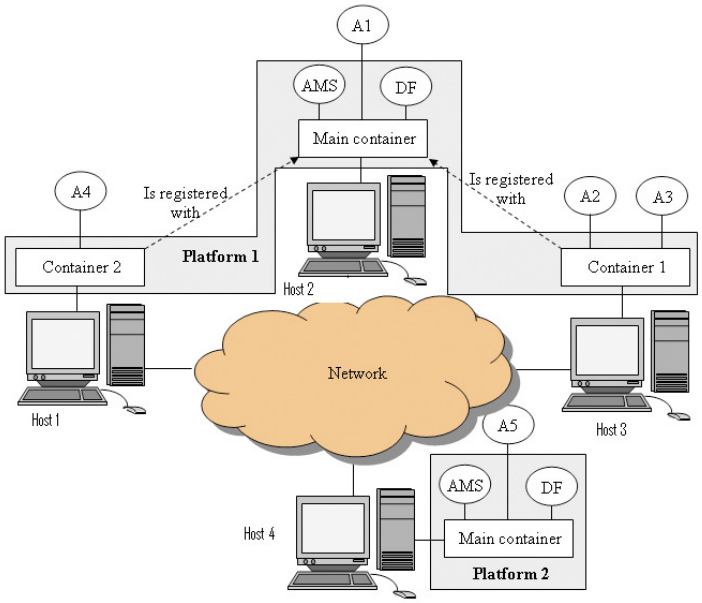


Figura 2.28: Arquitectura de JADE

- *Sniffer*: agente que ayuda en la depuración de las interacciones entre agentes.

- *Inspector*: agente que permite controlar el ciclo de vida de un agente en ejecución y los mensajes que intercambia, tanto los que haya en la cola de entrada como los de la cola de salida.

La principal particularidad añadida de esta arquitectura es que una plataforma de agentes se compone de uno o varios contenedores de agentes (*containers*), donde residen los agentes. Estos contenedores, al igual que la plataforma que los contiene, pueden encontrarse en un mismo equipo o estar distribuidos en una red. Existe un contenedor especial, el contenedor principal (*main container*), que es el núcleo de la plataforma y el primer contenedor en ponerse en funcionamiento, además de ser el responsable de albergar a los agentes necesarios para arrancar la plataforma. Un ejemplo de los agentes dentro del contenedor principal lo encontramos en la figura 2.29.

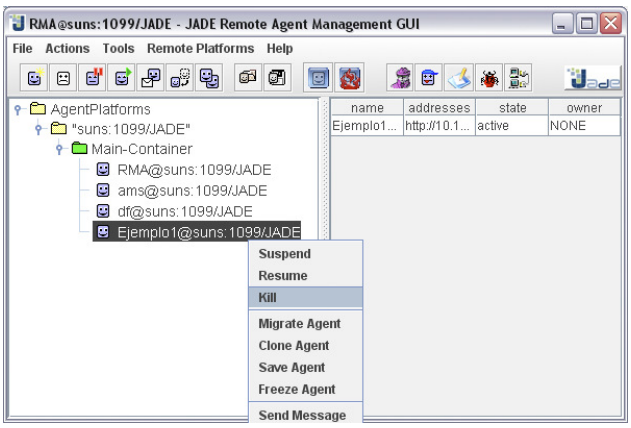


Figura 2.29: Entorno de JADE

### b) ABLE

ABLE [35](Agent Building and Learning Environment) es una plataforma de desarrollo y ejecución de agentes de IBM que permite implementar los agentes como componentes JavaBeans. Además de la definición del modelo de componentes y las herramientas para su programación, ABLE proporciona también algunos componentes para razonamiento automáticos y aprendizaje.

El entorno ABLE proporciona un conjunto de interfaces y clases Java que permiten construir componentes JavaBeans llamados AbleBeans. Hay multitud de AbleBeans a partir de los cuales los desarrolladores pueden construir otros, o bien extenderlos. Un AbleBean queda definido por:

- *BeanInfo*: proporciona una descripción del componente que se puede consultar en tiempo de ejecución.
- *Customizer*: permite personalizar cada uno de las características parametrizables del AbleBean, al igual que ocurre con los JavaBeans.
- *Inspector*: permite observar las diferentes entradas y salidas del AbleBean en cuestión.

Los agentes en ABLE se denominan AbleAgents y normalmente están constituidos por varios AbleBean de funcionalidad genérica. Los AbleBeans se pueden conectar entre sí, para formar otros más complejos y AbleAgent, de tres maneras:

1. *Con flujo de datos*: se utilizan en aplicaciones que siguen de manera natural un paradigma de proceso de flujo de datos.

2. *Por eventos*: siguen un modelo de suscripción/publicación donde los objetos se tienen que registrar como interesados en recibir eventos de un AbleBean. Estos eventos se implementan como AbleEvents y pueden recibirse de forma síncrona o asíncrona. Su ventaja es que desacoplan los AbleBeans.

3. *Mediante propiedades*: permite sincronizar dos propiedades en AbleBeans diferentes, de manera que cuando se modifica una propiedad en uno de ellos, se aplicará el método de modificación correspondiente en el otro.

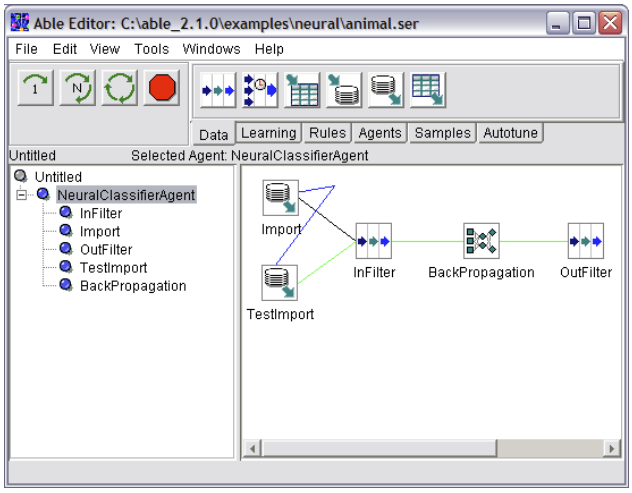


Figura 2.30: Entorno de ABLE

Los AbleBeans heredan de la clase AbleObject que proporciona los métodos que deben implementar los AbleBeans, siendo de especial importacia los métodos *init()*(para la inicialización del AbleBean), *process()*(para tratar los buffers de entrada) y *processTimerE-*

*vent()*(método que se invoca cuando el bean se configura para ser tratado como un agente autónomo).

El desarrollo de agentes en ABLE se puede realizar de manera sencilla aprovechando su interfaz gráfica, de modo que nos permite conectar AbleBeans utilizando los distintos mecanismos mencionados anteriormente y también definir el comportamiento de los agentes.

También hay dentro de ABLE herramientas de inspección que permiten obtener estadísticas sobre información de estado y datos de los agentes. Esto es útil para observar predicciones de series temporales y controlar el comportamiento de los agentes en el tiempo.

Actualmente ABLE proporciona la clase *AbleDefaultFIPAAgent*, la cual nos proporciona los métodos necesarios para construir agentes que tengan los comportamientos de agente que requiere el estándar FIPA.

## 2.9. Problemas y Carencias de los SMA

Como se ha visto en los apartados anteriores, los Sistemas Multi-Agente ofrecen unas características apropiadas para el desarrollo de aplicaciones distribuidas y complejas en entornos donde se disponga de numerosos componentes con diferentes niveles de experiencia e intereses conflictivos [27]. Los agentes, por su parte, añaden una serie de propiedades (autonomía, proactividad y comportamiento dirigido por objetivos) que permiten el desarrollo de numerosas actividades sin supervisión humana. Además, los agentes pueden incorporar habilidades de razonamiento y aprendizaje, que junto a la aplicación de técnicas de Inteligencia Artificial, permiten mostrar un comportamiento inteligente y realizar las tareas de manera óptima.

No obstante, a pesar de la cantidad de ventajas que supone la aplicación de esta tecnología, no son muchas las aplicaciones reales desarrolladas que exploten todo el potencial de la misma ya que en entornos reales se dan numerosos problemas que limitan su utilidad y

aplicabilidad. El principal inconveniente de los SMA radica en la comunicación entre agentes situados en diversas plataformas localizadas en servidores remotos, porque la utilización de protocolos propietarios no estándar (CORBA, RMI, DCOM) dificulta esta comunicación. Una medida que se está llevando a cabo es la utilización de Servicios Web, ya que al hacer uso de protocolos estándar como HTTP, evita que haya conflictos de comunicación en el paso de mensajes entre plataformas.

Otro problema añadido a los Sistemas Multi-Agente lo encontramos en la semántica de los procesos comunicativos. A pesar de que el estándar FIPA-ACL especifica formalmente un significado preciso para cada acto comunicativo o primitiva de comunicación, ninguna de las plataformas que implementan FIPA y satisfacen el estándar, es capaz de proporcionar el soporte adecuado para manejar esta dimensión semántica del lenguaje FIPA-ACL [90]. De este modo, se hace necesaria la existencia de un mecanismo que trate con los posibles problemas de interoperabilidad que puedan surgir entre los agentes cuando éstos llevan a cabo una comunicación o entren en un proceso de negociación.

La mayoría de los investigadores de IA en general y de SMA en particular, coinciden en que para que se produzca el salto de obtención de sistemas utilizables a gran escala a partir de los prototipos desarrollados hasta el momento, será necesario afrontar los retos planteados en [25]:

- **"Materialización"del concepto de agentes**: un agente se debe de identificar como una entidad computacional con una forma y un comportamiento bien definidos y de mayor nivel de abstracción que los elementos usuales en programación y debe tener asociado un patrón arquitectónico que especifique con claridad la forma externa (interfaces), la estructura interna (componentes de control), el comportamiento y el entorno computacional preciso para funcionar.
- **Alcanzar la compatibilidad con las tecnologías y plataformas computacionales existentes**. El funcionamiento de un agente no debe de depender de una plataforma propietaria, por lo que se hace necesaria la convergencia con otras tecnologías

y estándares, como los formalismos para representar ontologías (OWL) o los mecanismos y protocolos de descripción, búsqueda y acceso a los servicios y recursos (UDDI, WSDL,RDF,...).

- **Convergencia y complementariedad entre los métodos de ingeniería** como metodologías, plataformas y herramientas de desarrollo.
- **Complementar los modelos teóricos y los modelos de ingeniería** para detectar fallos y predecir las limitaciones lo antes posible.
- **Producción de aplicaciones BURI (Baratas, Útiles, Robustas e "Inteligentes")**

## Capítulo 3

# Arquitectura Multi-Agente

A lo largo de este capítulo revisaremos los conceptos tenidos en cuenta para desarrollar la arquitectura de nuestro SMA. En primer lugar, se describirán brevemente los mecanismos de negociación, pasando a ver en detalle el modelo de plataforma de negociación Kasbah. Seguidamente describiremos el diseño de la plataforma del SMA, definiendo los distintos tipos de agentes, componentes y tecnologías que serán utilizados en nuestro sistema. Por último, nos detendremos en las tecnologías basadas en los estándares FIPA utilizadas para la comunicación entre los agentes de nuestra plataforma , (ontologías) así, como en la comunicación entre la plataforma y el exterior.

## 3.1. Negociación

Los mecanismos de negociación [25] modelan la coordinación entre agentes autointeresados capaces de llegar a acuerdos vinculantes. Estos mecanismos permiten la convergencia hacia decisiones de coordinación conjuntas mediante una comunicación explícita. La mayoría de los mecanismos de negociación en SMA están inspirados en modelos tomados de las ciencias sociales y especialmente de la Economía [138]. A su vez, muchos de estos enfoques tienen su fundamento formal en la corriente cooperativa de la Teoría de Juegos [29].

Dentro de la negociación, podemos encontrar tres aspectos básicos:

- *Lenguaje*: trata de las primitivas de comunicación para negociar (proponer, refinar, confirmar, etc), su semántica, el objeto de la negociación (plan, oferta de tarea, etc) y los protocolos de negociación.
- *Decisión*: trata de qué aspectos tienen en cuenta los agentes para decidir en la negociación, cómo maximizar una función de utilidad, preferencias, estrategias de negociación (ser cooperativos, competitivos, etc).
- *Proceso*: estudio de modelos generales del proceso de negociación y de la conducta global de los participantes.

La complejidad de la negociación dependerá de los atributos involucrados, los tipos de valores y el número de agentes que negocian (uno a uno, uno a muchos, muchos a muchos). Jennings en 2001 propuso un modelo de negociación automática [102], extendido en 2005 por Narayanan y el propio Jennings [104], en el que distingue los siguientes cinco componentes:

1. **Protocolo de negociación**, el cual especifica formalmente las reglas del proceso de negociación (participantes, estados posibles, eventos, y acciones válidas).
2. **Objetos de la negociación**, a saber, los asuntos sobre los cuales los agentes se encuentran negociando (puede ser un único elemento, como el precio, o múltiples elementos de discordia, como calidad, condiciones, tiempo, etc).
3. **Preferencias de negociación de los participantes**, es decir, los objetivos de cada uno de los agentes que participan en el proceso de negociación.
4. **Estrategia de negociación de los participantes**, la cual indica cómo debe de responder un agente ante una situación determinada.
5. **Parámetros de negociación de los participantes**, los cuales establecen restricciones como el tiempo límite para alcanzar un acuerdo.

Existen tres técnicas o aproximaciones principales a la negociación automatizada [55, 102, 122]:

- **Técnicas basadas en teoría de juegos**: intentan determinar la estrategia óptima analizando la interacción como si fuera un juego entre participantes idénticos, y buscando su equilibrio.

- **Técnicas basadas en heurística**: estos métodos ofrecen aproximaciones a las decisiones que se pueden llegar a tomar a partir de estudios basados en Teoría de Juegos. Sin embargo, estas técnicas poseen dos inconvenientes muy importantes:

    1. Los modelos generalmente producen resultados sub-óptimos (adoptan una noción aproximada de racionalidad y no examinan el espacio completo de resultados posibles).

    2. Es muy difícil predecir de manera precisa, cómo se comportará el sistema.

- **Técnicas basadas en argumentación**: estas aproximaciones intentan superar las limitaciones impuestas por los modelos tradicionales, permitiendo a los agentes intercambiar información adicional o "argumentar" acerca de sus creencias y otras actitudes mentales durante el proceso de negociación. En este sentido, se entiende *"argumento"* como una pieza de información que puede permitir a un agente tanto justificar su postura como influenciar las posturas de los otros. El objetivo último es aumentar la probabilidad de alcanzar un acuerdo y maximizar la calidad de éste, intercambiando argumentos que influyan en el estado del resto de los agentes.

Podemos concluir, que no existe una única solución válida para todos los problemas y dominios [122], por lo que será el diseñador quien elija la técnica más acorde a las condiciones desencadenantes del proceso de negociación. En nuestro caso, los agentes integrantes del SMA tienen que cooperar para llegar a una solución común(a un problema de TDG), es decir, tendrán que negociar cambiando sus preferencias hasta que alcancen el grado mínimo de consenso deseado.

### 3.1.1. Kasbah. Ejemplo de Plataforma de Negociación

A lo largo de la literatura podemos encontrar distintos ejemplos de plataformas de SMAs de negociación. Por ejemplo MASFIT [12] y OASIS [25] citados en el capítulo 1. No obstante, se ha escogido Kasbah [47] como plataforma base de negociación, al ser una de las primeras plataformas implementadas de este tipo, así como una de las más utilizadas para temas didácticos.

Kasbah es un portal web donde los usuarios acceden para comprar y vender productos. El modo de hacer esto es creando *agentes vendedores* y *agentes compradores*, lo cuales interaccionaran en el *mercado* (la plataforma). Kasbah es, en definitiva, un Sistema Multi-Agente. El *mercado* está diseñado para manejar cualquier tipo de agente que sea compatible con los protocolos usados. Los agentes, aunque sean completamente autónomos, no son extremadamente complejos, ya que no emplean técnicas de Inteligencia Artificial ni de Aprendizaje Automático. Lo que hace a Kasbah verdaderamente interesante es el aspecto de interacción del SMA, así como la negociación entre los *agentes* en el *mercado*.

#### Agentes vendedores y compradores

##### *Agentes vendedores*

La idea de los *agentes vendedores* es similar a la de un anuncio clasificado. Cuando el usuario crea un *agente vendedor*, le asigna una descripción del artículo que desea vender. A diferencia de los tradicionales anuncios clasificados, los cuales permanecen a la espera de obtener noticias de algún usuario interesado, los *agentes vendedores* de Kasbah actúan de manera pro-activa. Básicamente, ellos intentan venderse a sí mismos, introduciéndose en el *mercado*, contactando con partes interesadas (*agentes compradores*) y negociando por ellos para llegar al mejor acuerdo. Los *agentes vendedores* son autónomos una vez que están en el *mercado*, ellos negocian y toman sus propias decisiones sin la necesidad de la intervención del usuario. No obstante, el usuario tiene una gran importancia en el comportamiento de su

*agente vendedor*, ya que cuando se crea un nuevo agente, éste se tiene que inicializar con unos parámetros en los que se basará el *agente vendedor* para realizar la venta. Estos parámetros son:

- *Fecha tope para la venta*. Las personas normalmente tienen una fecha tope en la que quieren vender algo.
- *Precio deseado*. Éste es el precio por el que al usuario le gustaría que se vendiese su artículo (Precio máximo).
- *Precio mínimo aceptable*. Éste es el precio más bajo por el que el usuario acepta vender su artículo.

Estos parámetros, definen el objetivo del *agente vendedor*, el cual intentará vender el artículo al mayor precio posible, idealmente intentará que sea al precio deseado, pero en caso de no ser así, intentará que esté lo más alejado posible del precio mínimo aceptable, de manera que se llegue al máximo beneficio [107]. En este caso, Kasbah emplea varias estrategias de negociación. Así, conforme se va acercando la fecha tope, el precio del artículo baja de manera lineal, exponencial o cuadrática (ver figura 3.1). El usuario puede ver las ofertas que ha tenido su *agente vendedor*, ver los agentes con que ha hablado y qué precios han ofrecido por sus artículos. Una vez que se llega a un acuerdo, el *agente vendedor*, avisa al usuario (vía email) para que éste realice la confirmación de la venta.

***Agentes Compradores***

En este caso, la idea es exactamente opuesta a la de los *agentes vendedores*. Debemos imaginar los *agentes compradores* como un anuncio en el que ponemos lo que estamos buscando. Cuando creamos un *agente comprador*, el usuario describe el artículo del que está interesado. En este caso, podremos especificar a un conjunto de *agentes vendedores*, que estén en el *mercado*, y mandar a nuestro *agente comprador* directamente a que negocie con ellos.

Al igual que con los *agentes vendedores*, a los *agentes compradores* hay que asignarles

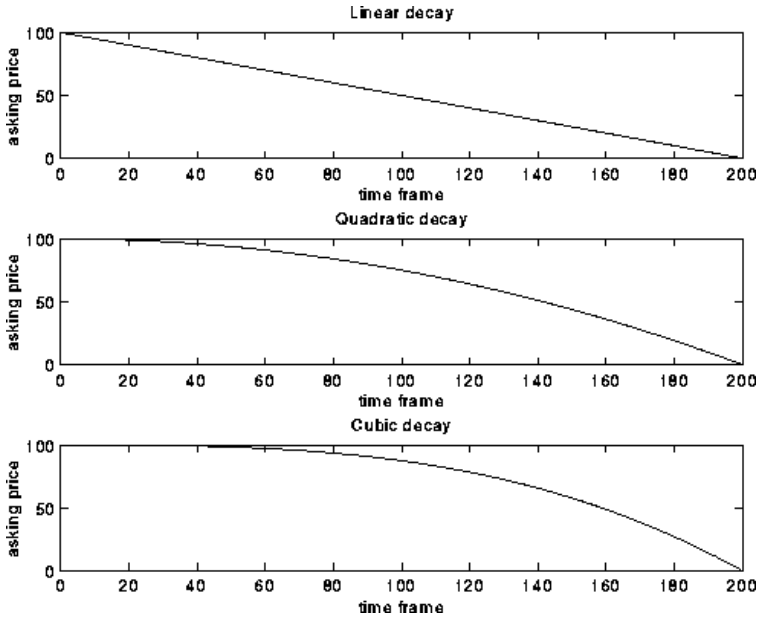


Figura 3.1: Estrategias de negociación agente-vendedor en Kasbah

unos parámetros que marcarán las pautas de las negociaciones. Estos parámetros son:

- *Fecha tope de compra*
- *Precio deseado*. Es el precio que le gustaría pagar al usuario (Precio mínimo).
- *Máximo precio aceptado*. Es el máximo precio que el usuario está dispuesto a pagar.

Una vez creado, el *agente comprador* es introducido en el *mercado* donde negociará con los *agentes vendedores*, intentando comprar el artículo al menor precio posible. Conforme se vaya acercando la fecha tope para la compra, el *agente comprador* irá aumentando el precio de manera lineal, cuadrática o cúbica, dependiendo de las indicaciones del usuario.

### Mercado

Los *agentes compradores* y *vendedores* encontrarán y negociarán en el *mercado*, el cual facilitará la interacción entre los agentes. Hay muchos roles que se dan dependiendo de los

mercados, pero como mínimo se necesita que los agentes hablen el mismo idioma de manera que se puedan entender. En el caso de Kasbah, cuando un *agente comprador* es creado, el mercado le facilita una lista de los posibles *agentes vendedores* para su artículos. Por otra parte, cuando se crea un *agente vendedor*, el *mercado* le hace llegar una lista de los posibles compradores.

La arquitectura del *mercado* es independiente, usando protocolos estándar, por lo que si un agente habla la lengua común de los mercados, entonces podrá participar en el *mercado* de Kasbah.

Cabe destacar, que para el desarrollo de este proyecto, se han considerado las estrategias de negociación descritas en Kasbah, las cuales se usarán para la modificación de preferencias durante el proceso de consenso sin necesidad de intervención humana.

## 3.2. Diseño de la plataforma: Arquitectura Multi-Agente

Para el diseño de la plataforma, vamos a concebir la plataforma como una "institución" donde se estipulan y se hacen cumplir *convenciones explícitas* de tres tipos:

- **Convenciones Ontológicas y de Comunicación** en las que se distinguen tanto los elementos necesarios para poder llevar a cabo una actividad de negociación, como el significado de los mensajes que se intercambian dentro de la plataforma.
- **Convenciones Sociales** que regulan las interacciones de los participantes en la negociación mientras intervienen en ella.
- **Reglas de conducta individual** a las que están sujetos los participantes en la negociación y que establecen los compromisos, obligaciones y derechos que les atañen.

A continuación veremos la arquitectura diseñada para el sistema Multi-Agente de negociación sobre el que se sustenta nuestro sistema. Dicha arquitectura cumple con el estándar

FIPA [3] y está implementada en la plataforma JADE [6], vista en el capítulo anterior.

El SMA diseñado es un sistema Multi-Agente cooperativo, donde los agentes tienen que colaborar entre sí, mediante la negociación, para alcanzar un objetivo ya sea individual o colectivo. Dado que en un acto de negociación se distinguen participantes que desempeñan distintos roles, en nuestro SMA tendremos diferentes agentes con roles determinados, definiéndose para cada uno de estos roles una serie de responsabilidades y líneas de actuación claras. En este caso, los agentes que se van a describir forman parte de la instanciación de la plataforma para consenso en TDG que se desarrollará en este proyecto.

- *Agente Moderador*: Este agente asume el rol de moderador humano en un proceso de negociación, siendo el responsable de garantizar un correcto desarrollo del proceso.
- *Agente Usuario*: Un agente usuario representa a un usuario humano en el sistema, actuando como tal de forma autónoma. El número de agentes usuarios existentes en el sistema variará dependiendo del número de usuarios participantes en el proceso de negociación.

Dado que un moderador humano realiza un gran número de tareas dentro de un proceso de negociación, en nuestro Sistema Multi-Agente se ha decidido añadir una serie de agentes específicos para dar soporte al moderador con algunas de las tareas de los que se componen el proceso de negociación. Estos agentes suponen diferentes especializaciones del moderador y son los siguientes:

- *Agente Evaluador de Resultados*: Este agente se encarga de la obtención de los resultados que se van dando en las distintas rondas de negociación y de notificar al agente moderador de esto.
- *Agente Recomendador*: Su responsabilidad se centra en llevar a cabo las operaciones necesarias para aconsejar a los agentes usuarios, de modo que la negociación pueda dar un resultado positivo y ambos agentes puedan cumplir sus objetivos.

- *Agente Analista*: Este agente asume la labor de almacenar la información relativa a cada proceso de negociación de forma persistente en una base de datos.

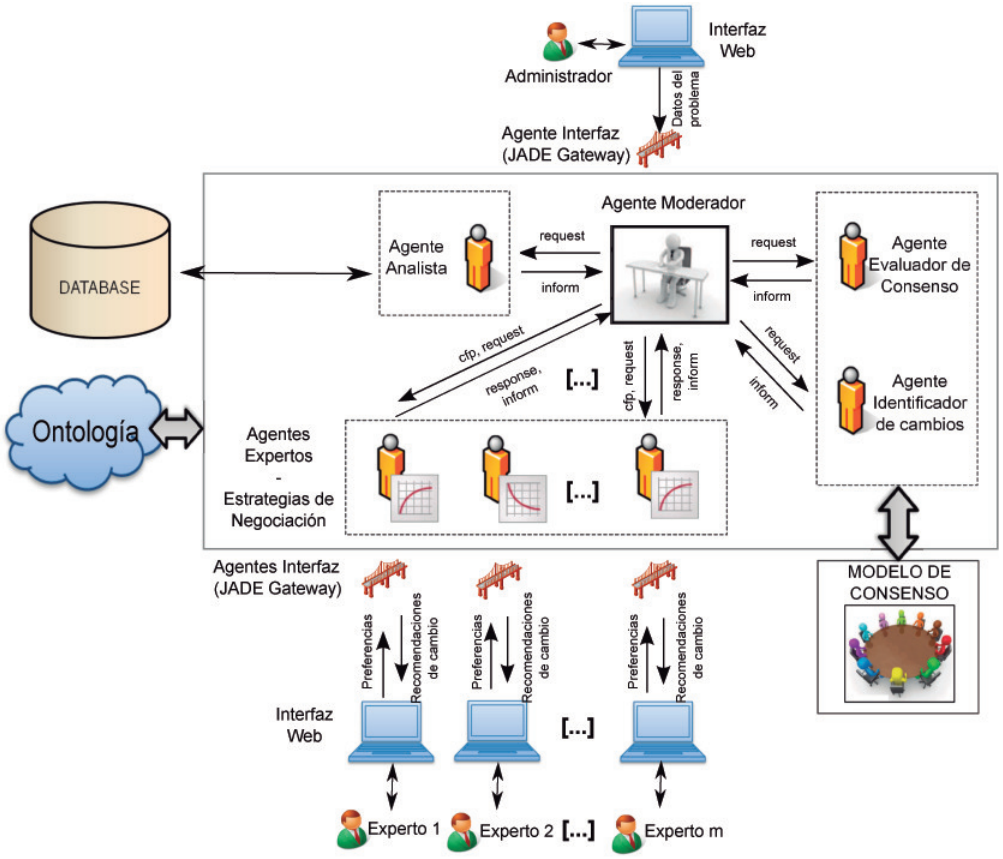


Figura 3.2: Arquitectura del SMA de Negociación

Otros agentes y componentes fundamentales en la arquitectura del sistema son los siguientes:

- *Agente Gateway*: Es un agente intermediario entre los agentes específicos del proceso de negociación y la interfaz web de usuario. Proporciona a los agentes de la plataforma los parámetros de entrada introducidos por medio de la interfaz y recoge los resultados de la ejecución del proceso de negociación llevado en la plataforma para mostrarlos en dicha interfaz.

- *Agentes y componentes FIPA*: La plataforma JADE, al estar basada en el estándar FIPA, proporciona una serie de agentes y componentes de utilidad para nuestra arquitectura, como son el agente *DF( Directory Facilitator)* que registra los servicios ofrecidos por cada agente específico del proceso de negociación, el *AMS (Agent Management System)* para administrar y controlar a todos los agentes del sistema durante su ciclo de vida, y el *MTS (Message Transport System)*, que permite una ejecución distribuida de los procesos de negociación, con agentes provenientes de diferentes plataformas.

- *Ontología y mensajes ACL*: Los agentes se comunican mutuamente por intercambio de mensajes FIPA ACL. Dado que el sistema cuenta con una ontología diseñada para que los agentes compartan el mismo vocabulario y semántica asociados a los problemas a tratar, los mensajes que estos intercambien tendrán como contenido expresiones con predicados o acciones de agente pertenecientes a esta ontología. Las ontologías se verán más detenidamente en el próximo apartado.

- *Base de Datos*: Contiene información sobre los procesos de negociación llevados a cabo por los agentes.

Estos elementos participantes en la arquitectura los podemos ver en la imagen 3.2.

## 3.3. Ontologías

El término *ontología*, en informática, hace referencia a la formulación de un exhaustivo y riguroso esquema conceptual dentro de un dominio dado, con la finalidad de facilitar la comunicación y el modo de compartir información entre diferentes sistemas.

Actualmente se están utilizando las ontologías dentro del marco tecnológico de la Inteligencia Artificial y de la representación del conocimiento.

La palabra ontología ha sido empleada desde hace siglos dentro del campo de la filosofía y del conocimiento. En la década de 1990, dicho concepto empezó a utilizarse en el campo de

la IA, la ingeniería, la representación del conocimiento y la lingüística computacional [42]. Dentro de la documentación, las ontologías son una herramienta para el intercambio y uso de conocimiento, ya que proveen una comprensión compartida y consensuada de un dominio del mismo, que puede ser comunicada entre personas y sistemas heterogéneos.

Los programas informáticos pueden utilizar las ontologías para una gran cantidad de propósitos, entre los que destacan el razonamiento inductivo [128], la clasificación [94] y una variedad de técnicas para resolución de problemas.

### 3.3.1. Definición

Según el diccionario de la Real Academia de la Lengua Española, el término *ontología* se define como la "parte de la metafísica que trata del ser en general y de sus propiedades trascendentales". Y es que fue la filosofía el primer campo de conocimiento donde se utilizó este término. Fue a partir de los primeros años de la década de los noventa cuando las ontologías comienzan a tener mayor protagonismo entre la ciencia de la computación y, más concretamente, en la Inteligencia Artificial y Procesamiento de Lenguaje Natural.

Las definiciones de ontología en Inteligencia Artificial se basan en la acepción de este término en filosofía, específicamente, en la interpretación del filósofo estadounidense Willar Van Orman Quine [136] quién aseveró que todo lo que puede ser cuantificado existe (en Inteligencia Artificial, lo que existe es exactamente aquello que puede ser representado computacionalmente). A continuación, se presentan las definiciones de ontología, dentro del ámbito de la informática, con más aceptación entre los investigadores de este tema.

En 1993, Gruber enunció:

*"Una ontología es una especificación de una conceptualización".*

El autor considera que una *conceptualización* está compuesta por objetos, conceptos y otras entidades que existen en una determinada área, y las relaciones que se dan entre ellos.

De un modo más abstracto, una *conceptualización* se puede definir como una interpretación estructurada de una parte del mundo que usan los seres humanos para pensar y comunicar sobre ella. Por *explícita*, se entiende que los conceptos y las restricciones se definen de forma explícita.

La definición de Gruber fue posteriormente refinada por Borst, en 1997, de la siguiente manera:

*"Una ontología es una especificación formal de una conceptualización compartida".*

En este contexto, *formal* se refiere a la necesidad de disponer de ontologías comprensibles por las aplicaciones informáticas. Por otro lado, *compartida* enfatiza la necesidad de consenso en la conceptualización, refiriéndose al tipo de conocimiento contenido en las ontologías, esto es, conocimiento consensuado y no privado.

Studer y su grupo, en 1998, se encargaron de fusionar las definiciones de Gruber y Borst, estableciendo la definición más extendida:

*"Una ontología es una especificación* ***formal*** *y* ***explícita*** *de una* ***conceptualización compartida****".*

Este enunciado fue a su vez, examinado en profundidad en el trabajo de ésta formulación, estableciendo lo siguiente:

*Conceptualización se refiere a un modelo abstracto de algún fenómeno en el mundo a través de la identificación de los conceptos relevantes de dicho fenómeno. Explícita significa que el tipo de conceptos y restricciones usados se definen explícitamente. Formal representa el hecho de que la ontología debería ser entendible por las máquinas. Compartida refleja la noción de que una ontología captura conocimiento consensual, esto es, que no es de un individuo, sino que es aceptado por el grupo."*

Por tanto, se puede comprobar que, a día de hoy, no existe una definición consensuada del término ontología. Muy al contrario, incluso los autores de las definiciones más reconocidas y aceptadas se replantean las mismas. Este es el caso de Gruber, el cual en [59], al ser preguntado sobre si cambiaría la definición que dio de ontología, responde:

*"Bien, los componentes más importantes de esa definición de ontología son que la ontología es un artefacto de representación (una especificación), distinta del mundo que modela, y que es un artefacto diseñado, construido para un propósito. Creo que la mayoría de científicos en computación obtienen la diferencia entre una especificación del mundo, incluso para mundos sintéticos. Respectivamente, no cambiaría la definición, pero intentaría enfatizar que nosotros diseñamos ontologías. La consecuencia de esta vista es que podemos aplicar una disciplina de ingeniería en su diseño y evaluación. Si las ontologías son cosas derivadas de una ingeniería, entonces no tenemos que preocuparnos tanto sobre si son correctas y favorecer el negocio de construirlas para hacer algo útil. Podemos diseñarlas para conocer objetivos funcionales y restricciones. Podemos construir herramientas que nos ayuden a gestionarlas y validarlas. Y podemos tener múltiples ontologías que se coordinen o compitan basadas en un criterio objetivo más que en una marca de fábrica o una autoridad."*

Finalmente, se puede destacar una definición extensiva de ontología indicando los componentes que la forman. En general, las ontologías proporcionan un vocabulario común de un área y definen, a diferentes niveles de formalismo, el significado de los términos y relaciones entre ellos. El conocimiento en ontologías se formaliza principalmente usando seis tipos de componentes: clases, atributos, relaciones, funciones, axiomas e instancias [58]:

- Una *clase* puede ser algo sobre lo que se dice algo, como por ejemplo un tipo de objeto, la descripción de una tarea, función, acción, estrategia, proceso de razonamiento, etc. Las clases en la ontología se suelen organizar en taxonomías. En todo caso, cabe destacar que ontología y taxonomía son dos elementos diferentes, aunque algunas veces la

noción de ontología se diluye en el sentido que las taxonomías se consideran ontologías completas [134]. Se suelen usar indistintamente los términos *clase* y *concepto*.

- Los *atributos* representan la estructura interna de los conceptos. Atendiendo a su origen, los atributos se clasifican en específicos y heredados. Los atributos específicos son aquellos que son propios del concepto al que pertenecen, mientras que los heredados vienen dados por las relaciones taxonómicas en las que el concepto desempeña el rol de hijo, y por tanto, hereda los atributos del padre. Los atributos vienen caracterizados por el dominio en el cual pueden tomar valor.

- Las *relaciones* representan un tipo de interacción entre los conceptos del dominio. Se definen formalmente como cualquier subconjunto de un producto de $n$ conjuntos, esto es: $R : C_1 \times C_2 \times ... \times C_n$.

- Las *funciones* son un tipo especial de relaciones en las que el n-ésimo elemento de la relación es único para los $n-1$ precedentes. Formalmente, definimos las funciones $(F)$ como $F : C_1 \times C_2 \times ... \times C_{n-1} \rightarrow C_n$. Como ejemplos, se pueden mencionar las funciones *"madre de"* y *"precio de un coche usado"*.

- Los *axiomas* son expresiones que son siempre ciertas. Pueden ser incluidas en una ontología con muchos propósitos, tales como definir el significado de los componentes ontológicos, definir restricciones complejas sobre los valores de los atributos, argumentos de relaciones, etc., verificando la corrección de la información especificada en la ontología o deduciendo nueva información.

- Por último, la *instancias* son las ocurrencias en el mundo real de los conceptos. En una instancia todos los atributos del concepto tienen asignado un valor concreto.

### 3.3.2. Tipos de Ontologías

Existen diferentes clasificaciones de tipos de ontologías entre las que se pueden agrupar en base a dos criterios[55]:

1. *Clasificación por el conocimiento que contienen.*

   Gertjan Van Heijst y su grupo propusieron los siguientes tres tipos de ontologías atendiendo al conocimiento que contienen [137]:

   - Ontologías terminológicas, lingüísticas: especifican los términos para representar conocimiento en un dominio determinado.
   - Ontologías de información: especifican la estructura de los registros de la base de datos. Los esquemas de bases de datos son un ejemplo.
   - Ontologías para modelar conocimiento: especifican conceptualizaciones de conocimiento. Este tipo de ontologías tiene una estructura interna mucho más rica que los anteriores y son las que interesan a los desarrolladores de sistemas basados en conocimiento.

2. *Clasificación por motivación.*

   Esta clasificación se hace dependiendo del motivo por el que se elabora la ontología. Se pueden distinguir cuatro grupos principales [133]:

   - Ontologías para la representación de conocimiento: permiten especificar las conceptualizaciones que subyacen a los formalismos de representación del conocimiento, por lo que también se denominan *meta-ontologías*.
   - Ontologías genéricas: definen conceptos generales y fundacionales del conocimiento como las estructuras parte/todo, la cuantificación, los procesos o los tipos de objetos. Estas ontologías son reutilizables en diferentes dominios.
   - Ontologías del dominio: permiten representar el conocimiento especializado pertinente de un dominio o subdominio, como la medicina, las aplicaciones militares, etc.
   - Ontologías de aplicación [60]: están ligadas al desarrollo de una aplicación concreta. Tales ontologías cubren los aspectos relacionados con aplicaciones particulares.

Típicamente estas ontologías toman conceptos de ontologías del dominio y genéricas, así como métodos específicos para realizar la tarea, por lo que frecuentemente no pueden ser utilizadas.

### 3.3.3. Lenguajes de Representación

Los principales lenguajes para la representación de ontologías son los siguientes [13, 144]:

- **RDF Schema (RDFS)**[14]: *Resource Description Framework Schema*. Es un entorno de trabajo para metados en la World Wide Web, desarrollado por el World Wide Web Consortium (W3C). Es un lenguaje primitivo de ontologías que proporciona los elementos básicos para la descripción de vocabularios.

- **SHOE** [91]: *Simple HTML Ontology Extensions*. Fue el primer lenguaje de etiquetado para diseñar ontologías en la Web. Este lenguaje nació antes de que se ideara la Web Semántica [16]. Las ontologías y las etiquetas se incrustaban en archivos HTML. Este lenguaje permite definir clases y reglas de inferencia, buscadores, Interfaces de Programación de Aplicaciones (API), etc; pero este proyecto fue abandonado a medida que se desarrollaron OIL y DAM; aunque también existe una serialización de este lenguaje en XML.

- **OIL** [2]: *Ontology Inference Layer*. Este lenguaje, derivado en parte de SHOE, fue impulsado también por el proyecto de la Unión Europea On-To-Knowledge, utiliza la sintaxis del lenguaje XML y está definido como una extensión de RDFS. Se basa tanto en la lógica descriptiva (declaración de axiomas) como en los sistemas basados en *frames* (taxonomía de clases y atributos). OIL posee varias capas de sub-lenguajes, entre ellas destaca la capa base que es RDFS, a la que cada una de las capas subsiguientes añade alguna funcionalidad y mayor complejidad. La principal carencia de este lenguaje es la falta de expresividad para declarar axiomas.

- **DAML y OIL** [2]: este lenguaje nació fruto de la cooperación entre OIL y DARPA y unifica los lenguajes DAML (*DARPA's Agent Markup Language*) y OIL. Se basa ya en estándares del W3C. El lenguaje DAML se desarrolló como una extensión del lenguaje XML y de Resource Description Framework (RDF) y para extender el nivel de expresividad de RDFS. DAML-OIL hereda muchas de las características de OIL, pero se aleja del modelo basado en clases (*frames*) y potencia la lógica descriptiva. Es más potente que RDFS para expresar ontologías. En la última revisión del lenguaje (DAML-OIL) ofrece ya un rico conjunto de elementos con los cuales se pueden crear ontologías y marcar la información para que sea legible y comprensible por un ordenador. También funciona como formato de intercambio. Sin embargo, este lenguaje presenta algunas carencias debido a su complejidad conceptual y de uso, complejidad que se intentó solventar con el desarrollo de OWL [11]. No obstante, se desarrollaron muchas aplicaciones que utilizan DAML-OIL, existiendo herramientas que permiten convertir de DAML a OWL.

- **OWL** [11]: *Web Ontology Language* o Lenguaje de Ontologías para la Web es un lenguaje de etiquetado semántico para publicar y compartir ontologías en la Web. Se trata de una recomendación del W3C, y puede usarse para representar ontologías de forma explícita. OWL, surge como una revisión al lenguaje DAML-OIL y es mucho más potente que éste. Al igual que OIL, OWL se estructura en capas que difieren en la complejidad y puede ser adaptado a las necesidades de cada usuario, al nivel de expresividad que precise y a los distintos tipos de aplicaciones existentes (motores de búsqueda, agentes, etc).

- **KIF** [7]: *Knowledge Interchange Format* es un lenguaje para representar ontologías basado en la lógica de primer orden. KIF, está basado en la lógica de predicados con extensiones para definir términos, metaconocimiento, conjuntos, razonamientos no monotónicos, etc; y pretende ser un lenguaje capaz de representar la mayoría de los conceptos y distinciones actuales de los lenguajes más recientes de representación del conocimiento. Se trata de un lenguaje diseñado para intercambiar conocimiento entre

sistemas de computación distintos, diferentes lenguas, etc; y no para la interacción entre seres humanos.

- **FOAF** [4]: Aunque no es exactamente un lenguaje para ontologías, ya que se trata de un vocabulario con definiciones que usa el lenguaje RDFS/OWL, FOAF hace más fácil que el software procese los términos del vocabulario FOAF para describir documentos. FOAF permite crear una base de datos unificada de información al normalizar una forma de definir categorías, tipos de relaciones, etc.

### 3.3.4. Ontologías en JADE

Como se vio en el capítulo anterior, la comunicación entre agentes se realiza mediante el paso de mensajes, los cuales seguían unos estándares (ACL y KQML). No obstante, también se comentó que cuando los Sistemas Multi-Agente requerían paso de mensajes complejos (con muchos atributos), era apropiado el uso de ontologías correspondientes al dominio de aplicación del SMA.

Una ontología en SMA supone la especificación de una conceptualización, la descripción de los conceptos y relaciones entre ellos, que pueden formar parte del conocimiento de un agente o una sociedad de agentes. En la actualidad, son muchas las propuestas y aplicaciones realizadas sobre SMA en las que se utilizan ontologías [56, 70, 76, 100].

Para incorporar a los distintos SMA ontologías, entornos como JADE nos ayudan a realizar la integración. Dado que para el desarrollo de nuestro Sistema Multi-Agente hemos utilizado JADE, en el siguiente apartado vamos a analizar la estructura de una ontología en dicho entorno. Las ontologías constan de una serie de componentes, que forman un *modelo de contenido*. En el caso de JADE, el modelo de contenido proporcionado se compone de los siguientes elementos [31]:

- **Conceptos**: son expresiones que representan objetos, cuya información se estructura en varios atributos. No aparecen aislados en los mensajes sino incluidos en predicados

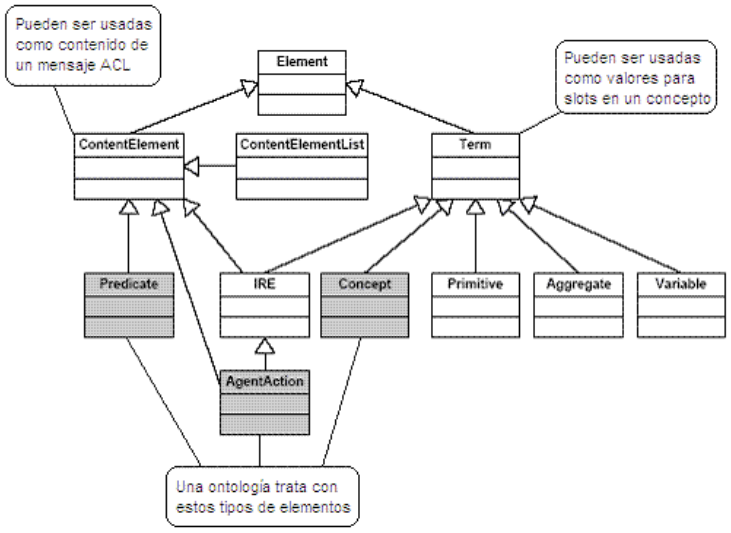


Figura 3.3: Modelo de contenido para ontologías en JADE

y acciones de agentes. Sus atributos pueden ser de tipos de datos simples (primitivas) o pueden ser instancias de otros conceptos.

- **Predicados**: son expresiones sobre el estado del mundo, que pueden ser verdaderas o falsas. Se suelen emplear en mensajes *Query-If* y en mensajes de respuesta *Inform*.

- **Acciones de los agentes**: son expresiones que indican acciones que pueden realizar los agentes. Normalmente constituyen el contenido de mensajes *Request*.

- **Otros elementos**: primitivas (elementos atómicos como números o cadenas de caracteres), agregaciones (atributos compuestos de múltiples elementos), expresiones (identifican las entidades para las que se cumple un predicado), variables.

Como se puede observar en la figura 3.3, el modelo de JADE consta de un amplio conjunto de componentes estructurados de forma jerárquica. En nuestro caso, como se verá en el próximo capítulo, sólo utilizaremos predicados y acciones de agente de forma directa. Estos a su vez se componen de uno o varios *términos*, los cuales pueden ser conceptos, primitivas o agregaciones, entre otros. Además, un agente debe encapsular todo predicado o acción de

agente como un objeto de la clase *ContentElement* para poder enviarlo a otros agentes como el contenido de un mensaje ACL. En el próximo capítulo se describirá en mayor detalle la ontología utilizada.

Tal y como hemos visto hasta ahora, la comunicación entre los agentes se realiza mediante el paso de mensajes ACL, los cuales definen su contenido mediante una ontología específica para el Sistema Multi-Agente de negociación. No obstante, los agentes necesitarán los datos que introduce el usuario humano para poder operar. Esto será labor del agente interfaz y lo realizará mediante el paquete jade.wrapper.gateway que se detallará en la siguiente sección.

## 3.4. Integración con Interfaz Web de Usuario

Llegados a este punto, es la hora de plantearnos cómo se realizará la comunicación desde el exterior con la plataforma y viceversa.

Una posible solución sería realizar el paso de información utilizando un lugar de almacenamiento intermedio (por ejemplo una base de datos) donde desde la interfaz se podrían guardar los datos y posteriormente éstos se recuperarían por parte de los agentes de la plataforma. Esta solución, podría funcionar con un número relativamente pequeño de agentes, pero no es la más adecuada, ya que el gasto de recursos a la hora de acceder y operar en el sistema de almacenamiento sería excesivo. Además, de no estar aprovechando la potencia de la tecnología de agentes, al reducir todo a una comunicación basada en cliente/servidor.

Desde jade.tilab se ha desarrollado el paquete jade.wrapper.gateway que trata de solventar estas deficiencias de manera que se puedan sacar las máximas prestaciones posibles, haciendo que la comunicación entre la interfaz y la plataforma Multi-Agente sea efectiva y eficiente. En el siguiente apartado se procede a describir su funcionalidad, ya que será el recurso utilizado para el desarrollo del proyecto.

## jade.wrapper.gateway

El paquete principal de Jade que utilizaremos para la comunicación con el servlet es jade.wrapper.gateway, que incluye las clases:

**JadeGateway**: Es una clase que a modo de pasarela nos permitirá conectar código no JADE con SMA basados en JADE. Esta pasarela mantendrá un agente (de la clase **GatewayAgent**) que será el que se comunique directamente con nuestro servlet. La activación y terminación de este agente se realiza mediante la clase JadeGateway, por lo que resulta transparente para el programador.

JadeGateway permite realizar la implementación de la pasarela de comunicación de dos maneras:

1. Creando un agente que derive de GatewayAgent. Así, solo tendremos que implementar dos métodos:

   - **processCommand:** que será el método encargado de establecer la comunicación, recibiendo como parámetro un objeto que contiene la información con la que queremos operar en la plataforma.
   - **releaseCommand:** es el método que devuelve el objeto pasado por parámetro a la plataforma al servlet, tras haber realizado las operaciones oportunas.

   Éste es el modo de implementación utilizado en el desarrollo de este proyecto.

2. Implementando un comportamiento que derive de la clase Gateway-Behaviour, para que un agente cualquiera pueda funcionar como pasarela. Este comportamiento también tendrá que implementar los métodos *processCommand* y *releaseCommand*.

Un ejemplo de JadeGateway lo podemos ver en la figura 3.4

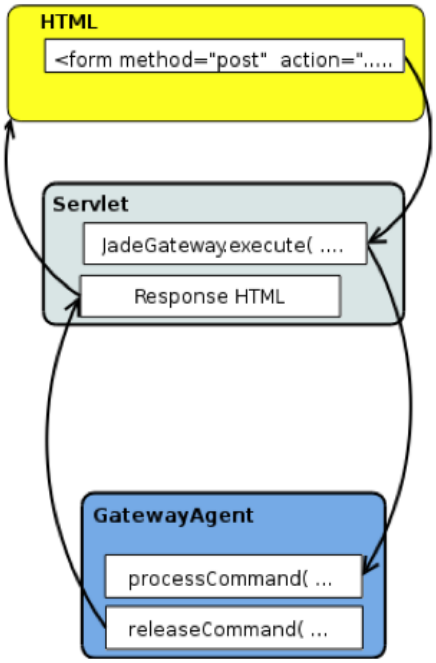


Figura 3.4: Funcionamiento de JadeGateway

# Capítulo 4

# Desarrollo del Proyecto

Después de haber revisado el estado del arte de la teoría de agentes, más en concreto de los Sistemas Multi-Agente, y haber descrito la arquitectura del SMA de negocio realizado, en este capítulo pasamos a describir en detalle el desarrollo del proyecto.

## 4.1. Descripción del proyecto

A grandes rasgos, el proyecto realizado consiste en una aplicación mediante la que se pueden llevar a cabo distintos procesos para que varios usuarios puedan llegar a sus objetivos en un proceso de negociación, moderados por un administrador y apoyados en un sistema Multi-Agente.

Tras comentar de modo muy general el cometido del proyecto, vamos a abordar el mismo siguiendo las etapas de Ingeniería del Software, utilizando metodologías de modelado de agentes. No hay una definición única y estandarizada para el término "Ingeniería del Software", no obstante a continuación se presentan dos definiciones que pueden ser perfectamente válidas:

- *Ingeniería del Software* es la construcción de software de calidad con un presupuesto limitado y un plazo de entrega en contextos de cambio continuo [49].

- *Ingeniería del Software* es el establecimiento y uso de principios y métodos firmes de ingeniería para obtener software económico que sea fiable y funcione de manera eficiente en máquinas reales [66].

Las fases que conforman la Ingeniería del Software son las siguientes:

1. **Especificación de Requerimientos:** se obtiene el propósito del sistema, las propiedades y restricciones del mismo.
2. **Análisis del sistema:** en esta etapa se obtiene un modelo del sistema correcto, completo, consistente, claro y verificable.
3. **Diseño del sistema:** fase en la que se definen los objetivos del proyecto y las estrategias a seguir para conseguirlos.
4. **Implementación:** es la etapa en la que se traduce el modelo a código fuente.
5. **Pruebas:** realizar pruebas unitarias para verificar la funcionalidad del sistema.
6. **Validación:** validar el sistema.
7. **Mantenimiento:** es la etapa más larga, ya que es la que se inicia en el momento en que la aplicación está puesta en marcha y se encarga del mantenimiento diario de la misma.

En la figura 4.1 se pueden apreciar las fases descritas anteriormente.

Hay que dejar claro, que la creación de software no es una ciencia exacta, por lo que en la mayoría de las ocasiones vamos a necesitar realizar un proceso de reingeniería para añadir o modificar consideraciones no tenidas en cuenta o tomadas de manera errónea. En los siguientes apartados de este capítulo se profundizará en estas fases y cómo se han llevado a cabo dentro de este proyecto. Cabe destacar la inclusión del modelado característico para los agentes que compondrán la plataforma, para lo que seguiremos el estándar AUML, visto

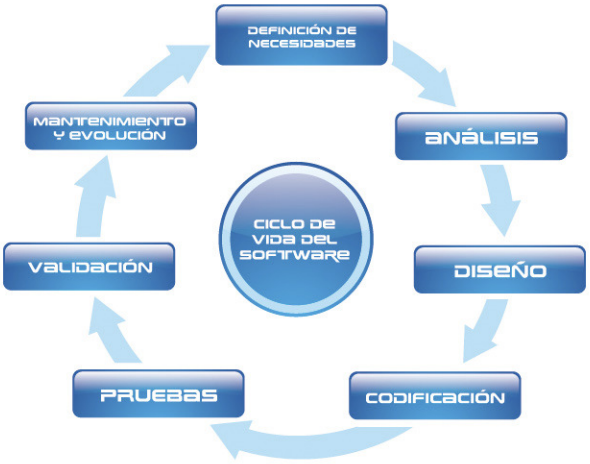


Figura 4.1: Fases de la Ingeniería del Software

en el capítulo 2. Por otra parte, hay que tener en cuenta que, aunque este proyecto se centra en el desarrollo de una plataforma multiagente de negociación, para realizar un caso práctico se va a instanciar un modelo de consenso, de manera que pueda utilizarse como herramienta de apoyo en problemas de toma de decisión en grupo (TDG). Los detalles del modelo de consenso instanciado, así como las características de los problemas de TDG se recogen en el anexo A.

## 4.2. Especificación de requerimientos

El primer paso de la Ingeniería del Software trata de especificar qué capacidades y condiciones (*requerimientos*) debe cumplir el sistema y más ampliamente el proyecto. Para ello habrá que encontrar, comunicar y registrar lo que se necesita realmente, de manera que tenga un significado claro para el cliente y los miembros del equipo de desarrollo. Esta etapa es vital, ya que el resto de etapas de la Ingeniería del Software se basarán en los resultados de la misma. Por este motivo, merece la pena realizar una especificación de requerimientos exhaustiva, dedicando el tiempo necesario (estudiando el dominio del problema, realizando entrevistas,...) para dejar bastante claro el cometido del proyecto. Para realizar una correcta

separación de requisitos y poder comunicar la información del sistema a los diferentes tipos de lectores, podemos diferenciar:

- *Los requerimientos del usuario*, que son declaraciones, en lenguaje natural y en diagramas, de los servicios que se espera que el sistema provea y de las restricciones bajo las cuales debe operar.
- *Los requerimientos del sistema*, los cuales establecen con detalle los servicios y restricciones del sistema. El documento de requerimientos del sistema, algunas veces denominado especificación funcional, debe ser preciso. Éste sirve como un contrato entre el comprador del sistema y el desarrollador de software.
- *Una especificación del diseño del software*, que es una descripción abstracta del diseño del software que es una base para un diseño e implementación detallados. Esta especificación agrega detalle a la especificación de requerimientos del sistema.

A menudo, los requerimientos de sistemas de software se clasifican en:

- *Requerimientos funcionales*: son declaraciones de los servicios que proveerá el sistema, de la manera en que éste reaccionará a entradas particulares y de cómo se comportará en situaciones particulares. En algunos casos, los requerimientos funcionales de los sistemas también declaran explícitamente lo que el sistema no debe hacer.
- *Requerimientos no funcionales*: son restricciones de los servicios o funciones ofrecidas por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo, estándares, etcétera.
- *Requerimientos del dominio*: se tratan de requerimientos que provienen del dominio de aplicación del sistema y que reflejan las características de ese dominio. Éstos pueden ser funcionales o no funcionales.

En los siguientes apartados nos centraremos en los requisitos funcionales y no funcionales a tener en cuenta en nuestro proyecto.

### 4.2.1. Requerimientos funcionales

Los requerimientos funcionales de un sistema describen la funcionalidad o los servicios que se espera que éste proveerá. En nuestro caso, para mostrar de manera ordenada las funcionalidades que ofrece el sistema, presentamos los requerimientos funcionales organizados por el tipo de usuarios que tienen permiso para utilizarlos: usuario "Administrador" y usuario "Experto".

- Usuarios "Administrador" y "Experto":
  - *Identificación y validación cuando un usuario accede al sistema.*
    Para poder acceder a nuestra aplicación, el sistema debe proporcionar al usuario un formulario en el que introducirá su nombre de usuario y contraseña. El sistema validará los datos y en el caso de ser correctos, el usuario accederá a la aplicación iniciando una sesión.
  - *Cerrar sesión.*
    El sistema proporcionará la opción " Cerrar sesión" de modo que el usuario pueda abandonar la aplicación y cerrar la sesión.
  - *Consultar un problema.*
    Se debe permitir consultar la información referente a un problema. En el caso particular de procesos de alcance de consenso, dicha información incluirá: descripción, expertos asignados, alternativas, umbral de consenso, número de rondas, estado actual del problema, resultado del proceso de consenso tras su finalización,etc.
- Usuario "Administrador":
  - *Crear un problema.*
    El sistema deberá proporcionar al usuario "Administrador" la posibilidad de crear un nuevo problema en la aplicación, en el que se introducirá el identificador del problema, una breve descripción del mismo, la temática del problema, el número

máximo de expertos participantes, el número de alternativas, así como el resto de parámetros necesarios (número de rondas, umbral de consenso,...).

- *Modificar un problema.*

  Debido a que los cambios en un problema pueden ser interesantes, el sistema permitirá al usuario "Administrador" modificar los parámetros y campos que se introdujeron en la creación del problema.

- *Eliminar un problema.*

  El usuario "Administrador" tendrá la posibilidad de eliminar los problemas.

- *Gestionar usuario de tipo "Experto"*

  Esta funcionalidad permitirá crear, eliminar y modificar los usuarios de tipo "Experto" en la aplicación.

- Usuario "Experto":

  - *Introducir/modificar perfil de experto.*

    Existe la posibilidad de que el usuario "Experto" necesite modificar datos de su perfil, por ejemplo, el usuario puede seleccionar aquella especialidad o especialidades (ej. gastronomía, viajes, etc) de cuyos problemas de TDG quiera ser invitado a participar. Por ello esta funcionalidad es necesaria para que se puedan realizar estas actualizaciones del perfil sin necesidad de la intermediación del usuario "Administrador".

  - *Introducir y modificar preferencias de un problema.*

    Esta es la funcionalidad por la que el usuario "Experto" puede introducir las preferencias de un problema cuando se inicia el proceso de consenso. Del mismo modo, puede modificar sus valoraciones atendiendo a las recomendaciones recibidas a lo largo de dicho proceso.

  - *Consultar información de un problema.*

    El usuario de tipo "Experto" puede solicitar información sobre el problema en el que está participando. Dicha información mostrará el estado del problema y, en

caso de haber finalizado, se indicará si se ha alcanzado consenso.

### 4.2.2. Requerimientos no funcionales

Los requerimientos no funcionales son aquellos que limitan a los requerimientos funcionales. Por este motivo, son tan importantes como éstos, pudiendo incluso llegar a ser críticos para la aceptación del sistema. Estos requerimientos, normalmente, especifican propiedades del sistema o del producto en sí (plataforma, velocidad, rendimiento, ...) y del diseño de la interfaz gráfica con el usuario, además de todas las restricciones impuestas por la organización (políticas de empresa, estándares, ...).

Los requerimientos no funcionales necesarios para este proyecto son los relacionados con las necesidades hardware y software de los equipos informáticos. Una vez cumplidos estos requisitos, el prototipo se podrá ejecutar sin ningún problema, ofreciendo al usuario todas sus prestaciones.

- **Requerimientos informáticos**

  Cuando nos referimos a los requerimientos informáticos, ya que éste proyecto se apoya en la arquitectura Cliente/Servidor, ejecutando parte en el Servidor (plataforma de agentes y base de datos) y otra parte en el Cliente, tenemos que distinguir entre las prestaciones necesarias para cada parte de esta arquitectura.

  Por un lado, los requerimientos del equipo informático del cliente son bastante simples, ya que únicamente necesitará un equipo con conexión a Internet y en el que tenga instalado un navegador que cumpla con los principales estándares de accesibilidad de la W3C.[1]

  En el caso del servidor, al ser donde estén alojados la plataforma Multi-Agente y la Base de Datos para gestionar usuarios y problemas, se aconseja que sea un equipo dedicado. En este caso, sus requerimientos los podemos dividir en hardware y software.

[1] http://www.w3.org

- **Requerimientos de Hardware**
  - *Velocidad de procesamiento.* El equipo debe procesar los datos lo suficientemente rápido como para ejecutar la aplicación en el menor tiempo posible. En nuestro caso, al poder darse una gran cantidad de interacciones entre agentes dentro de la plataforma, éstas deben ser atendidas sin que ello suponga una sobrecarga del sistema.
  - *Memoria RAM.* Este recurso es vital, ya que conforme vaya aumentando la cantidad de agentes activos en la plataforma, el gasto de memoria principal será mayor, con lo que el límite de agentes estará condicionado por la cantidad de memoria principal que se tenga.
  - *Memoria de Almacenamiento.* Se deberá tener una capacidad de almacenamiento lo suficiente como para almacenar la base de datos del prototipo, así como el software necesario para ejecutarlo.
  - *Tarjeta Gráfica.* En este caso, bastaría con cualquier tarjeta gráfica de las que se montan en los equipos informáticos actuales, puesto que no se requiere una gran potencia gráfica.
  - *Monitor.* Debe soportar una resolución de 1024 x 768 o superiores, prefiriéndose en cualquier caso monitores de pantalla plana con un amplio ángulo de visión.
  - *Conexión a Internet.* Para que la aplicación sea accesible desde el exterior, es necesario que el servidor éste conectado a Internet con una conexión en la que la subida de datos sea alta.
  - *Sistema de Alimentación Ininterrumpida (SAI).* Con este dispositivo, se garantiza la disponibilidad de la aplicación en caso de fallos eléctricos, protegiéndose también al equipo informático del servidor de posibles picos de tensión.
- **Requerimientos de Software**
  - *Sistema Operativo*: el servidor estará instalado en un sistema operativo basado

en UNIX.

- *Navegador*: Se precisará un navegador que cumpla con los principales estándares de accesibilidad de la W3C. Por ejemplo Mozilla Firefox o Google Chrome.
- El resto del software necesario para la instalación se proporcionará al Administrador de la aplicación, el cual tendrá como referencia un manual de instalación en el Anexo B

- Otros

  - *Tiempo de respuesta*. La aplicación está diseñada para dar un tiempo de respuesta corto, lo que favorece la utilidad de la misma.
  - *Disponibilidad*. El sistema tratará de garantizar una disponibilidad durante las 24 horas del día, los 365 días del año. Para ello, tal y como se comentó en el apartado de hardware, contaremos con un sistema de alimentación ininterrumpida, lo que garantizará el suministro eléctrico en caso de avería.
  - *Documentación*. La aplicación, además de contar con un manual de instalación (en el Anexo B), cuenta con dos manuales de uso dirigidos al usuario "Administrador" y al usuario "Experto". Estos manuales se pueden encontrar en los Anexos C y D.
  - *Seguridad*. La aplicación tiene que cumplir con todas aquellas medidas preventivas y reactivas que permiten resguardar y proteger la información de los usuarios y de los problemas ejecutados, de modo que mantengan la confidencialidad, la disponibilidad e integridad de la misma.
  - *Calidad*. Las herramientas empleadas a la hora de desarrollar la aplicación, así como los algoritmos implementados, están lo suficientemente contrastados para garantizar que los resultados que darán a los usuarios finales van a ser útiles y de calidad.
  - *Rendimiento*. Una de las premisas que se han tenido en cuenta a la hora de llevar a cabo este proyecto ha sido el aprovechamiento de recursos, de modo que el

rendimiento sea óptimo.

- *Aspectos legales y licencias.* El software empleado para el desarrollo de este proyecto tiene licencia Creative Commons, de modo que está permitido su uso de manera libre y gratuita.
- *Modificabilidad.* La aplicación se ha desarrollado de manera que tenga alta cohesión y bajo acoplamiento, pensando en que no suponga un gran esfuerzo la introducción de modificaciones en el futuro.

## 4.3. Análisis del sistema

En este apartado vamos a describir la funcionalidad de nuestro sistema. Para ello utilizaremos las herramientas basadas en el modelo de casos de uso que nos proporciona UML para su representación. En primer lugar, vamos a representar la funcionalidad del sistema en general, mediante el diagrama frontera como se muestra en la figura 4.2.

### 4.3.1. Diagrama Frontera

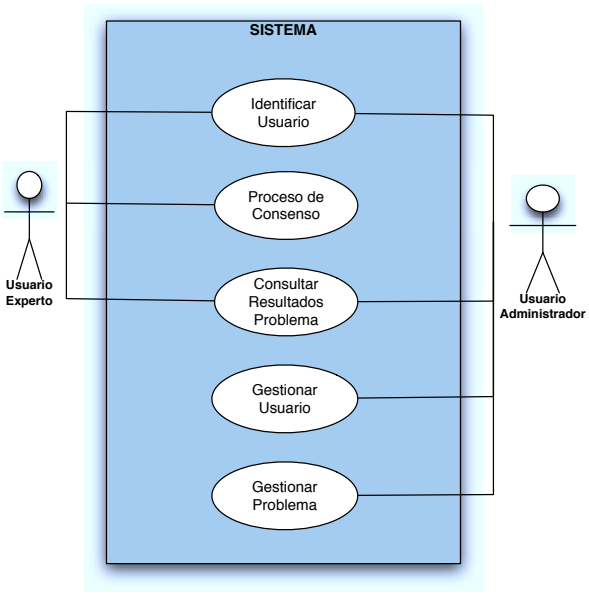


Figura 4.2: Diagrama Frontera

Una vez visto el diagrama frontera, pasamos a describir los distintos casos de uso, los cuales muestran las distintas funcionalidades del sistema de manera más específica.

### 4.3.2. Principales casos de uso

Un caso de uso representa una clase de funcionalidad dada por el sistema como un flujo de eventos.

#### Caso de Uso 1: Identificar Usuario

***Nombre:*** IdentificarUsuario

***Actores participantes:*** Administrador y Experto

***Diagrama del caso de uso:***

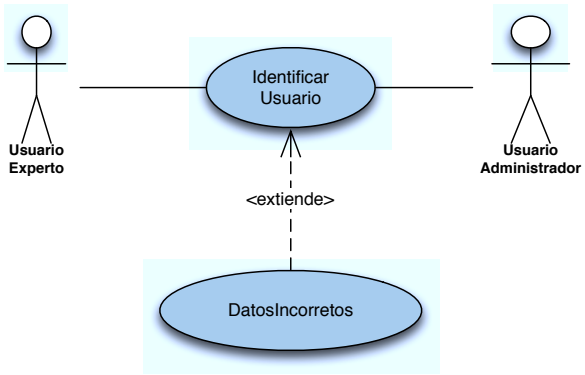


Figura 4.3: Diagrama de Caso de Uso IdentificarUsuario

***Condición de entrada:*** Deben de existir usuarios en la plataforma con nombre de usuario y contraseña, tanto para el usuario "Administrador" como para el usuario "Experto".

Condición de salida: El usuario "Administrador" o "Experto" es identificado en la plataforma.

***Flujo de eventos:***

1. El Sistema muestra el formulario de acceso a la plataforma.

2. El Usuario rellena los campos del formulario introduciendo su nombre de usuario y su contraseña. Posteriormente pulsa el botón *Aceptar*.

3. El Sistema comprueba en la base de datos que los datos introducidos con correctos (E-1).

4. Si el Usuario es:

    a Administrador: el Sistema mostrará el panel de control del usuario "Administrador".

b Experto: el Sistema mostrará el panel de control del usuario "Experto".

***Excepciones:***

**E-1** : Los datos introducidos por el usuario son incorrectos. El Sistema muestra un mensaje de error y vuelve a mostrar el formulario de entrada.

### Caso de Uso 2: Gestión de Usuarios

***Nombre:*** GestionUsuarios

***Actor participante:*** Administrador

***Diagrama del caso de uso:***

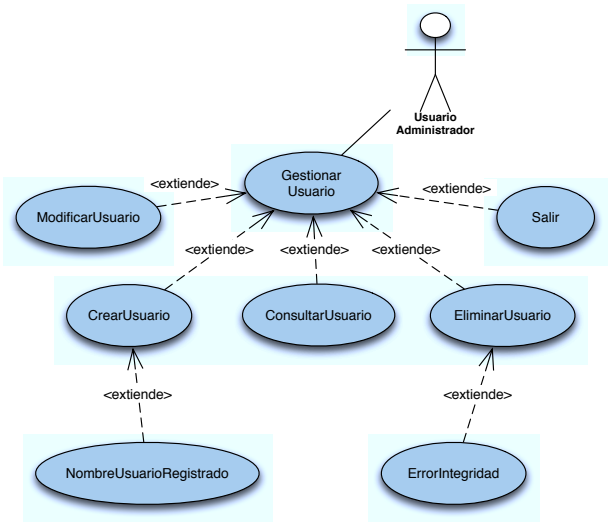


Figura 4.4: Diagrama de Caso de Uso GestionUsuarios

***Condición de entrada:*** Que el usuario "Administrador" esté dado de alta en el sistema.

***Condición de salida:*** El usuario "Administrador" lleva a cabo la acción elegida dentro del menú Gestión de Usuarios.

***Flujo de eventos:***

1. El sistema muestra un menú principal con las siguientes opciones, entre las que el usuario "Administrador" tendrá que elegir una:

   1.1. CrearUsuario, se llevará a cabo S-1.

   1.2. ModificarUsuario, se ejecutará S-2.

   1.3. ConsultarUsuario, se hará S-3.

   1.4. EliminarUsuario, se realizará S-4.

   1.5. Salir, se termina el caso de uso.

***Subflujo de eventos:***

**S-1** : CrearUsuario

1. El Sistema muestra un formulario en el que hay que elegir si se va a crear un usuario de tipo administrador o experto.
2. El Administrador elige el tipo de usuario que desea crear.
3. El Sistema muestra un formulario con los campos necesarios para introducir todos los datos del usuario que se va a crear.
4. El Administrador rellena el formulario (E-1).
5. Se inicia de nuevo el caso de uso GestionUsuarios.

**S-2** : ModificarUsuario

1. El Sistema muestra un listado con los usuarios dados de alta.
2. El Administrador selecciona el usuario que desea modificar.
3. El Sistema muestra todos los datos del usuario seleccionado, ofreciendo la posibilidad de modificarlos.

4. El Administrador modifica los campos deseados.
5. El Sistema guarda los cambios en la BBDD.
6. Se inicia de nuevo el caso de uso GestionUsuarios.

**S-3** : ConsultarUsuario

1. El Sistema muestra un listado con los usuarios dados de alta.
2. El Administrador selecciona un usuario.
3. El Sistema muestra toda la información relativa a ese usuario.
4. Se inicia de nuevo el caso de uso GestionUsuarios.

**S-4** : EliminarUsuario

1. El Sistema muestra un listado con los usuarios creados.
2. El Administrador selecciona al usuario que desea eliminar.
3. El Sistema elimina al usuario de la BBDD y guarda los cambios (E-2).
4. Se inicia de nuevo el caso de uso GestionUsuarios.

***Excepciones:***

E-1 : El nombre de usuario introducido para el nuevo usuario ya existe. El Sistema muestra un mensaje de error y le pide al Administrador que vuelva a introducir el nombre de usuario.

E-2 : El usuario seleccionado para eliminar es el Administrador que está realizando el caso de uso. El Sistema muestra un mensaje de error indicando que no se puede eliminar un usuario "Administrador" a sí mismo. Se vuelve al punto 1 de S-4.

### Caso de Uso 3: Gestionar Problemas

***Nombre:*** GestionarProblema

***Actor participante:*** Administrador

***Diagrama del caso de uso:***

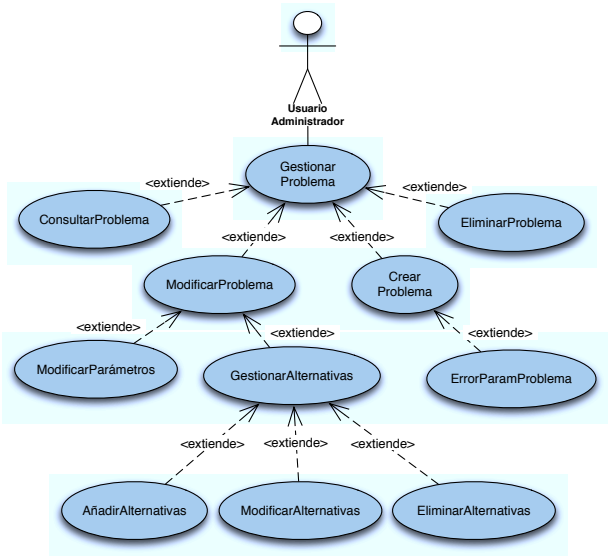


Figura 4.5: Diagrama de Caso de Uso GestionProblemas

***Condición de entrada:*** El usuario "Administrador" tiene que estar creado en el Sistema.

***Condición de salida:*** Se lleva a cabo la acción realizada por el Administrador.

***Flujo de eventos:***

1. El Sistema muestra un menú principal con las siguientes opciones entre las que el Administrador tiene que elegir una:

   1.1. CrearProblema, se realizará S-1.

1.2. ModificarProblema, se hará S-2.

1.3. ConsultarProblema, se ejecutará S-3.

1.4. EliminarProblema, se llevará a cabo S-4

1.5. Salir, se finalizará el caso de uso.

***Subflujo de eventos:***

**S-1** : CrearProblema

1. El Sistema muestra el formulario con los campos necesarios para la creación de un problema.
2. El Administrador introduce el número mínimo de expertos que intervienen en el problema y el número de alternativas. Así como el resto de parámetros para la creación de un problema (E-1).
3. El Sistema almacena la información en la BBDD.
4. Se inicia de nuevo el caso de uso GestionProblemas.

**S-2** : ModificarProblema

1. El Sistema muestra un listado con todos los problemas creados.
2. El Administrador selecciona el problema que desea modificar.
3. El Sistema muestra un menú con las siguientes opciones:

   3.1. ModificarParametros, se llevará a cabo S-2-1.

   3.2. GestionarAlternativas, se ejecutará S-2-2.

   3.3. Salir, finalizar el caso de uso.

   **S-2-1** : ModificarParametros

   3.1. El Sistema muestra todos los parámetros con los valores que se les asignaron cuando se creó el problema.

3.2. El Administrador modifica los parámetros que desee.

3.3. El Sistema almacena los cambios en la BBDD.

3.4. Se inicia de nuevo el caso de uso ModificarProblema.

**S-2-2** : GestionarAlternativas

3.1. El Sistema muestra las siguientes opciones:

3.1.1. AñadirAlternativas, se llevará a cabo S-2-2-1.

3.1.2. ModificarAlternativas, se realizará S-2-2-2.

3.1.3. EliminarAlternativas, se ejecutará S-2-2-3.

3.1.4. Salir, finalizar el caso de uso.

**S-2-2-1** : AñadirAlternativas

3.1.1. El Sistema muestra las alternativas asignadas para el problema.

3.1.2. El Administrador añade las alternativas que desee.

3.1.3. El Sistema guarda los cambios en la BBDD.

3.1.4. Se inicia de nuevo el caso de uso GestionarAlternativas

**S-2-2-2** : ModificarAlternativas

El Sistema muestra las alternativas asignadas para el problema.

El Administrador modifica la descripción de las alternativas que desee.

El Sistema guarda los cambios en la BBDD.

Se inicia de nuevo el caso de uso GestionarAlternativas

**S-2-2-3** : EliminarAlternativas

El Sistema muestra las alternativas asignadas para el problema.

El Administrador selecciona las alternativas que desea eliminar.

El Sistema guarda los cambios en la BBDD.

Se inicia de nuevo el caso de uso GestionarAlternativas

**S-3** : ConsultarProblema

1. El Sistema muestra la lista de problemas creados.

2. El Administrador selecciona un problema.

3. El Sistema muestra la información del problema seleccionado.

4. Se inicia de nuevo el caso de uso GestionProblemas.

**S-4** : EliminarProblema

1. El Sistema muestra la lista de problemas creados.

2. El Administrador selecciona el problema que desea eliminar.

3. El Sistema elimina el problema y guarda los cambios en la BBDD.

4. Se inicia de nuevo el caso de uso GestionProblemas.

***Excepciones:***

**E-1** : El número de expertos o alternativas introducidos no es entero positivo. El sistema muestra un mensaje de error.

**Caso de Uso 4: Proceso de Consenso**

***Nombre:*** ProcesoConsenso

***Actor participante:*** Experto

***Diagrama del caso de uso:***

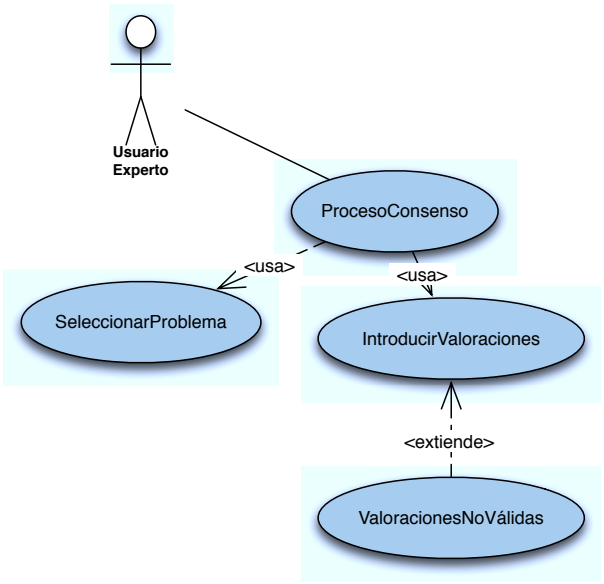


Figura 4.6: Diagrama de Caso de Uso Proceso Consenso

***Condición de entrada:*** El Experto tiene que estar dado de alta en el Sistema.

***Condición de salida:*** Se participa en un proceso de toma de decisiones en grupo en el que se tratará de llegar al consenso. Si pasadas las rondas establecidas, no se ha llegado al grado de consenso indicado en la creación del problema, el proceso finalizará sin haber llegado al consenso. Para iniciar el proceso de consenso, es necesario que haya aceptado participar en el mismo el número mínimo de expertos indicados en los parámetros, antes de que concluya la fecha límite para iniciar el proceso.

***Flujo de eventos:***

1. El Sistema muestra la lista de problemas activos en los que puede participar el Experto.
2. El Experto selecciona un problema.
3. El Sistema le indica si desea participar en dicho proceso para llegar al consenso.
4. El Experto acepta.

5. El Sistema muestra un formulario para que el Experto introduzca sus valoraciones y seleccione la estrategia de negociación.

6. El Experto introduce sus valoraciones (E-1).

7. El Sistema recoge las valoraciones del Experto y la estrategia de negociación y las procesa.

8. Una vez finalizado el proceso de consenso el Sistema muestra al Experto los resultados del mismo.

***Excepciones:***

**E-1** : Las valoraciones introducidas no están en un formato válido. Se muestra un mensaje de error y se vuelve al punto 5.

### Caso de Uso 5: Consultar Datos Problema

***Nombre:*** ConsultarDatosProblema

***Actores participantes:*** Administrador y Experto

***Diagrama del caso de uso:***

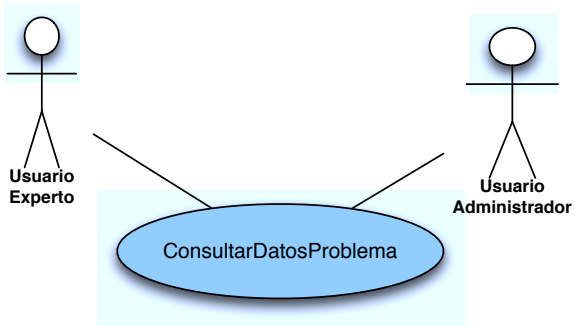


Figura 4.7: Diagrama de Caso de Uso Consultar Datos Problema

***Condición de entrada:*** Los usuarios tienen que estar dados de alta en el Sistema.

***Condición de salida:*** Se mostrará la información sobre el problema seleccionado.

***Flujo de eventos:***

1. El Sistema muestra la lista de los problemas creados.
2. El usuario selecciona el problema del que desea ver la información.
3. El Sistema muestra la información del problema.

### 4.3.3. Escenarios

Un caso de uso es una representación abstracta, una abstracción, de una funcionalidad del sistema a realizar. La representación concreta de un caso de uso se realiza mediante la creación de uno o más escenarios que muestren todas las posibles interacciones entre el sistema y los usuarios.

Los escenarios son historias ficticias que describen posibles interacciones de un caso de uso. Son de gran utilidad ya que permiten a los diseñadores anticiparse a los problemas. A pesar de ser historias ficticias, deben desarrollarse con el mayor nivel de detalle posible. Así, por ejemplo, los personajes deben tener nombre, motivaciones para usar la aplicación, deben ser situados en un contexto real con las restricciones que ello supone, etc. De esta manera es mucho más sencillo para los diseñadores discutir sobre la interfaz, ya que a las personas les cuesta más trabajo discutir las posteriores decisiones de diseño sobre una situación abstracta.

Esta forma de proceder, además, fuerza a los diseñadores a tener en cuenta el rango de usuarios que va a usar el sistema y el rango de actividades para las que lo van a usar. Los escenarios permiten hacer diferentes combinaciones de usuarios y actividades de forma que se tengan un cuenta todas las posibilidades.

Un escenario está formado por los siguientes elementos:

- Un nombre único y unívoco.
- Una descripción.
- Los actores participantes.
- El flujo de eventos.

Como se ha indicado, para cada caso de uso puede haber varios escenarios. Para este proyecto se han creado sólo los escenarios de los casos de uso más representativos del mismo, de manera que tengamos un ejemplo de las funcionalidades más importantes del sistema.

**Escenario 1: Gestión de Usuarios**

**Nombre**: ModificarUsuario.
**Descripción**: Modificar la especialidad de un usuario.
**Actores participantes**: Pablo (usuario Administrador) y Santi (usuario Experto)
**Flujo de eventos**:

1. Pablo accede a la aplicación con las credenciales del administrador (admin, admin) y selecciona “Gestión de Usuarios”.
2. El sistema muestra un menú principal con las siguientes opciones:
   - CrearUsuario.
   - ModificarUsuario.
   - EliminarUsuario.
   - ConsultarUsuario.
   - Salir
3. Pablo elige la opción “ModificarUsuario”.

4. El sistema muestra un listado con los usuarios dados de alta.

5. Pablo selecciona al usuario que desea modificar, en este caso el usuario elegido es Santi.

6. El sistema muestra todos los datos de Santi, ofreciendo la posibilidad de modificarlos.

7. Pablo modifica las especialidades de Santi, añadiendo las especialidades en *Enología* y *Gastronomía* y quitando la especialidad en *Dispositivos Móviles*.

8. El sistema guarda los cambios en la BBDD.

9. Pablo sale de la aplicación.

**Escenario 2: Gestión de Problemas**

**Nombre**: CrearProblema.
**Descripción**: Creación de un nuevo problema.
**Actores participantes**: José (usuario Administrador).
**Flujo de eventos**:

1. José accede a la aplicación con las credenciales del usuario Administrador (admin, admin) y selecciona "Gestionar Problema".

2. El sistema muestra un menú con las siguientes opciones:

   - CrearProblema.
   - ModificarProblema.
   - ConsultarProblema.
   - EliminarProblema.
   - Salir.

3. José elige la opción "CrearProblema".

4. El sistema muestra un formulario con los siguientes campos:
   - *Descripción:*
   - *Título:*
   - *Especialidad:*
   - *Número de Alternativas:*
   - *Alternativas:*
   - *Umbral de Consenso:*
   - *Número de Rondas:*
   - *Número Máximo de Expertos:*
   - *Fecha de Comienzo:*
5. José, rellena los distintos campos de la siguiente manera:
   - *Descripción:* Un grupo de amigos quieren llegar a un acuerdo en la elección de un restaurante para cenar el día 28 de junio.
   - *Título: Cena de amigos*
   - *Especialidad:* Gastronómica.
   - *Número de Alternativas:* 4.
   - *Alternativas:*
     a) Casa Antonio.
     b) La Picaña.
     c) O Mammamia.
     d) Tamesis London.
   - *Umbral de Consenso:* 0,75.
   - *Número Máximo de Rondas:* 4.
   - *Número Mínimo de Expertos:* 4.

- *Fecha de Comienzo:* 5/5/2013.

6. El sistema almacena la información en la BBDD.

7. José abandona la aplicación.

**Escenario 3: Proceso de Consenso**

**Nombre**: ProcesoConsenso.
**Descripción**: Proceso de consenso sobre dispositivos móviles.
**Actores participantes**: Iván (usuario Experto), Francisco Javier (usuario Experto), Luis (usuario Experto), Francis (usuario Experto), Raúl (usuario Experto).
**Flujo de eventos**:

1. Los usuarios Expertos acceden a la aplicación con sus credenciales.

2. El sistema invita a todos los expertos con especialidad en gastronomía (Iván, Francisco Javier, Luis, Francis y Raúl) a participar en el proceso de negociación.

3. Los usuarios responden a las peticiones:

   - Iván: Acepta participar.
   - Francisco Javier: Acepta participar.
   - Luis: Acepta participar.
   - Francis: Acepta participar.
   - Raúl: Rechaza participar.

4. El sistema muestra un formulario a cada usuario Experto para que introduzca sus valoraciones.

5. Los usuarios Iván, Francisco Javier, Luis y Francis introducen las preferencias, mediante relaciones de preferencia difusa (véase Anexo A) para las distintas alternativas

y seleccionan la estrategia de negociación deseada para delegar en el sistema para la modificación de preferencias durante el proceso de consenso (ver sec.3.1.1).

- Iván:

$$\begin{pmatrix} - & 0,3 & 0,5 & 0,8 \\ 0,7 & - & 0,2 & 1 \\ 0,5 & 0,8 & - & 0,5 \\ 0,2 & 0 & 0,5 & - \end{pmatrix}$$

  Estrategia de negociación: Cuadrática

- Francisco Javier:

$$\begin{pmatrix} - & 0,1 & 0,7 & 0,25 \\ 0,9 & - & 0,5 & 0,5 \\ 0,3 & 0,5 & - & 0,3 \\ 0,75 & 0,5 & 0,7 & - \end{pmatrix}$$

  Estrategia de negociación: Lineal

- Luis:

$$\begin{pmatrix} - & 0,8 & 0,55 & 1 \\ 0,2 & - & 0 & 0,7 \\ 0,45 & 1 & - & 0,25 \\ 0 & 0,3 & 0,75 & - \end{pmatrix}$$

  Estrategia de negociación: Cúbica

- Francis:

$$\begin{pmatrix} - & 1 & 0,3 & 0,75 \\ 0 & - & 0,5 & 0,3 \\ 0,7 & 0,5 & - & 0,35 \\ 0,25 & 0,7 & 0,65 & - \end{pmatrix}$$

  Estrategia de negociación: Cuadrática

6. El sistema recoge las valoraciones y las estrategias de los expertos (Iván, Francisco Javier, Francis y Luis) y las procesa.

7. El sistema almacena y muestra los resultados del proceso de consenso.

**Escenario 4: Consultar Datos del Problema**

**Nombre**: ConsultarDatosProblema.
**Descripción**: El usuario experto Iván, consulta los datos del problema en el que participó anteriormente.
**Actores participantes**: Iván (usuario Experto).
**Flujo de eventos**:

1. El usuario Iván, accede a la aplicación y selecciona la opción "Ver Mis Problemas".

2. El sistema muestra una lista con los problemas en los que Iván ha participado.

3. Iván selecciona el problema número 4.

4. El sistema muestra la información del problema número 4. Esta información es la siguiente:

5. *Descripción:* Un grupo de amigos quieren llegar a un acuerdo en la elección de un restaurante para cenar el día 28 de junio.

    - *Especialidad:* Gastronómica.
    - *Número de Alternativas:* 4.
    - *Alternativas:*

        a) Casa Antonio.

        b) La Picaña.

        c) O Mammamia.

d) Tamesis London.

- *Umbral de Consenso:* 0,75.
- *Número Máximo de Rondas:* 4.
- *Número Mínimo de Expertos:* 4.
- *Fecha de Comienzo:* 5/5/2013.
- *Estado:* Finalizado con Consenso.
- *Grado de Consenso*: 0,78

- *Número de rondas empleadas*: 3

6. Iván abandona la aplicación.

## 4.4. Diseño del sistema

Una vez que tenemos modelado el sistema, describiendo la funcionalidad del mismo de manera correcta, completa, consistente, clara y verificable, la siguiente etapa dentro de la ingeniería del software va a tratar de definir los objetivos del proyecto, los subsistemas y las estrategias en torno al sistema y los datos. Para ello, vamos a utilizar los distintos recursos que nos brinda UML como modo de representación. Además, se considerarán los distintos tipos de agente definidos en nuestro sistema, según se vio en el capítulo 3.

### 4.4.1. Diseño de objetos

Durante el diseño de objetos, se desarrolla una solución lógica basada en el paradigma orientado a objetos. Lo esencial de esta solución es la creación de los diagramas de interacción, que van a representar el modo en el que los objetos colaboran para satisfacer los requisitos. En

paralelo, se representarán los diagramas de clases de diseño. Éstos resumen la definición de las clases software (e interfaces) que se van a implementar en el software. Hay que destacar que entre los dos artefactos (diagrama de interacción y diagrama de clases), el que más importancia adquiere desde el punto de vista del diseñador es el primero ya que requiere mayor grado de esfuerzo creativo. Además, para su creación se requiere la aplicación de los principios para la asignación de responsabilidades y el uso de los principios y patrones de diseño.

#### 4.4.1.1. Diagrama de clases de diseño

Los diagramas de clases representan la estructura del sistema y se utilizan durante el análisis de requerimientos para modelar los conceptos del dominio del problema, el diseño del sistema para modelar los subsistemas e interfaces y durante el diseño de objetos para modelar clases. La figura 4.8 muestra el digrama de clases de diseño de nuestro sistema.

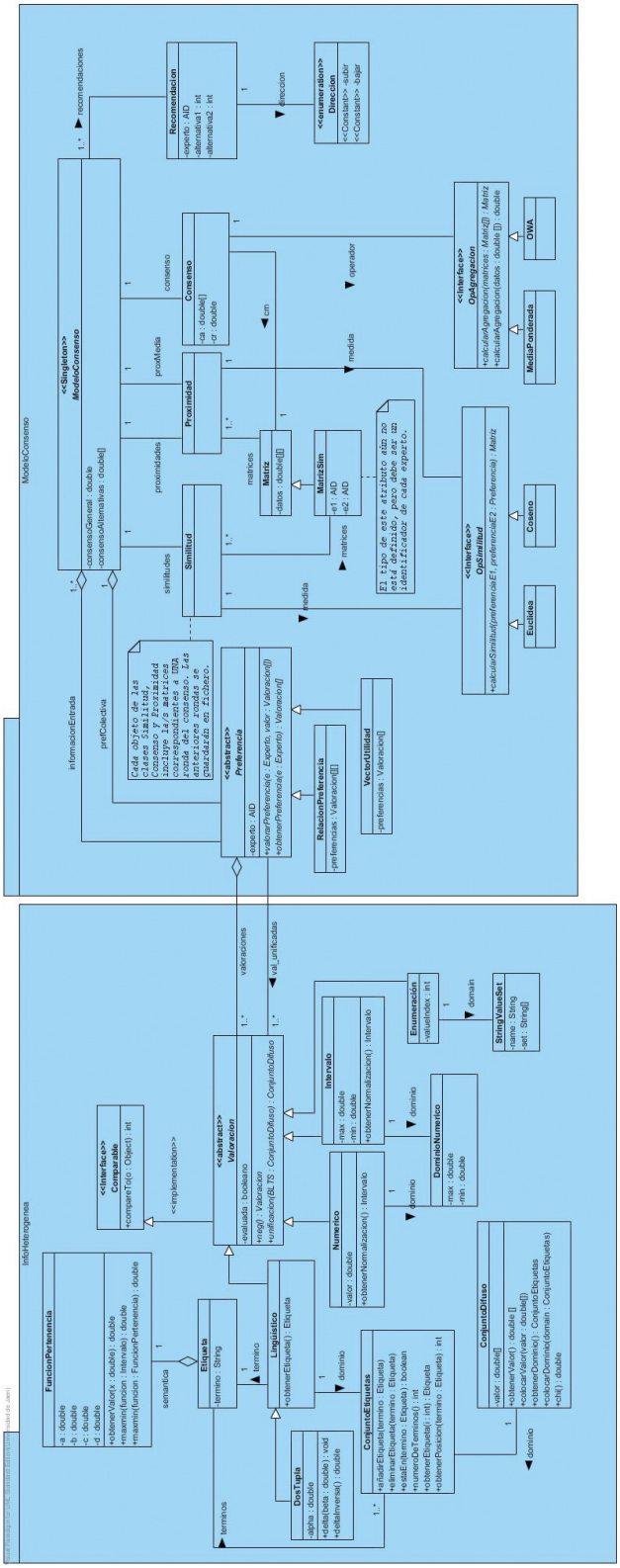


Figura 4.8: Diagrama de Clases de Diseño.

#### 4.4.1.2. Diagramas de Secuencia

Estos diagramas sirven para modelar aspectos dinámicos de los sistemas, uniendo los casos de uso con los objetos. Estas uniones describen interacciones entre los objetos y sus relaciones, así como los mensajes que intercambian.

En este apartado se van a presentar los diagramas de secuencia más importantes, en concreto los referentes al proceso de consenso en sí, en los que se indicarán los agentes que intervienen, así como el paso de mensajes entre estos.

Destacar que los diagramas de interacción mostrados en esta memoria se centran en mostrar la comunicación entre agentes pertenecientes a la plataforma de negociación, ya que es donde se produce un mayor número de comunicación entre agentes.

1. **Agente Moderador - Agente Experto.**

   **Diagrama:**(Véase figura 4.9)

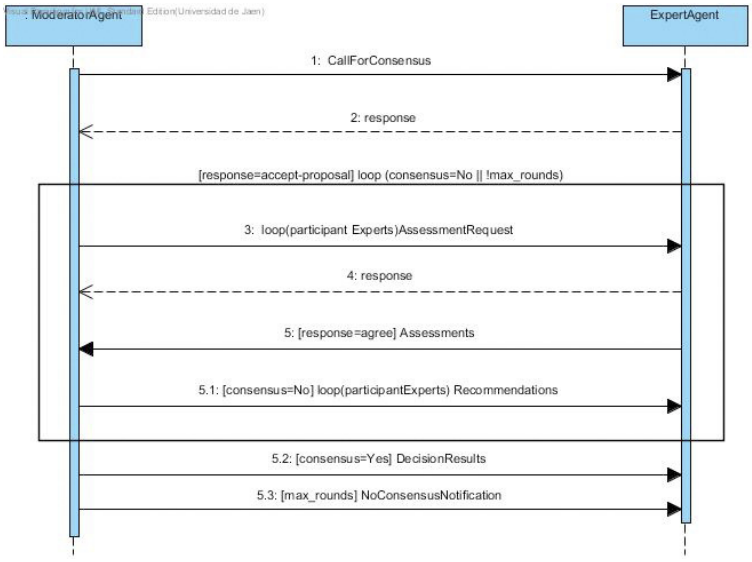


Figura 4.9: Diagrama de Secuencia Agente Moderador - Agente Experto.

**Decripción:**

1.1. El agente moderador realiza una llamada general de participación en el consenso.

1.2. El agente experto responde ante la llamada, aceptando o rechazando la propuesta.

1.3. A todos los agentes expertos que han aceptado la participación, el agente moderador les envía una solicitud de valoraciones.

1.4. Cada agente experto responde diciendo que acepta, rechaza o no entiende la petición.

1.5. Si la respuesta ha sido de aceptación, el agente experto proporciona sus valoraciones al agente moderador.

1.6. Una vez calculado el grado de consenso, si este no es lo suficientemente alto, el agente moderador envía a los expertos pertinentes las recomendaciones de cambio.

1.7. Se repiten los pasos 1.3 a 1.6 hasta que se alcance el grado de consenso deseados o se sobrepase el máximo número de rondas permitido.

1.8. Si se ha alcanzado consenso, el agente moderador informa de los resultados a los agentes expertos.

1.9. Si se ha alcanzado el máximo número de rondas, el agente moderador notifica a los agentes expertos de que no se ha alcanzado el consenso.

**2. Agente Moderador - Agente Evaluador de Consenso.**

**Diagrama:**(Véase figura 4.10)

**Decripción:**

2.1. El agente moderador realiza una llamada general de participación en el consenso.

2.2. El agente evaluador de consenso responde ante la llamada, aceptando o rechazando la propuesta.

2.3. El agente moderador envía al agente evaluador de consenso que participa en el problema una solicitud de cálculo del grado de consenso, proporcionándole para ello las valoraciones de los expertos.

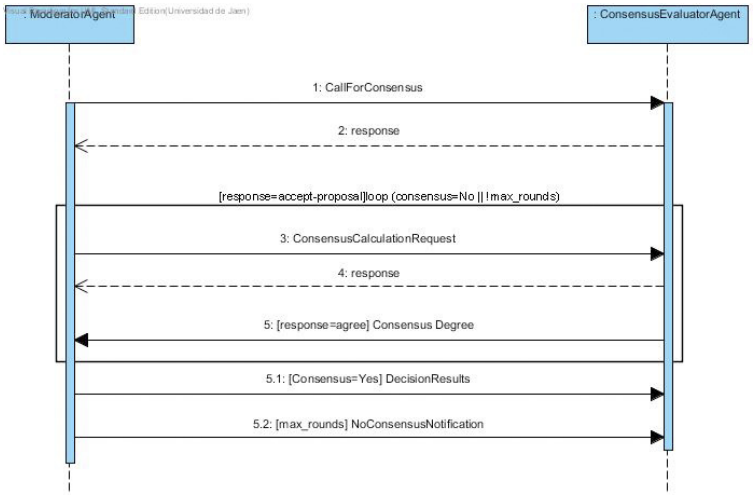


Figura 4.10: Diagrama de Secuencia Agente Moderador - Agente Evaluador de Consenso.

2.4. El agente evaluador de consenso responde diciendo que acepta, rechaza o no entiende la petición.

2.5. Si la respuesta ha sido de aceptación, el agente evaluador de consenso proporciona el grado de consenso al agente moderador.

2.6. Se repiten los pasos 2.3 a 2.5 hasta que se alcance el grado de consenso deseado o se sobrepase el máximo número de rondas permitido.

2.7. Si se ha alcanzado consenso, el agente moderador informa de los resultados al agente evaluador de consenso.

2.8. Si se ha alcanzado el máximo número de rondas, el agente moderador notifica al agente evaluador de consenso que no se ha alcanzado el consenso.

**3. Agente Moderador - Agente Identificador de Cambios.**

**Diagrama:** (Véase figura 4.11)

**Decripción:**

3.1. El agente moderador realiza una llamada general de participación en el consenso.

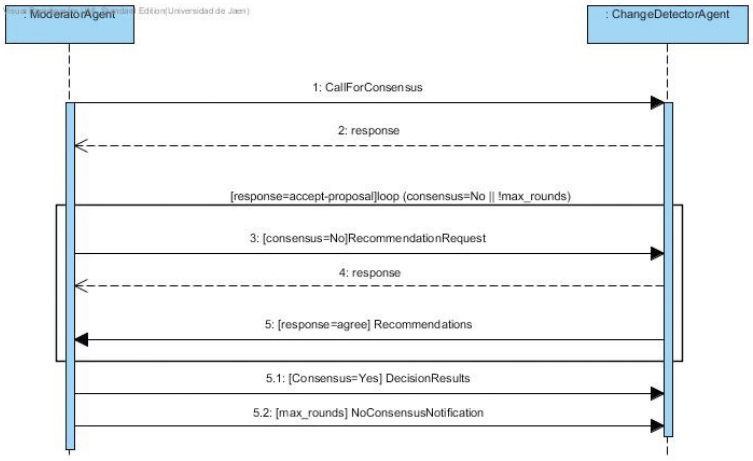


Figura 4.11: Diagrama de Secuencia Agente Moderador - Agente Identificador de Cambios

3.2. El agente identificador de cambios responde ante la llamada, aceptando o rechazando la propuesta.

3.3. Tras obtener el grado de consenso de la ronda actual, si éste no es el deseado, el agente moderador envía al agente identificador de cambios que participa en el problema una solicitud de recomendaciones de cambio para los expertos.

3.4. El agente identificador de cambios responde diciendo que acepta, rechaza o no entiende la petición.

3.5. Si la respuesta ha sido de aceptación, el agente identificador de cambios proporciona las recomendaciones de cambio al agente moderador.

3.6. Se repiten los pasos 3.3 a 3.5 hasta que se alcance el grado de consenso deseados o se sobrepase el máximo número de rondas permitido.

3.7. Si se ha alcanzado consenso, el agente moderador informa de los resultados al agente identificador de cambios.

3.8. Si se ha alcanzado el máximo número de rondas, el agente moderador notifica al agente identificador de cambios de que no se ha alcanzado el consenso.

**4. General.**

**Diagrama:** (Véase figura 4.12)

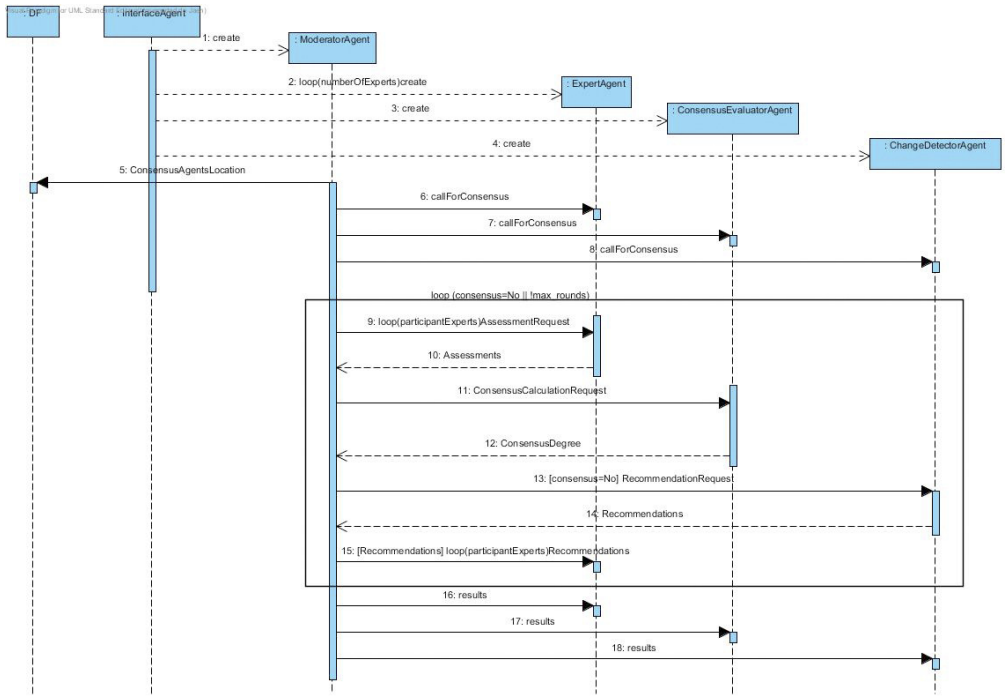


Figura 4.12: Diagrama de Secuencia General

**Decripción:**

4.1. El agente interfaz (que en nuestro caso concreto ha sido implementado mediante la clase JadeGatewayAgent, ver sec. 3.4) se encarga de instanciar a los agentes propios de un problema de consenso: agente moderador, agentes expertos, agente evaluador de consenso y agente identificador de cambios.

4.2. El agente moderador accede al servicio de páginas amarillas (DF) para localizar a los posibles agentes apropiados para iniciar el problema en cuestión.

4.3. El agente moderador realiza una llamada general de participación en el consenso.

4.4. A todos los agentes expertos que han aceptado participar, el agente moderador les envía una solicitud de valoraciones.

4.5. El agente experto que ha aceptado la solicitud envía al agente moderador sus valoraciones.

4.6. El agente moderador envía al agente evaluador de consenso que participa en el problema una solicitud de cálculo del grado de consenso, proporcionándole para ello las valoraciones de los expertos.

4.7. El agente evaluador de consenso, previa aceptación de la solicitud, proporciona el grado de consenso al agente moderador.

4.8. Tras obtener el grado de consenso de la ronda actual, y si este no es el deseado, el agente moderador envía al agente identificador de cambios que participa en el problema una solicitud de recomendaciones de cambio para los expertos.

4.9. El agente identificador de cambios, previa aceptación de la solicitud, proporciona las recomendaciones de cambio al agente moderador.

4.10. El agente moderador envía a cada agente experto sus recomendaciones de cambio.

4.11. Se repiten los pasos 4.4 a 4.10 hasta que se alcance el grado de consenso deseados o se sobrepase el máximo número de rondas permitido.

4.12. El agente moderador informa al resto de agentes participantes sobre si se ha alcanzado o no el consenso deseado.

### 4.4.2. Diseño de interfaz

Dentro del diseño del sistema se encuadra el diseño de la interfaz del sistema. Una vez que hemos representado las distintas clases de nuestro sistema y la manera de comunicación entre clases para cumplir los requerimientos establecidos en la fase de análisis, en este apartado mostramos el diseño de la interfaz del sistema, que es el medio con el que se va a comunicar el usuario del sistema con éste. En este apartado, diseñaremos las pantallas con las distintas funcionalidades del sistema y posteriormente veremos los caminos de navegación propios de

estas funcionalidades mediante los *storyboards*. Al ser una narración gráfica de una historia en cuadros consecutivos, vamos a poder indicar los caminos de navegación a partir de los resultados de las interacciones del usuario. Por último diseñaremos la guía con los estilos que utilizaremos en la interfaz.

#### 4.4.2.1. Diseño de Pantallas

**Pantalla de Acceso**

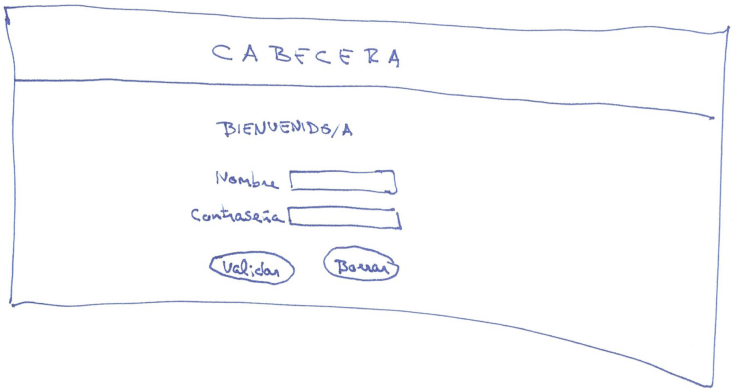


Figura 4.13: Pantalla de Acceso

### Pantalla Menú Principal del Usuario Administrador

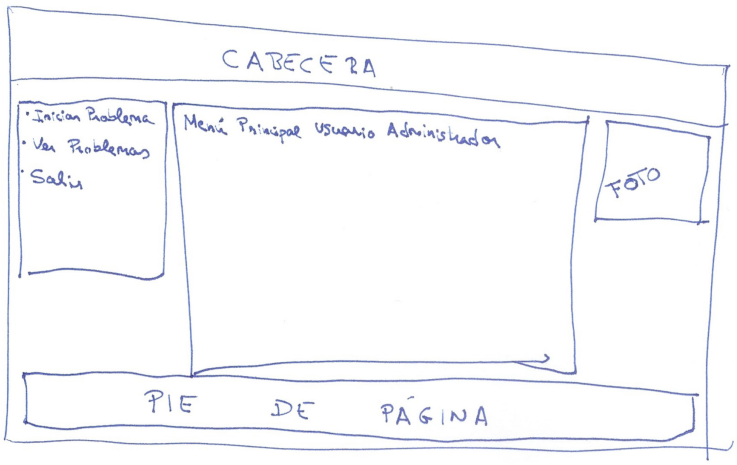


Figura 4.14: Pantalla Menú Principal del Usuario Administrador

### Pantalla Iniciar Problema

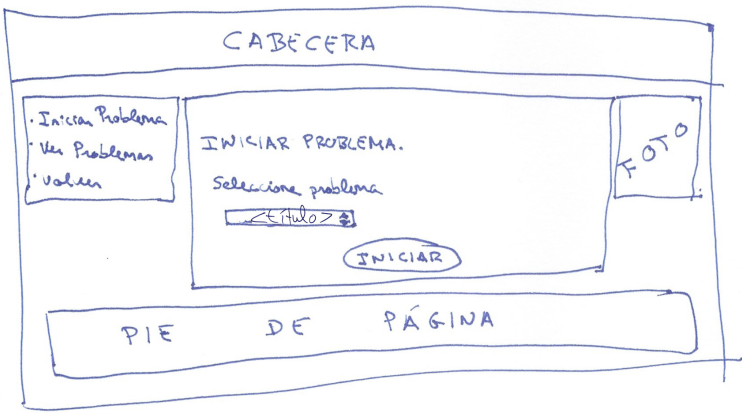


Figura 4.15: Pantalla Iniciar Problema

### Pantalla Ver Problemas

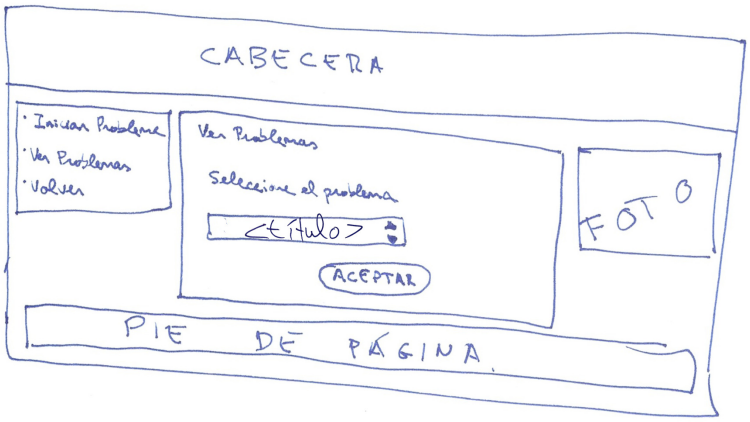


Figura 4.16: Pantalla Ver Problemas

### Pantalla Parámetros de un Problema

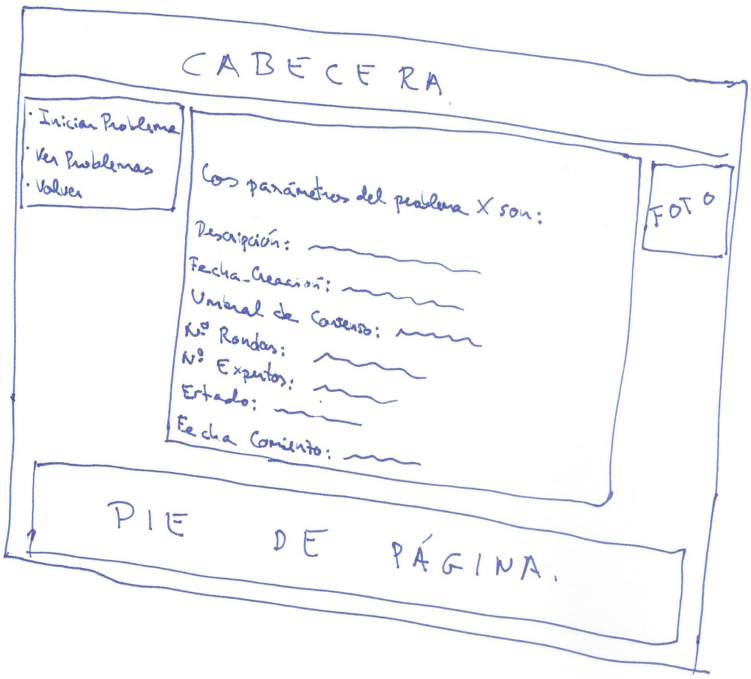


Figura 4.17: Pantalla Parámetros de un Problema

### Pantalla Menú Principal Usuario Experto

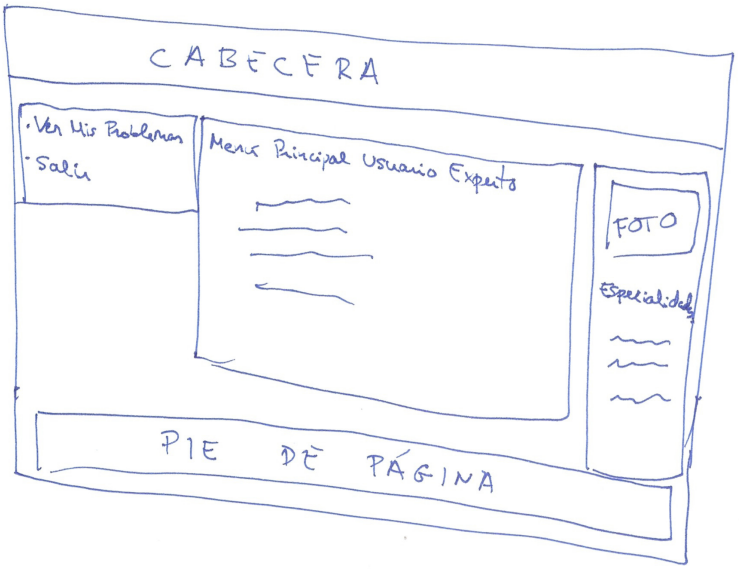


Figura 4.18: Pantalla Menú Principal Usuario Experto

### Pantalla Ver Mis Problemas

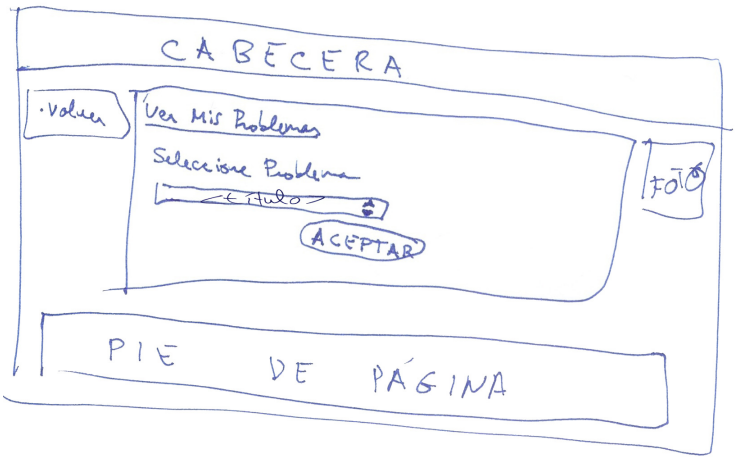


Figura 4.19: Pantalla Ver Mis Problemas

### Pantalla Ver Datos de mi Problema

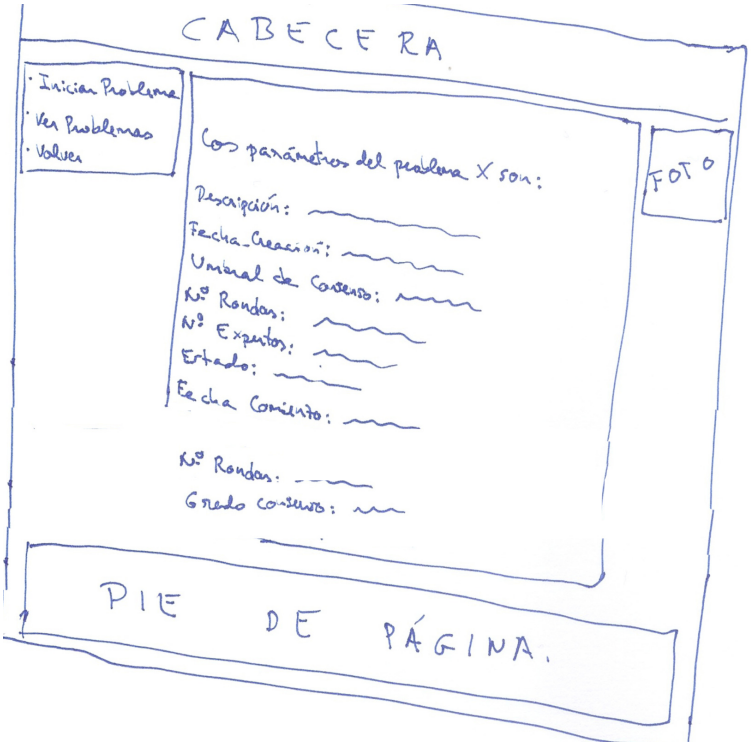


Figura 4.20: Pantalla Ver Datos de mi Problema

### Pantalla Participar en Proceso de Negociación

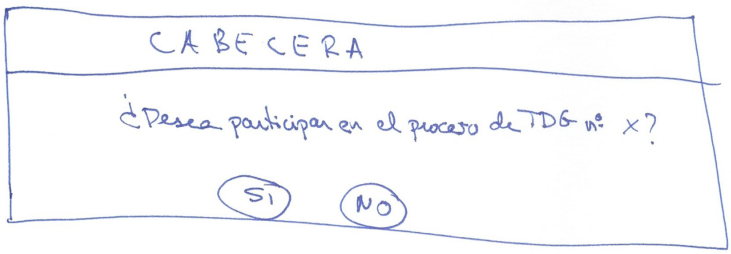


Figura 4.21: Pantalla Participar en Proceso de Negociación

**Pantalla Introducir Valoraciones**

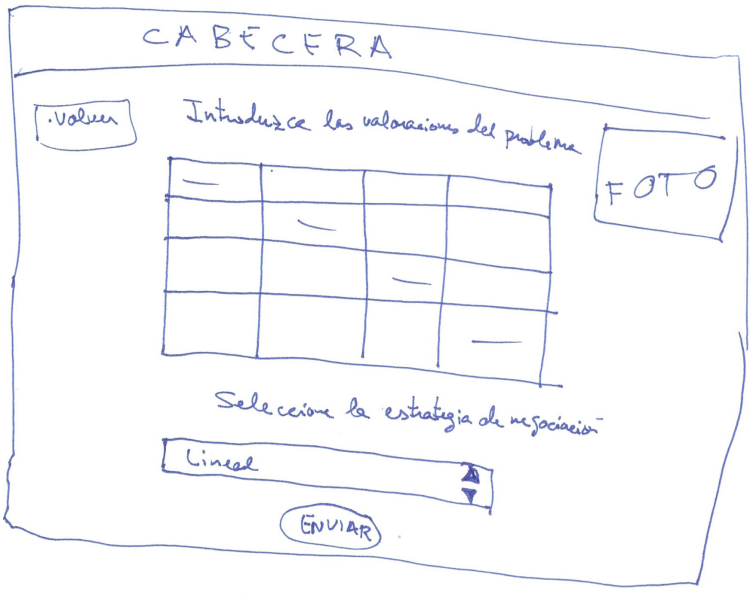


Figura 4.22: Pantalla Introducir Valoraciones

#### 4.4.2.2. Caminos de Navegación

Hasta ahora hemos visto el aspecto que tiene la interfaz de una manera estática mediante el diseño de pantallas, no obstante necesitamos diseñar la interfaz en movimiento, de manera que las acciones transcurran de forma fluida y comprensible al usuario.

Para estudiar los caminos de navegación emplearemos la herramienta conocida como *storyboard*, que consiste en mostrar, a modo de secuencia, las diferentes pantallas por las que el usuario va pasando al realizar una determinada acción sobre la aplicación.

El procedimiento es el siguiente: se sitúan capturas de pantallas en la interfaz unidas mediante flechas para indicar el camino que sigue la interacción. La posición de origen de las flechas indica qué elemento es el que desencadena el paso de una pantalla a otra.

Los storyboards sirven de prototipo para ser evaluado por el usuario y poder introducir

correcciones en fases tempranas, ya que cuanto más tiempo se tarde en validar una interfz, más coste de tiempo y trabajo acarreará.

A continuación, se muestran los storyboards para las acciones más importantes que se pueden llevar a cabo en nuestro sistema:

- Acceso usuario Administrador.
- Iniciar Problema.
- Ver Problemas.
- Salir del Menú del usuario Administrador.
- Acceso usuario Experto.
- Ver Mis Problemas.
- Participar en el Proceso de Negociación.
- Salir del Menú del usuario Experto.

### Acceso usuario Administrador

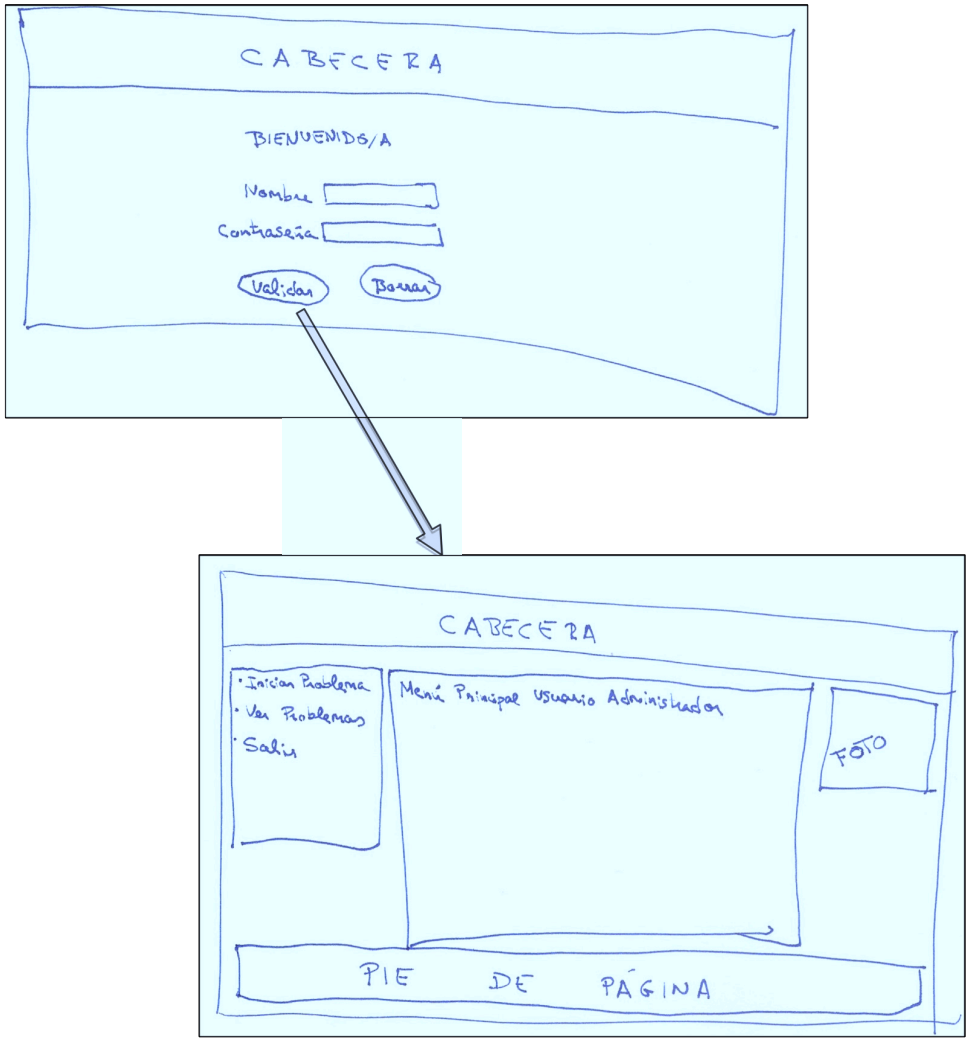


Figura 4.23: Storyboard Acceso usuario Administrador

### Iniciar Problema

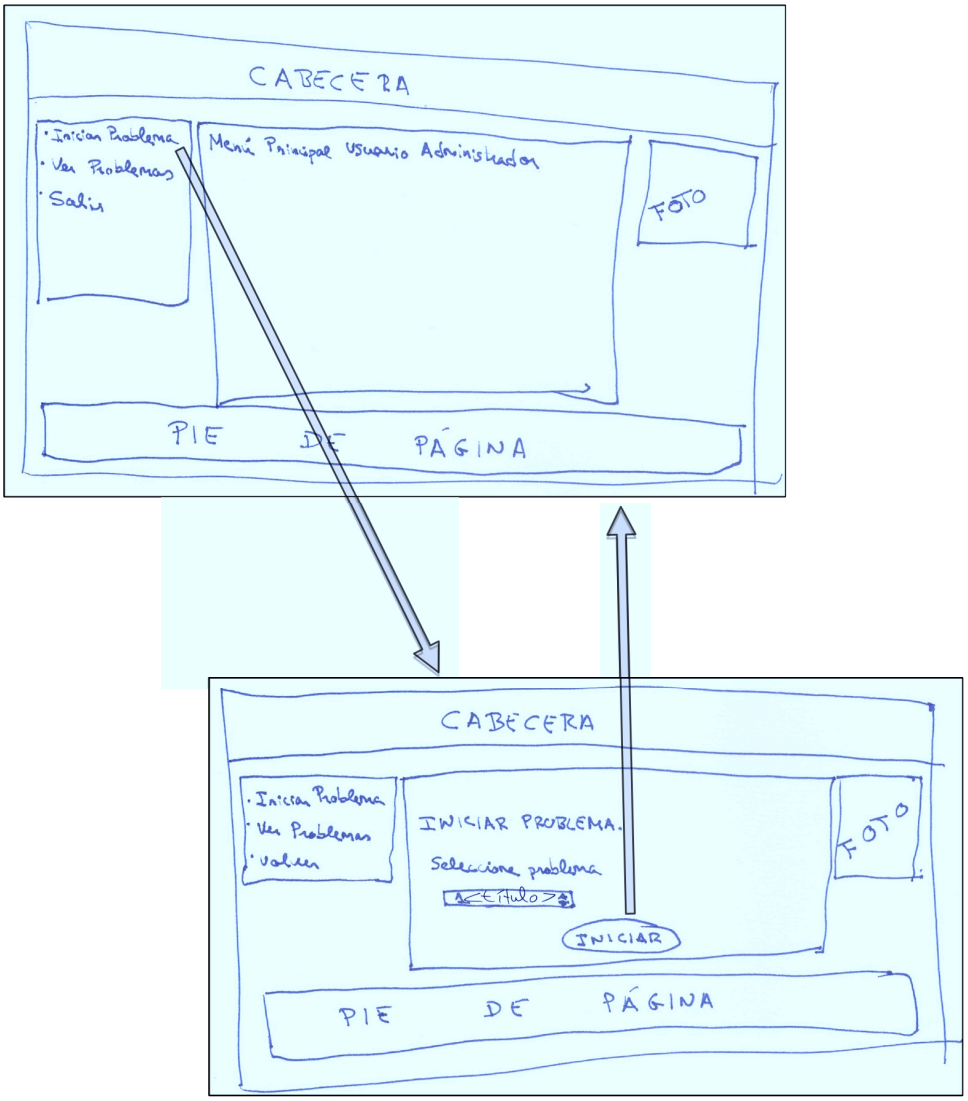


Figura 4.24: Storyboard Iniciar Problema

### Ver Problemas

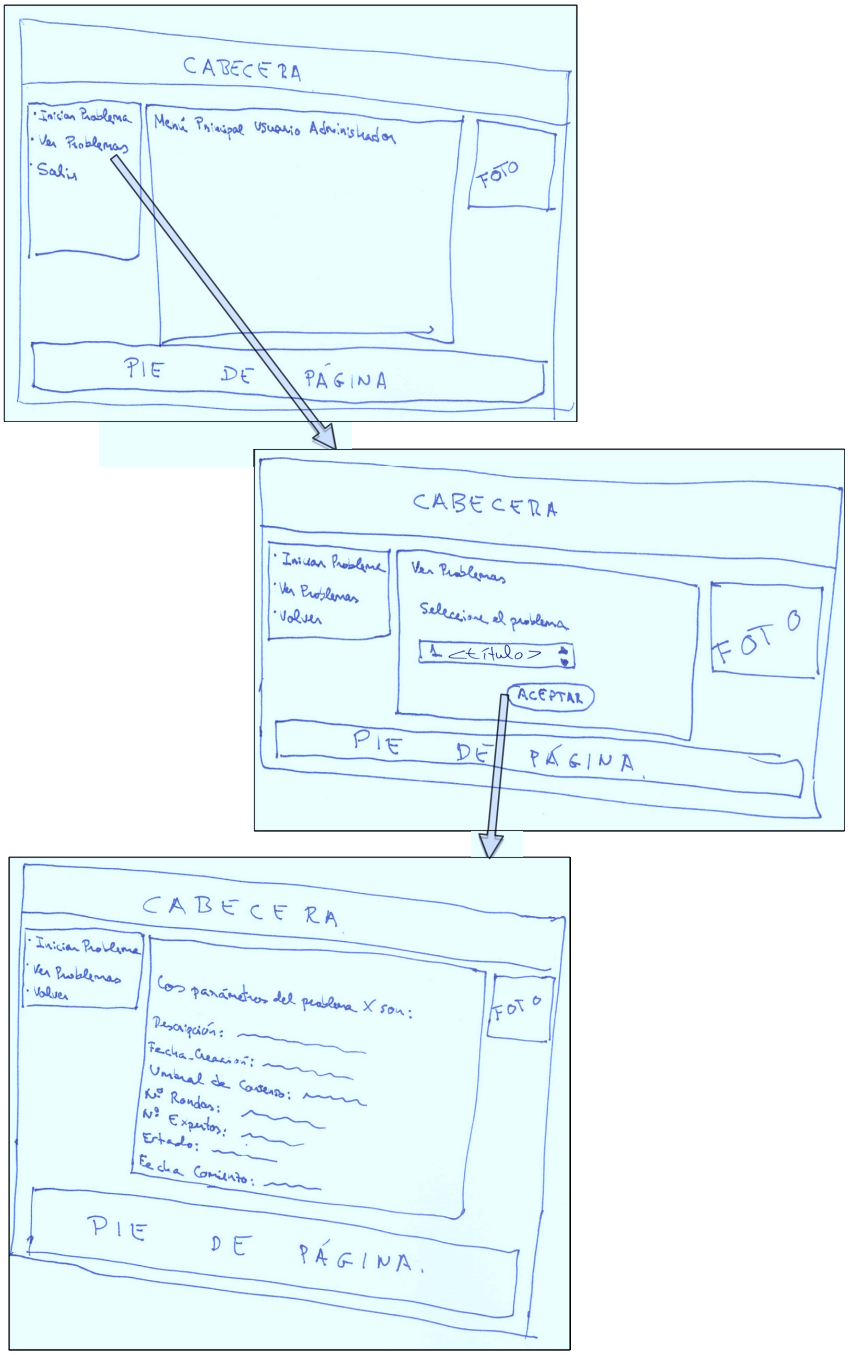


Figura 4.25: Storyboard Ver Problemas

**Salir del Menú del usuario Administrador**

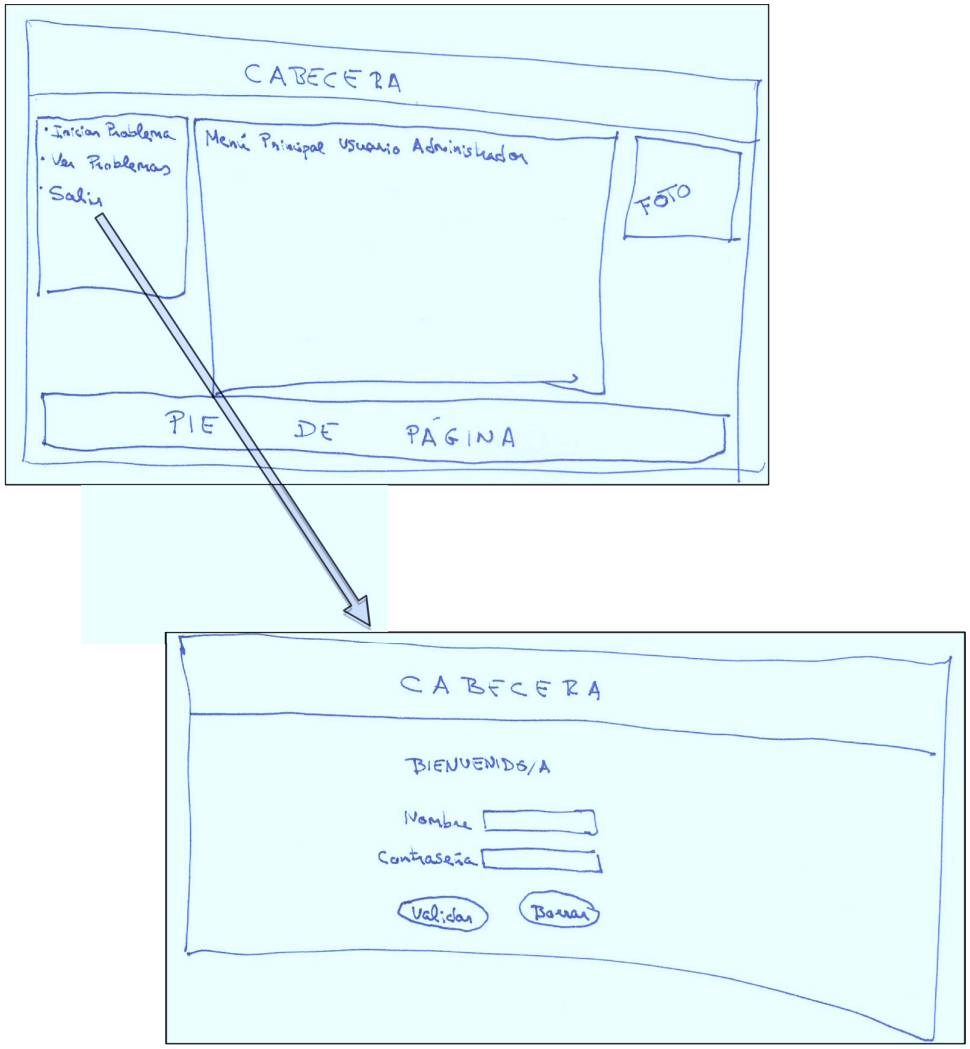


Figura 4.26: Storyboard Salir del Menú Administrador

### Acceso usuario Experto

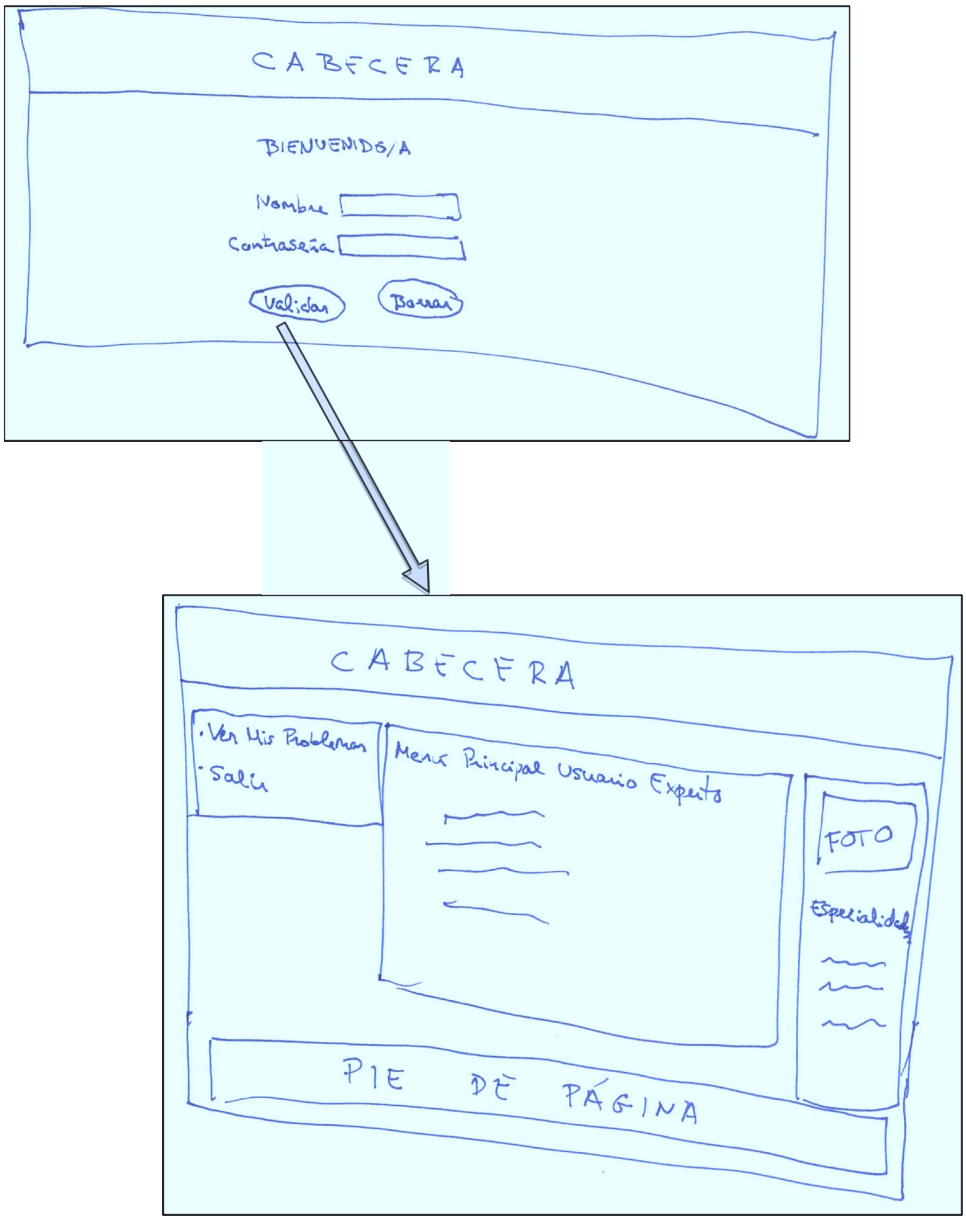


Figura 4.27: Storyboard Acceso usuario Experto

### Ver Mis Problemas

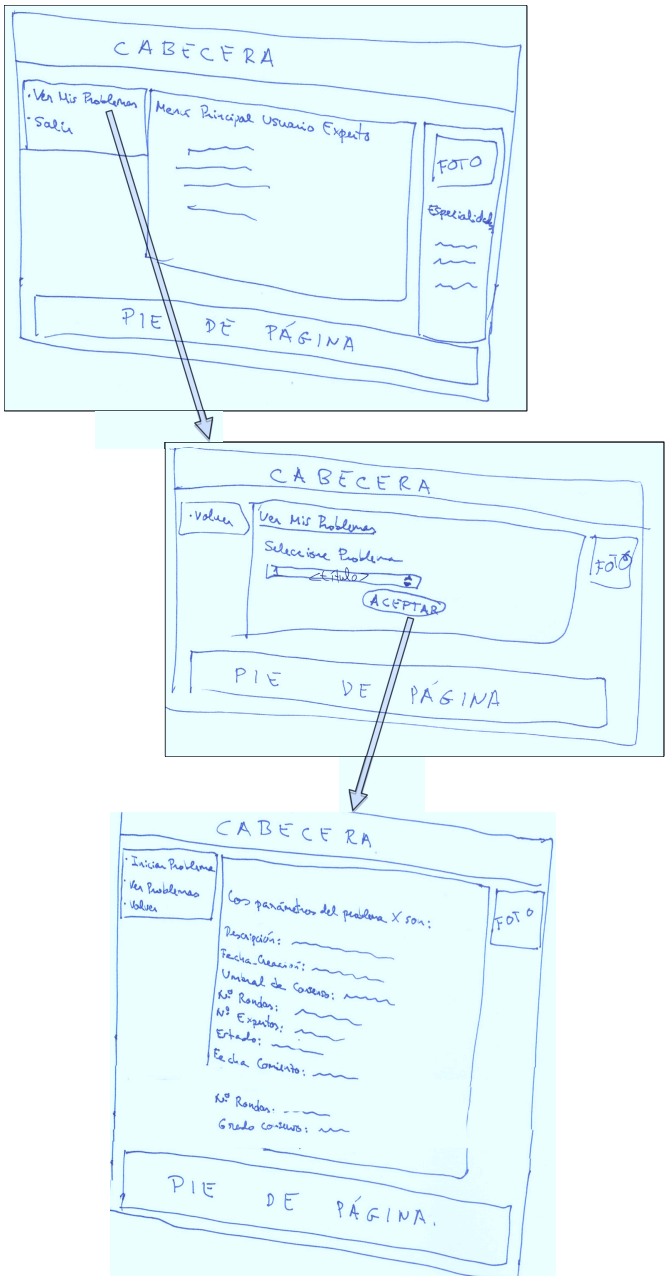


Figura 4.28: Storyboard Ver Mis Problemas

### Participar en el Proceso de Negociación

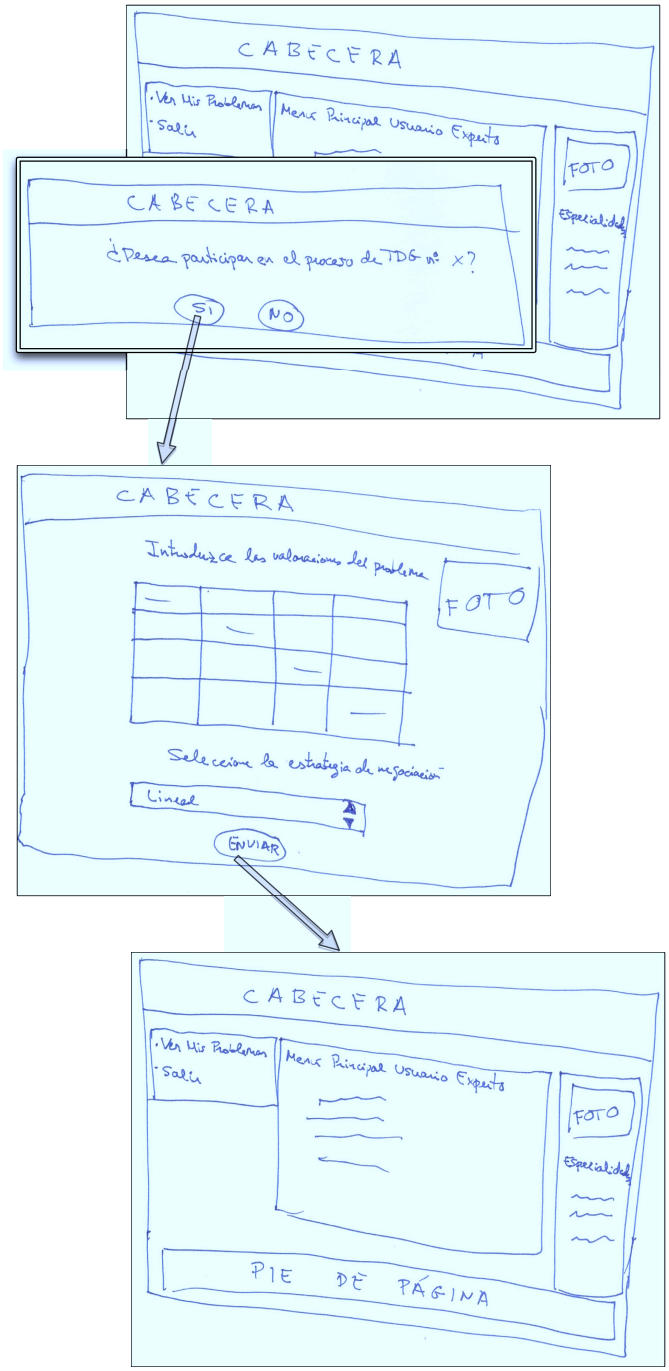


Figura 4.29: Storyboard Participar en el Proceso de Negociación

**Salir del Menú del usuario Experto**

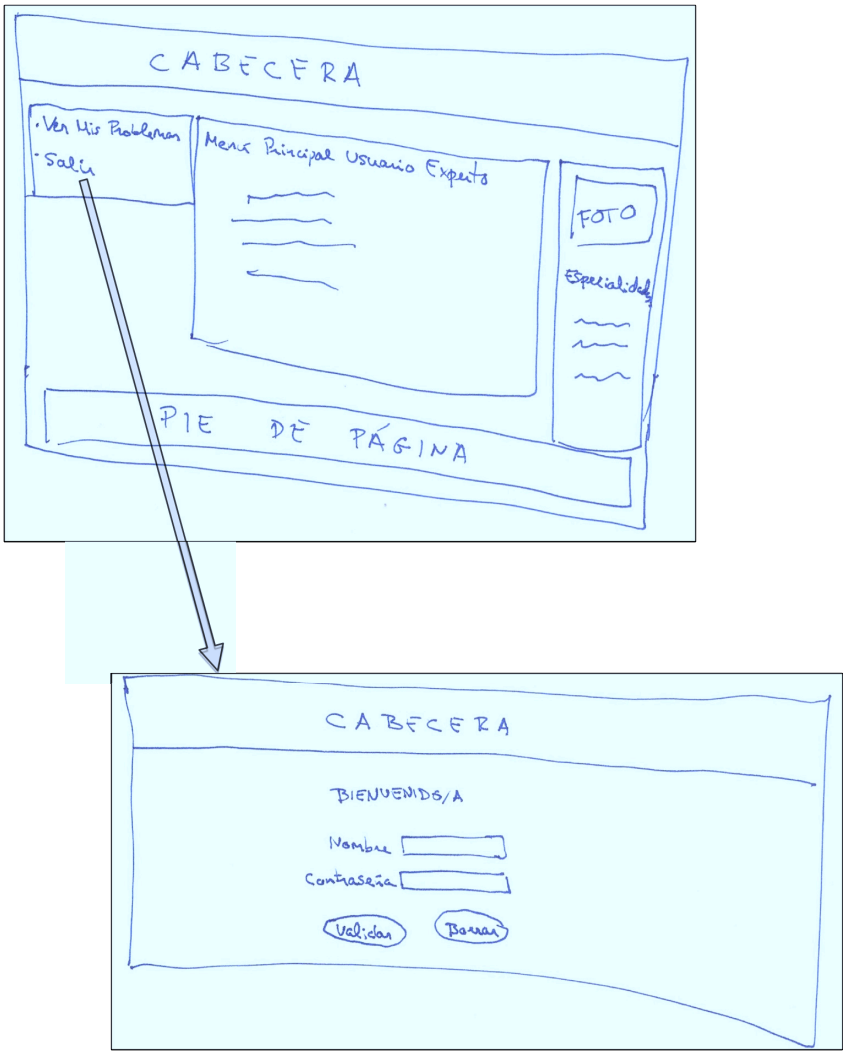


Figura 4.30: Storyboard Salir del Menú Experto

#### 4.4.2.3. Guía de Estilo

La creación de una guía de estilos cobra gran importancia en el caso en que el diseño corra a cargo de más de un diseñador, no obstante también es de gran utilidad definir dicha guía de estilos cuando sólo hay un diseñador encargado en la interfaz.

Esto es debido a varias razones:

- En ocasiones es posible que mantener la coherencia y consistencia de una interfaz, si esta es muy grande o muy ambiciosa, sea complicado incluso si sólo hay un diseñador.
- El diseñador primitivo puede abandonar el diseño. Por tanto, es de gran utilidad para sus sustitutos contar con una guía de estilo predefinida para no tener que empezar desde cero nuevamente. Esto es también aplicable cuando no es el diseñador original el que se encarga de la actualización o el mantenimiento de la interfaz.

Una vez demostrada la utilidad del uso de guías de estilo, pasamos a definir las reglas y normas que contendrán la guía de estilo de nuestra interfaz:

**Fuentes**

- *Párrafo*:
  - Tipo: Verdana, Arial, Helvetica, sans-serif
  - Color: #586885
  - Tamaño: 11px
  - Interlineado: 120
- *Título*:
  - Tipo: Verdana, Arial, Helvetica, sans-serif
  - Color: #666666
  - Tamaño: 25px
  - Alineación del texto: Izquierda
- *Subtítulo*:

- Tipo: Verdana, Arial, Helvetica, sans-serif
- Color: #486895
- Tamaño: 20px
- Formato: Negrita
- Alineación del texto: Centro

- *Apartado*:
  - Tipo: Verdana, Arial, Helvetica, sans-serif
  - Color: #666666
  - Tamaño: 18px
  - Alineación del texto: Izquierda
- *Subapartado*:
  - Tipo: Verdana, Arial, Helvetica, sans-serif
  - Color: #666666
  - Tamaño: 15px
  - Formato: Negrita
  - Alineación del texto: Izquierda
- *Tablas*:
  - Tipo: Verdana, Arial, Helvetica, sans-serif
  - Color: #586885
  - Tamaño: 12px
  - Alineación del texto: Centro
- *Enlaces activos*:
  - Tipo: Verdana, Arial, Helvetica, sans-serif

- Color: #0099CC
- Tamaño: 12px

- *Enlaces visitados*:
  - Tipo: Verdana, Arial, Helvetica, sans-serif
  - Color: #325FA0
  - Tamaño: 12px

#### Colores

Para el fondo se utilizará 3 colores, dependiendo del lugar.

- Blanco (#FFF) para la parte exterior, el marco principal y el pie de página.
- Gris (#EFEFEF) para la cabecera y los marcos laterales.
- Azul plomizo (#ACC7D9) para el contenedor principal donde están situados los marcos laterales, el pie de página y el marco principal.

### 4.4.3. Diseño de los datos

En este apartado, de la etapa de diseño del sistema, trataremos de diseñar de qué manera se va a almacenar la información de nuestro sistema en la base de datos. Para ello vamos a utilizar las herramientas de representación del modelo entidad-relación. Concretamente los esquemas conceptuales y las tablas.

#### Modelo Entidad-Relación

El modelo Entidad-Relación (E-R) es una técnica de modelado de datos que emplea diagramas entidad-relación. No es la única técnica de modelado pero sí es la más extendida

y utilizada.

Un diagrama entidad-relación está compuesto por tres tipos de elementos principales:

- *Entidades*. Son objetos (cosas, conceptos o personas) sobre los que se tiene información. Se representan mediante rectángulos etiquetados con un nombre en su interior. Llamando instancia a cualquier ejemplar concreto de una entidad.

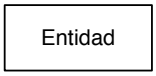


Figura 4.31: Entidad

- *Relaciones*. Interdependencias entre una o más entidades. Se representan mediante rombos etiquetados en su interior con un verbo. Si la relación es entre una entidad consigo misma se denomina reflexiva, si es entre dos entidades se denomina binaria, si es entre tres ternaria y múltiple si es entre cuatro o más entidades.

Figura 4.32: Relación

- *Atributos*. Describen características propias de una entidad o relación. Se representan mediante elipses etiquetadas con un nombre en su interior.

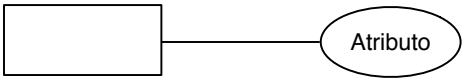


Figura 4.33: Atributo

En los diagramas E-R también hay que tener en cuenta otros aspectos como los que indicamos a continuación:

- *Entidades débiles*. Son aquellas entidades que no se pueden identificar unívocamente sólo con sus atributos, sino que necesitan estar relacionadas con otras entidades para existir. Se representan con dos rectángulos concéntricos de distinto tamaño con un nombre en el interior del más pequeño.

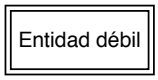


Figura 4.34: Entidad Débil

- *Cardinalidad de las relaciones*. Existen tres tipos de cardinalidades de una relación según el número de instancias de cada entidad que involucren.

  - *Uno a Uno*. Una instancia de la entidad A se relaciona solamente con una instancia de la entidad B. Se representan como 1:1.

Figura 4.35: Cardinalidad Uno a Uno

  - *Uno a Muchos*. Cada instancia de la entidad A se relaciona con varias instancias de la entidad B. Se representan como 1:*.

Figura 4.36: Cardinalidad Uno a Muchos

- *Muchos a Muchos*. Cualquier instancia de la entidad A se relaciona con cualquier instancia de la entidad B. Se representa como *:*.

Figura 4.37: Cardinalidad Muchos a Muchos

- *Clave*: Cada entidad de un diagrama entidad-relación debe tener una clave, que debe estar formada por uno o varios de sus atributos. Dicha clave es la que distingue unívocamente una instancia de la citada entidad del resto.

Figura 4.38: Clave de un único atributo

Una vez conocidos los elementos que forman parte de un diagrama entidad-relación, podemos empezar a desarrollar el modelo E-R. Los pasos a seguir son los siguientes:

1. Convertir el enunciado del problema (en nuestro caso los elementos del sistema software) en un Esquema Conceptual del mismo.
2. Convertir este Esquema Conceptual (EC) en uno más refinado, denominado Esquema Conceptual Modificado (ECM).
3. Obtener las tablas de la base de datos a partir del Esquema Conceptual Modificado.

### Normalización en el modelo Entidad-Relación

La normalización es un proceso que consiste en imponer a las tablas ciertas restricciones mediante una serie de transformaciones consecutivas. Con ello nos aseguramos de que las

tablas contengan los atributos necesarios y suficientes para describir la realidad de la entidad que representan, separando aquellos que pueden contener información cuya relevancia permite la creación de otra nueva tabla.

Para asegurar la normalización, Codd estableció tres formas normales, las cuales hacen que toda base de datos que las cumpla se considere normalizada.

Estas formas normales son:

- *Primera Forma Normal (FN1)*: Una tabla está en FN1 si todos los atributos no clave dependen funcionalmente de la clave, o lo que lo mismo, no existen grupos repetitivos para un valor de clave.
- *Segundo Forma Normal (FN2)*: Una tabla está en FN2 si está en FN1 y además todos los atributos que no pertenecen a la clave dependen funcionalmente de ella de forma completa. De esta definición se puede concluir que una tabla en FN1 y cuya clave está compuesta por un único atributo ya está en FN2.
- *Tercera Forma Normal (FN3)*: Una tabla está en FN3 si está en FN2 y además no existen atributos no clave que dependan transitivamente de la clave.

Debido a la complejidad de las tablas que obtendremos para el modelo de datos de este proyecto, no realizaremos el proceso de normalización. Por tanto, las entidades obtenidas en el Esquema Conceptual definirán las tablas de nuestra base de datos.

#### 4.4.3.1. Esquema Conceptual

Necesitamos convertir nuestros elementos de información en entidades o relaciones.

En nuestro caso las entidades serán: Usuario, Rol, Problema, Especialidad, Alternativa y Estado. Estas entidades estarán unidas por las relaciones que existen entre ellas: Participa, Se definen, Tiene y Presenta.

Una vez vistos los distintos elementos, el Esquema Conceptual quedará como se puede ver en la figura 4.39.

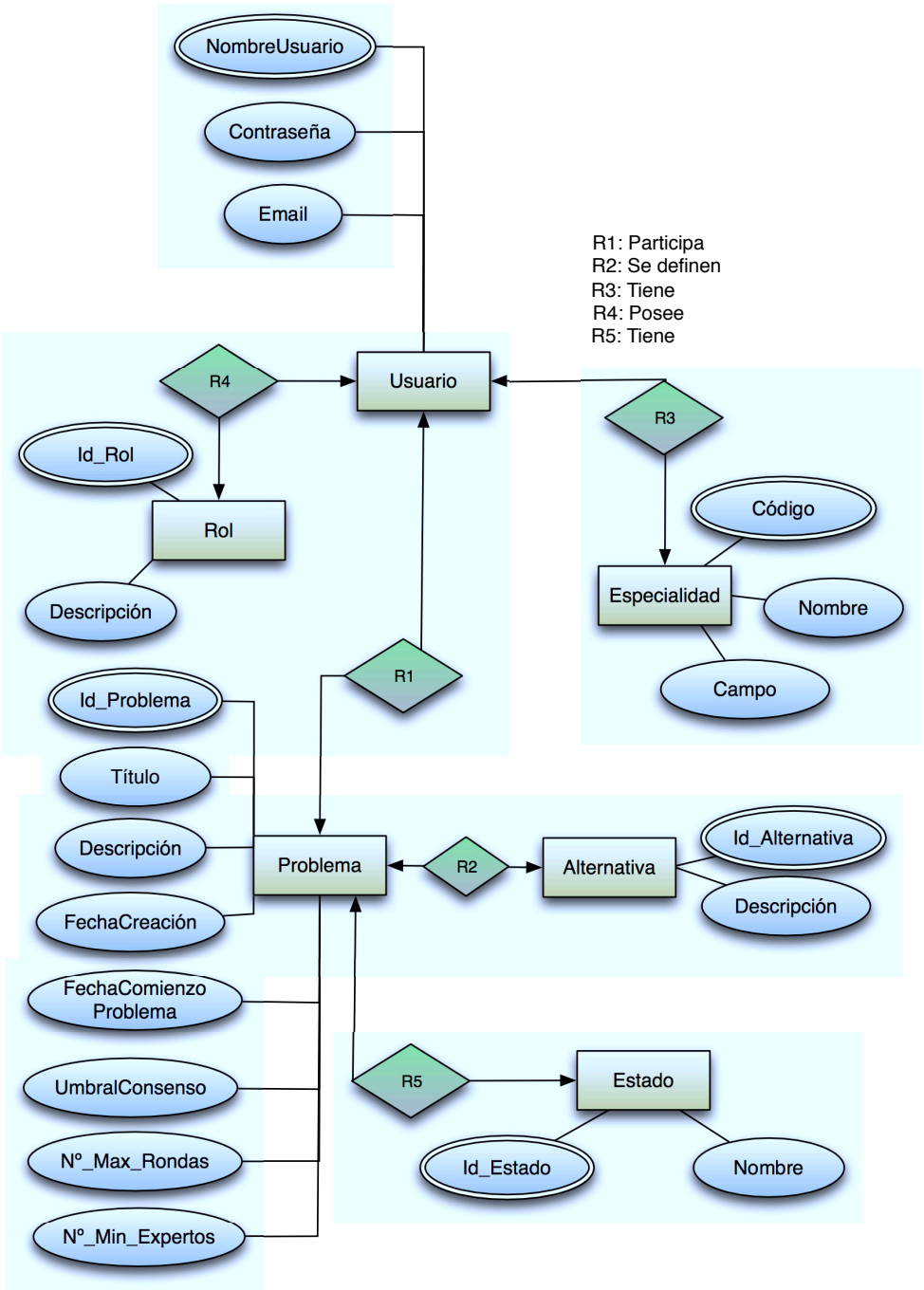


Figura 4.39: Esquema Conceptual

#### 4.4.3.2. Esquema Conceptual Modificado

Para la obtención del Esquema Conceptual Modificado a partir del Esquema Conceptual se deben hacer los siguientes cambios:

- Eliminar todas las entidad débiles.
- Eliminar las relaciones de muchos a muchos.
- Eliminar las relaciones con atributos existentes en el Esquema Conceptual.

En nuestro caso, realizando las modificaciones oportunas, obtenemos las nuevas entidades UsuarioRol, UsuarioEspecialidad, UsuarioProblema, ProblemaAlternativa y ProblemaEstado; derivadas de las relaciones muchos a muchos existentes en el EC.

Tras estas modificaciones el Esquema Conceptual Modificado quedaría tal y como muestra la figura 4.40 .

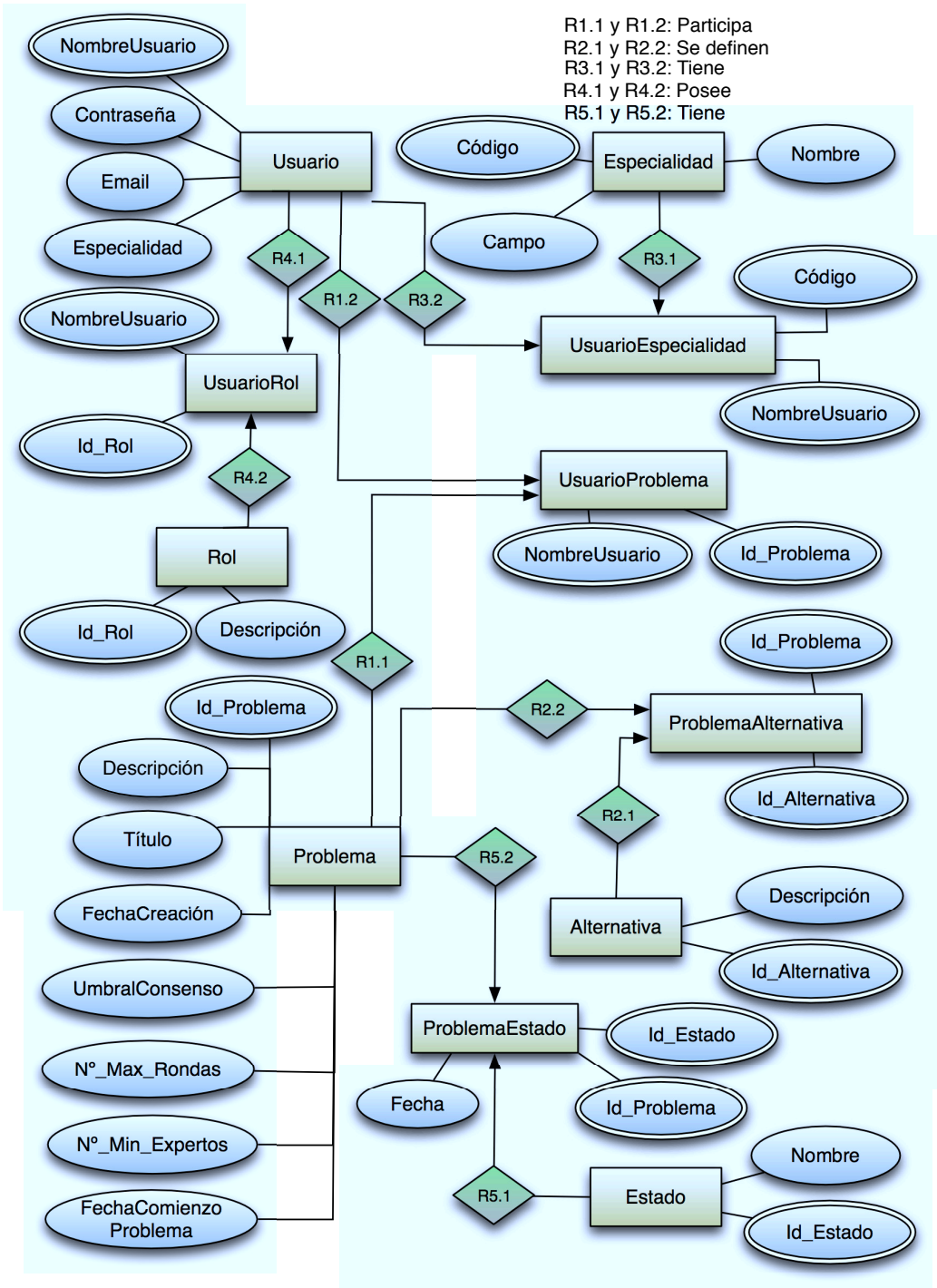


Figura 4.40: Esquema Conceptual Modificado

### 4.4.4. Diseño de la Ontología

Tras haber tratado a fondo el tema de las ontologías en el capítulo anterior, en este apartado se detallará la ontología empleada para la implementación de ésta aplicación. Desde el punto de vista de las ontologías, este proyecto se basa en la idea enunciada por Kacprzyk y Zadrozny en [80], donde se define una ontología basada en el dominio (en nuestro caso, sería una común ontología para todos los procesos de consenso) y una ontología de la aplicación (en la que se definen conceptos, predicados y acciones de agentes; es decir, es específica del problema a tratar).

En nuestro caso, se ha optado por diseñar una única ontología, pero con elementos de estas dos ontologías, de modo que se tengan en cuenta tanto los conceptos necesarios para poder llevar a cabo un proceso de alcance de consenso, como los relativos a los agentes, de manera que éstos puedan razonar sobre el conocimiento relativo al problema.

A continuación, se explicará cada componente de la ontología, los cuales están ilustrados en la figura 4.41.

**Conceptos**

- **AID**: Concepto abstracto que contiene la información básica para identificar a un agente. Contiene entre otros elementos de información el nombre de un agente y su dirección. Actualmente se utiliza como un atributo del concepto *ExpertAgent*, aunque el diseño de la ontología queda abierto para su uso por otros tipos de agente.

- **Alternative**: Representación de una de las alternativas para el problema de TDG. Estos elementos forman parte de la descripción general del problema que el agente moderador debe proponer al resto de agentes. Se compone de los siguientes atributos:

  - *description*: Atributo de tipo cadena. Contiene la descripción de la alternativa.

- **Consensus**: Contiene información sobre el nivel de consenso alcanzado en un momento

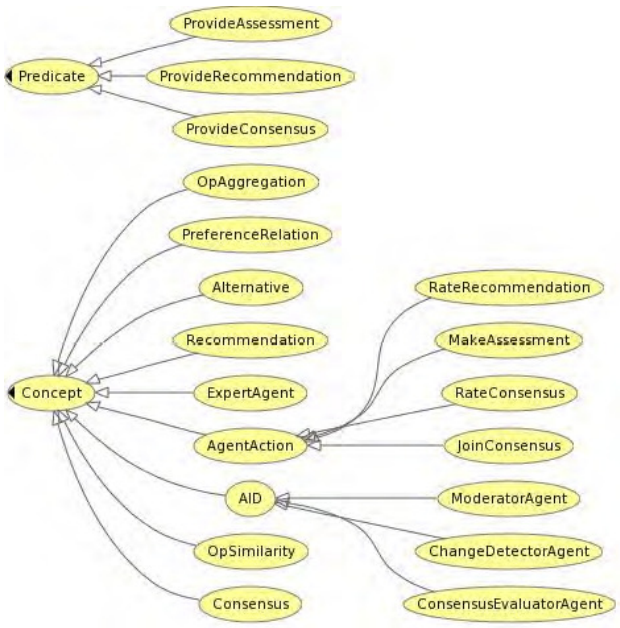


Figura 4.41: Ontología de la Plataforma de Negociación orientada al Consenso

dado. Consta de los siguientes atributos:

- *currentConsensus*: Atributo de tipo real, que contiene el grado de consenso actual.
- *consensusMatrix*: Atributo múltiple (agregado) de tipo real. Contiene los elementos de la matriz de consenso obtenida en la ronda actual.
- *consensusThreshold*: Atributo de tipo real. Contiene el umbral de consenso.

- **ExpertAgent**: Este concepto se refiere a uno de los agentes expertos participantes en el consenso. Se compone del siguiente atributo:
  - *aid*: Atributo de tipo concepto *AID*. Identificador del agente experto.
- **OpAggregation**: Operador de agregación que se utilizará durante el proceso de consenso. Se compone del siguiente atributo:
  - *aggOperator*: Atributo de tipo cadena, que indica el nombre del operador de agregación a utilizar.

- **OpSimilarity**: Especifica el operador de similitud con el que se trabajará durante el proceso de consenso. Se compone de un atributo:
  - simOperator: Atributo de tipo cadena. Contiene el nombre del operador de similitud.
- **PreferenceRelation**: Este concepto tiene como objetivo representar una relación de preferencia completa, que como sabemos es la estructura empleada por un experto para representar sus opiniones sobre un conjunto de alternativas. Contiene el siguiente atributo:
  - *assessments*: Atributo agregado de tipo real. Contiene el conjunto de valoraciones dadas por el experto a cada par de alternativas.
- **Recommendation**: Sirve para representar una recomendación de cambio individual sugerida a un experto para que éste modifique sus preferencias en vista a alcanzar un acuerdo. Consta de los siguientes atributos:
  - *aid*: Atributo de tipo concepto *AID*. Identificador del agente experto a quien es destinada la recomendación.
  - *numAlt1*: Atributo de tipo entero. Número de la primera alternativa existente en el par a modificar.
  - *numAlt2*: Atributo de tipo entero. Número de la segunda alternativa existente en el par a modificar.
  - *direction*: Atributo de tipo cadena. Dirección en la que el experto debe modificar la valoración, puede tomar los valores "Increase.º "Decrease".

**Acciones de Agente**

- **JoinConsensus**: Contiene toda la información necesaria para elaborar una propuesta de participación en un problema de TDG a resolver mediante consenso. Los atributos de que se componen son:

- *maxRounds*: Atributo de tipo entero. Indicador del número máximo de rondas permitido en el proceso, para dar una idea aproximada de la duración del mismo.
- *setOfAlternatives*: Atributo agregado de tipo concepto *Alternative*. Conjunto de alternativas existentes en el problema.
- *problemDescription*: Atributo de tipo cadena. Descripción del problema propuesto.

- **MakeAssessment**: Acción utilizada para solicitar las preferencias a los expertos. Se compone de los atributos siguientes:
  - *setOfRecommendations*: Atributo agregado de tipo concepto *Recommendation*. Conjunto de recomendaciones de cambio para los expertos. Este atributo se utiliza únicamente a partir de la segunda ronda, ya que en la primera se solicitan las valoraciones iniciales.
  - *round*: Atributo de tipo entero. Número de ronda actual.
- **RateConsensus**: Acción de vital importancia para el buen funcionamiento del sistema. Contiene toda la información de entrada necesaria para determinar el grado de consenso actual, mediante el sistema de apoyo al consenso desarrollado. Pasamos a describir cada uno de los atributos que contiene:
  - *maxRounds*: Atributo de tipo entero. Indica el número máximo de rondas permitido.
  - *numAlt*: Atributo de tipo entero. Indica el número de alternativas existentes en el problema.
  - *opSim*: Atributo de tipo concepto *OpSimilarity*. Operador de similitud empleado.
  - *consensus*: Atributo de tipo concepto *consensus*. Contiene la estructura necesaria para incluir el grado de consenso obtenido, una vez calculado.
  - *opAgg*: Atributo de tipo concepto *opAggregation*. Operador de agregación empleado.

- *setOfExperts*: Atributo agregado de tipo concepto *ExpertAgent*. Conjunto de expertos que participan en el problema, identificados a través de su agente experto correspondiente.
- *round*: Atributo de tipo entero. Número de ronda actual.
- *setOfPreferences*: Atributo agregado de tipo concepto *PreferenceRelation*. Contiene el conjunto de relaciones de preferencia, con las opiniones actuales de todos los expertos.

- **RateRecommendation**: Esta acción contiene la información necesaria para solicitar el cálculo de las recomendaciones de cambio tras un control de consenso con resultado negativo. Contiene los siguientes atributos:

  - *numAlt*: Atributo de tipo entero. Número de alternativas existentes.
  - *opSim*: Atributo de tipo concepto *OpSimilarity*. Operador de similitud empleado, de utilidad también en el cálculo de las matrices de proximidad.
  - *consensus*: Atributo de tipo concepto *consensus*. Contiene el grado de consenso a diferentes niveles.
  - *setOfExperts*: Atributo agregado de tipo concepto *ExpertAgent*. Conjunto de expertos que participan en el problema, identificados a través de su agente experto correspondiente.
  - *setOfPreferences*: Atributo agregado de tipo concepto *PreferenceRelation*. Contiene el conjunto de relaciones de preferencia, con las opiniones actuales de todos los expertos.

**Predicados**

- **ProvideAssessment**: Este predicado es utilizado por un agente experto como respuesta al recibir una acción de agente de tipo *MakeAssessment*. Consta de un atributo:

  - *preference*: Atributo de tipo concepto *PreferenceRelation*. Consiste en la relación de preferencia actual proporcionada por el experto.

- **ProvideConsensus**: Predicado que proporciona el grado de consenso recién obtenido en la ronda actual. Incluye el siguiente atributo:

  - *consensus*: Atributo de tipo concepto *Consensus*. Grado de consenso obtenido en la ronda actual.

- **ProvideRecommendation**: Este predicado proporciona el conjunto de recomendaciones de cambio generadas en cada ronda. Los atributos de que se componen son:

  - *setOfExperts*: Atributo agregado de tipo concepto *ExpertAgent*. Contiene el conjunto de expertos a los que se les debe sugerir recomendaciones.
  - *setOfRecommendations*: Atributo agregado de tipo concepto *Recommendation*. Contiene las recomendaciones generadas en la actual ronda, ordenadas según el experto a quien van dirigidas.
  - *numRecommendationsByExpert*: Atributo agregado de tipo entero. Indica cuántas de las recomendaciones en el atributo anterior pertenecen a cada experto. Dado que tanto este atributo como *setIfExperts* mantienen una relación de orden entre sí, se consigue de esta manera enviar a cada experto sus recomendaciones correspondientes.

Una vez vista la estructura de la ontología, pasamos a ver en qué etapas del proceso se utiliza cada predicado y acción de agente.

1. Acción *JoinConsensus*: Esta acción es utilizada por el agente moderador como el contenido de un mensaje ACL-Propose para invitar al resto de agentes a participar en el problema, en el comportamiento *CallForConsensus*. Un ejemplo de instancia de esta acción es el siguiente:

```
(JoinConsensus :maxRounds 10 :setOfAlternatives (sequence
(Alternative :description Marqués de Cáceres)
(Alternative :description Los Molinos)
(Alternative :description Viña Mayor)
(Alternative :description René Barbier))
:problemDescription "Spanish Wine Selection Problem")
```

2. Acción *MakeAssessment*: Esta acción es utilizada como contenido de un mensaje ACL-Request en el comportamiento *AssessmentRequest*, del agente moderador. Sus destinatarios son una serie de agentes expertos que deben decidir si aceptar o no una petición de sus preferencias para el agente moderador. Recordemos que esta acción de agente es utilizada en la segunda y siguientes rondas para proporcionar a los agentes expertos las recomendaciones de cambio, por lo que en la primera ronda utilizaremos una recomendación "vacía"(referida a un par de alternativas ficticio, (0,0)) para indicar que no existen aún recomendaciones. A continuación mostramos un ejemplo de la instancia de este predicado:

```
(MakeAssessment :setOfRecommendations (sequence
(Recommendation :numAtl2 0 :numAlt1 0 :aid (agent-identifier :name "")
:direction Increase)) :round 1)
```

3. Predicado *ProvideAssessment*: Este predicado supone la respuesta de un agente experto a la acción *MakeAssessment*, cuando este acepta proporcionar sus preferencias al moderador. Se encapsula como el contenido de un mensaje ACL-Inform en el comportamiento *MakeAssessmentHandler*. Su estructura es similar a la del siguiente ejemplo, en el que un agente experto asociado al experto $e_i$ expresa sus preferencias $p_i^{lk}$ sobre 4 alternativas (en este caso, sólo se incluyen los valores que aportan datos, excluyendo la diagonal principal):

```
((ProvideAssessment (PreferenceRelation :assessments (sequence
0.6725 0.1725 0.9375
0.32875 0.5 0.32875
0.6725 0.5 0.32875
0.0625 0.82875 0.5))))
```

4. Acción *RateConsensus*: Tras recibir las preferencias por parte de los agentes expertos, el agente moderador envía un mensaje ACL-Request al agente evaluador de consenso, cuyo contenido es una instancia de esta acción de agente. Esto se hace en el comportamiento *ConsensusCalcRequest* del agente moderador. En el siguiente ejemplo mostramos parte del contenido de esta acción:

```
(RateConsensus :maxRounds 10 :numAlt 4 :opSim (OpSimilarity
:simOperator Euclidean) :consensus (Consensus :currentConsensus
0.7089 :consensusMatrix (sequence 0.0) :consensusThreshold 0.75)
:opAgg (OpAggregation :aggOperator WeightedMean) :setOfExperts
(sequence (ExpertAgent :aid <IDENTIFICADORES DE LOS AGENTES EXPERTOS>
)) :round 3 :setOfPreferences (sequence (PreferenceRelation
:assessments (sequence 0.18375000000000002 0.18375000000000002 0.5025
0.7175 0.5525 0.7175 0.7525 0.28375000000000006 0.2525
0.5025 0.35250000000000004 0.6525))
<RESTO DE PREFERENCIAS DE EXPERTOS>))
```

5. Predicado *ProvideConsensus*: Se utiliza como contenido del mensaje ACL-Inform con el que el agente evaluador de consenso responde al agente moderador, una vez calculado el grado de consenso en la actual ronda. Este predicado es creado en el comportamiento *ConsensusCalcRequestHandler*, y su contenido sigue la estructura mostrada a continuación:

```
((ProvideConsensus (Consensus :currentConsensus 0.7631399
:consensusMatrix (sequence
-1.0 0.728705357142857 0.7678571428571429 0.758169642857143
0.7615624999999999 -1.0 0.7011160714285712 0.8450892857142857
0.7582589285714286 0.7386160714285717 -1.0 0.7760267857142856
0.7270982142857145 0.8029017857142858 0.7922767857142857 -1.0)
:consensusThreshold 0.75)))
```

6. Acción *RateRecommendation*: Si durante la etapa de control del grado de consenso, el agente moderador determina que éste no es suficiente, deberá preparar un mensaje ACL-Request destinado al agente identificador de cambios con la información necesaria para que este le devuelva un conjunto de sugerencias de cambio. El contenido de dicho mensaje es una instancia de esta acción de agente, la cual se crea y prepara para su envío en el comportamiento *RecommendationRequest*.

```
(RateRecommendation :numAlt 4 :opSim (OpSimilarity
:simOperator EuclideanDistance) :consensus (Consensus
:currentConsensus 0.7089658 :consensusMatrix (sequence
<VALORES DE LA MATRIZ DE CONSENSO> )
:consensusThreshold 0.75) :setOfExperts
<IDENTIFICADORES DE LOS AGENTES EXPERTOS>
 :setOfPreferences <PREFERENCIAS DE LOS EXPERTOS> )
```

7. Predicado *ProvideRecommendation*: Este predicado supone el contenido del mensaje ACL-Inform que el agente identificador de cambios da como respuesta al agente moderador, e incluye las recomendaciones de cambio sugeridas a los expertos. Es creado en el comportamiento *RecommendationRequestHandler* y su estructura es análoga a la del siguiente ejemplo, el cual aparece resumido debido a que este predicado puede llegar a tener una gran extensión:

```
((ProvideRecommendation (sequence 7 4 3 5 6 4 2 3)
(sequence (Recommendation :numAtl2 1 :numAlt1 0
:aid <IDENTIFICADOR DEL AGENTE EXPERTO>
 :direction Increase) <RESTO DE RECOMENDACIONES>)
(sequence (ExpertAgent <...>) <AGENTES EXPERTOS>)))
```

## 4.5. Implementación

La implementación es la última actividad en la Ingeniería del Software, donde todo el diseño realizado en las partes anteriores es concretado en código fuente. En este apartado vamos a tratar sobre decisiones tomadas a la hora de elegir el tipo de arquitectura sobre la que construir la aplicación y los lenguajes de programación utilizados.

### 4.5.1. Arquitectura del Sistema

La arquitectura elegida será una arquitectura cliente/servidor y una interfaz Web de comunicación con los usuarios. Éste tipo de arquitectura divide la aplicación en la parte del servidor, al cual se accede por medio de un software cliente, en nuestro caso un navegador Web. Una vez que ha accedido a la aplicación, el usuario realiza peticiones que son atendidas por el servidor, generando así una respuesta adecuada y comprensible para el cliente.

Una arquitectura cliente/servidor (véase la figura 4.42) elimina del usuario final de la aplicación la tarea de tener que instalar dicha aplicación en su máquina, consiguiendo además que cada usuario acceda únicamente a la información que le corresponde. Además, dado su diseño modular, esta arquitectura es fácilmente escalable y ampliable tanto en nuevos clientes como en servidores añadidos.

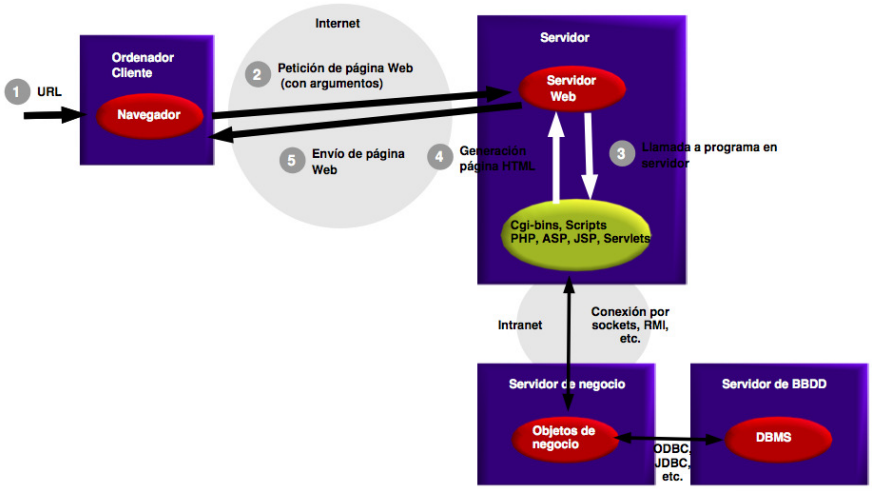


Figura 4.42: Arquitectura Cliente/Servidor

### 4.5.2. Lenguajes de programación entornos de desarrollo

Resulta obvio ante la arquitectura y el funcionamiento previsto de nuestra aplicación que el uso de HTML simple y llano no es adecuado sino que, se necesita otro lenguaje capaz de generar contenido más dinámico. Para este fin, se ha utilizado JSP.

Java Server Pages (JSP) es una tecnología de Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo.

Los JPS's permiten la utilización de código Java mediante scripts. Además, es posible la utilización de algunas acciones de JSP predefinidas mediante etiquetas. Estas etiquetas pueden ser enriquecidas mediante la utilización de Librerías de Etiquetas (TagLibs o Tag Libraries) externas e incluso personalizadas.

JSP no se puede considerar un script al 100 % ya que antes de ejecutarse el Servidor de Aplicaciones compila el contenido del documento JSP (script y etiquetas), y genera una clase Servlet. Por lo tanto, se puede decir que aunque este proceso sea transparente para el programador no deja de ser una tecnología compilada.

La principal ventaja de JSP frente a otros lenguajes es que permite integrarse con clases Java lo que permite separar en niveles las aplicaciones web, almacenando en clases java las partes que consumen más recursos y dejando la parte encargada de formatear el documento HTML en el archivo JSP. La idea fundamental de este criterio es la de separar la lógica de negocio de la presentación de la información.

Para realizar la interfaz de una manera adecuada y más atractiva para el usuario, es necesario utilizar las llamadas Hojas de Estilo(*CSS*).

CSS (Cascade Style Sheet), es un lenguaje formal que ayuda a separar la estructura interna de un documento de su presentación externa. En este proyecto se ha realizado una hoja de estilos que es invocada por los distintos archivos de la interfaz, de manera que se tiene un estilo centralizado. Esto facilita el mantenimiento de la coherencia y consistencia de la aplicación.

### 4.5.3. Herramientas de Desarrollo

Después de haber descrito los lenguajes de programación empleados para este proyecto, procedemos a especificar las herramientas empleadas para el desarrollo del mismo.

#### NetBeans

Para generar código Java necesitamos de una herramienta o entorno de desarrollo. En este caso, se ha optado por utilizar el entorno de desarrollo integrado (IDE) **NetBeans 7.2**.

NetBeans es uno de los entornos de desarrollo multilenguaje más utilizados tanto por programadores nóveles como por profesionales. Es de código abierto y fue creado por la compañía de Java, SUN. La figura 4.43 muestra el entorno NetBeans.

Entre sus principales características nos encontramos con:

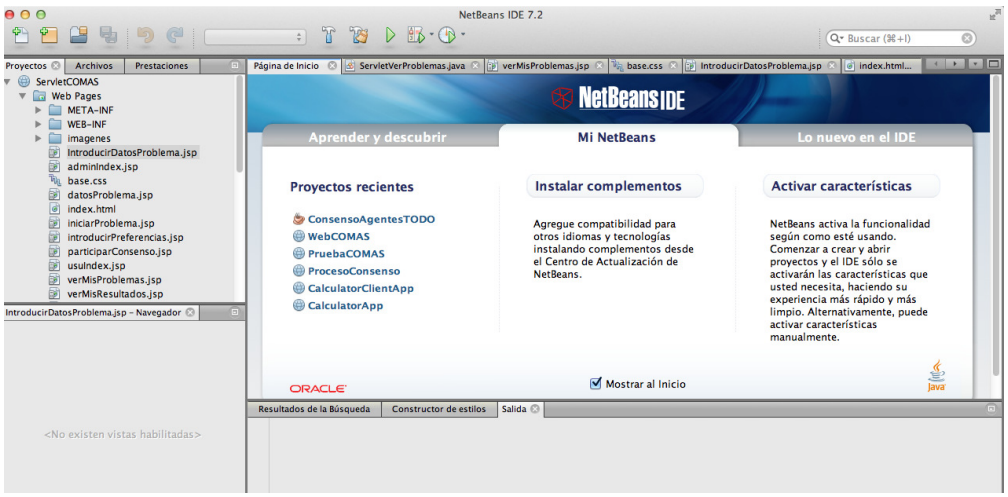


Figura 4.43: Entorno de programación NetBeans 7.2

- Integración de *Ant*. Esta herramienta permite automatizar la compilación de proyectos complejos y realizarla después pulsando solamente un botón.

- Posibilidad de refactorizar código. De este modo podemos ordenar y alinear el código mediante tabulaciones. Así como modificar el nombre de variables, aplicándose estos cambios a todo el código.

- Autocompletado de código. Con esta opción, NetBeans nos ayuda a autocomplear variables, métodos, etc. Esta utilidad ahorra bastante tiempo al desarrollador.

- Soporta el control de versiones con CSV. Así, permite trabajar con el mismo código a más de un programador, existiendo un único almacén de archivos (el sistema registra todos los cambios realizados de forma que cada programador pueda ver, resaltado en colores, qué cambios ha hecho su compañero).

### JADE

Para poder implementar el SMA hemos utilizado la plataforma JADE de la que hemos hablado largo y tendido en el apartado 2.8.2.

### Protégé

Protégé es una herramienta libre de código abierto que ha sido desarrollada en Java por la Universidad de Stanford. Es extensible y proporciona un ambiente "plug-and-play" que la convierte en una base flexible y rápida para la creación de prototipos y desarrollo de aplicaciones.

Esta herramienta encuentra su aplicación en el campo perteneciente a los sistemas de representación y procesamiento automático del conocimiento y las aplicaciones desarrolladas con ella son empleadas frecuentemente en resolución de problemas de toma de decisiones orientadas a dominios específicos.

La plataforma de Protégé se enfrenta a los procesos de modelado mediante dos modalidades principales: ProtégéOWL y ProtégéFrames.

Con Protégé se puede representar el mundo como una colección de objetos que tienen ciertos atributos, funcionalidades y relaciones con otros objetos. Los objetos son el principal elemento constitutivo de las bases de conocimiento y sirven para representar, tanto elementos físicos (entidades tangibles), como conceptos (entidades abstractas). Los objetos se relacionan entre sí para capturar las interacciones y relaciones que existen entre las entidades que representan.

La herramienta Protégé emplea una interfaz de usuario que facilita la creación de una estructura de "frames" mediante la creación de un proyecto en Protégé. Un proyecto Protégé consiste en el desarrollo de una ontología o estructura de conocimiento mediante la integración de diversos elementos: clases, slots, formularios, instancias y consultas. En Protégé, las clases representan un grupo o una colección de objetos que reúnen las mismas características atendiendo a las cuales se ha realizado la clasificación. Una instacia es un objeto específico creado a partir de una clase mediante la asignación de valores a los atributos o slots que la definen. Los formularios son la utilidad que ofrece Protégé para la introducción de información en las diferentes instancias de nuestra estructura de conocimiento. Cada clase tendrá su propio

formulario, el cual es creado inicialmente de forma automática por Protégé.

Tras ver una descripción de Protégé, pasamos a ver el plugin que hemos utilizado para exportar nuestra ontología, creada en Protégé, a clases Java.

**OntologyBeanGenerator**

OntologyBeanGenerator es un plugin que genera archivos de Java que representan la ontología a la que se le haya aplicado, de manera que ésta se pueda utilizar en el entorno JADE.

Para poder utilizar el plugin BeanGenerator se debe configurar Protégé para que lo reconozca:

- Se debe copiar el fichero *beangenerator* en el directorio de plugins de Protégé. Además de incluirse en el CLASSPATH.

- Copiar el archivo *beangenerator.properties* en el directorio raíz de Protégé.

- Para tener el esqueleto de una ontología en Protégé se debe cargar el proyecto *SimpleJADEAbstractOntology* que definirá las clases *Concept, AgentAction, AID* y *Predicate*. En caso de tratarse de un proyecto OWL se deberá cargar el proyecto *OWLSimpleJADEAbstractOntology*. Para crear los elementos de la ontología simplemente se deberán extender estas clases mediante subclases. En la figura 4.44 se puede apreciar el esqueleto de una ontología JADE:

- Este plugin, al tratarse de un tabplugin, no aparece al inicarse Protégé, con lo que será necesario acceder al menú Project, opción Configure y seleccionarlo. Una vez seleccionado aparecerá la pestaña asociada.

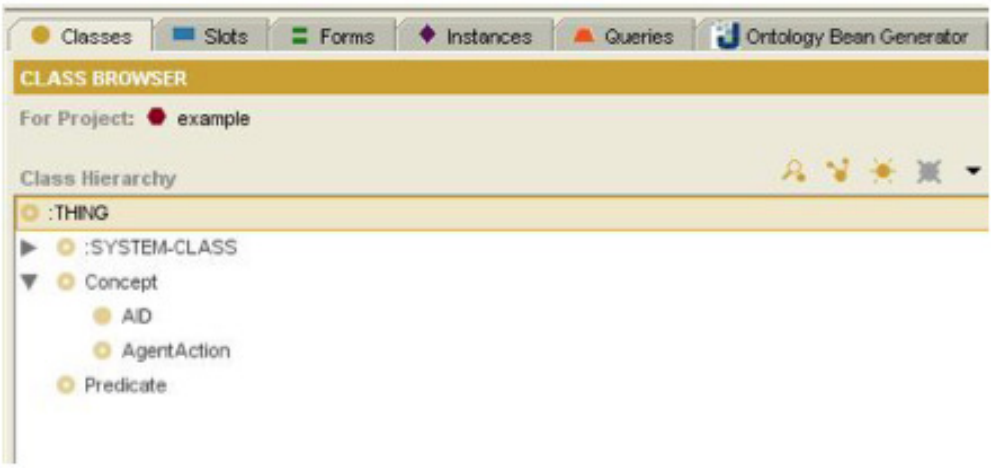


Figura 4.44: Esqueleto Ontología JADE

### PhpMyAdmin

Ésta es una herramienta escrita en PHP con la intención de manejar la administración de MySQL a través de páginas Web, utilizando Internet. Actualmente permite la creación y eliminación de bases de datos, creación, eliminación y modificación de tablas, borrar, editar, y añadir campos, ejecutar sentencias SQL, administrar claves en campos, administrar privilegios y exportar e importar datos en varios formatos. Está disponible en 50 idiomas y bajo licencia GPL.

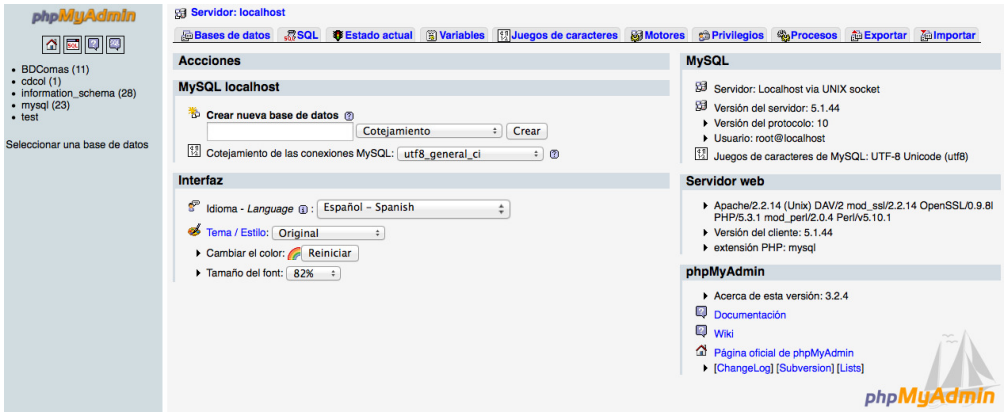


Figura 4.45: Pantalla principal de PhpMyAdmin

# Capítulo 5

# Conclusiones

## 5.1. Conclusiones finales

El propósito de este proyecto, ha sido el de implementar un prototipo de una plataforma Multi-Agente colaborativa que permitiese introducir comportamientos autónomos e inteligentes en problemas de negociación, y su aplicación como un Sistema de Soporte al Consenso para la resolución de problemas de Toma de Decisión en Grupo. Dicho propósito ha sido cumplido mediante el desarrollo y puesta en funcionamiento de dicha plataforma.

No debemos olvidar, que para poder llevar la finalización de este proyecto a buen puerto, se han tenido que superar una serie de problemas que han ido surgiendo conforme se iba avanzando en el desarrollo del mismo. En primer lugar, nos hemos tenido que familiarizar con la plataforma JADE, así como con el método de integración de la ontología. Hecho que sin la ayuda de Iván Palomares Carrascosa y de Giovanni Caire, desarrollador de la plataforma JADE, no hubiera sido posible. Una vez que tuvimos una primera versión del prototipo en modo "aplicación de escritorio", el paso hacia una aplicación web no fue nada fácil, ya que aunque en un primer momento se optó por utilizar la interacción con el usuario mediante la tecnología cliente/servidor, finalmente se vio que el correcto funcionamiento de los SMA distribuidos era mediante mediante la clase JadeGateWayAgent. Por este motivo, nos tuvimos que familiarizar con este recurso, así como con los métodos que ofrecía para

poder realizar correctamente la interacción desde la interfaz con la plataforma.

Hay que destacar, la importancia que tiene la implantación de este sistema, ya que en él, se tienen en cuenta las opiniones de cada uno de los participantes a la hora de alcanzar una solución final. Una vez expresadas las opiniones, el sistema recomendará modificar las valoraciones que generan más conflictos a la hora de alcanzar el consenso (véase Anexo A). Además, este prototipo está dirigido a todos los ámbitos sociales, de manera que se pueda usar tanto en una empresa en la que sus expertos deciden las líneas futuras de negocio, como en un grupo de amigos a la hora de elegir un sitio para ir a cenar, por mencionar algunos ejemplos.

Podemos decir, que el propósito y los objetivos del proyecto, indicados en el capítulo 1, se han alcanzado con el software desarrollado, así como con la elaboración de esta memoria. Quisiera resaltar la exhaustiva revisión realizada sobre los Sistemas Multi-Agente en general y la plataforma JADE en particular, las Ontologías, la integración con tecnologías y los Modelos de Consenso. El propósito de estas extensas revisiones, no es otro que servir de apoyo para aquellas personas, incluyendo posibles futuros alumnos, que quieran iniciarse en estos temas.

También, comentar que para diseñar y desarrollar este prototipo se han seguido las fases de desarrollo de la Ingeniería del Software. Para ello, hemos comprendido los objetivos de cada etapa y se han puesto en práctica cada una de ellas, comprobando que el resultado final ha sido satisfactorio y ha cumpliendo los objetivos que se marcaron al principio.

Finalmente, no debemos olvidar que para llevar a cabo este proyecto, se han puesto en práctica muchas técnicas y metodologías estudiadas durante los años de formación académica, al igual que se han tenido que adquirir otras para el correcto desarrollo del mismo, destacando el trabajo en equipo realizado a lo largo de estos meses.

## 5.2. Trabajos futuros

Deteniéndonos un instante, nos podemos hacer la siguiente pregunta:

*¿Qué trabajos futuros se pueden realizar con este prototipo?*

El número de posibles trabajos se hace incalculable. Y es que, al reunir tecnologías con una oferta de posibilidades tan amplia, las posibles aplicaciones son infinitas.

Centrándonos exclusivamente en la instanciación del prototipo de nuestra plataforma de negociación como Sistema de Soporte al Consenso, las mejoras pueden ir encaminadas en:

- Implementación de nuevos modelos de consenso, de modo que se puedan comparar con fines investigadores. Así, el prototipo podría usarse para probar diferentes modelos de consenso.
- Implementación de distintos operadores de agregación, medidas de similitud y sistemas de generación de recomendaciones en los modelos de consenso implementados, con el mismo propósito que en el punto anterior [109].
- Implementar mecanismos que permitan el manejo de información heterogénea, de esta manera las valoraciones de los usuarios expertos se podrían dar no sólo de manera numérica, sino con otros formatos que permitan el manejo de incertidumbre tales como información lingüística e intervalar.
- Añadir perfiles de comportamiento a los usuarios, de manera que los agentes asociados a cada usuario actúen de manera similar a como lo haría el usuario humano, aumentando así más aún el grado de autonomía de los agentes.
- Introducir aprendizaje mediante inferencia. De esta manera, conforme se vayan realizando más participaciones en problemas de consenso, los agentes pueden adquirir conocimiento, lo que podría derivar en que el usuario humano delegue al agente la

participación en el proceso de consenso (modo automático). De esta manera, mediante inferencia se podrían estimar problemas de más interés para determinados expertos, así como perfeccionar las recomendaciones generadas por el sistema..

Si por el contrario, pensamos en posibles aplicaciones del prototipo en el mundo actual, como posibles mejoras tendríamos:

- **Adaptación a sensores**. Realizando unos cambios básicos en el prototipo, de manera que cada agente experto sea un sensor y que tuviese que realizar un problema de TDG, podríamos emplear el sistema en:
  - *Control de incendios*. Dentro de un edificio donde varios sensores detectan valores alterados, se lanza el proceso de consenso y, en décimas de segundo, se toma la decisión de lanzar el plan de contingencia oportuno.
  - *Catásfrofes naturales*. Por ejemplo, una inundación en la que hay que decidir poblaciones a evacuar, vías de evacuación, modos de evacuación, etc. Mediante éste sistema, se podría tomar la solución óptima, casi al instante.
  - *Domótica*.
  - *Optimizar recursos*. Los agentes optimizan los recursos (por ejemplo, agua, presión, caudal) para producciones agrícolas y ganaderas.
- **Creación de "avatares"**. Ésta es sin lugar a dudas la aplicación más importante de las que puedo imaginar, principalmente porque ahorra tiempo y en esta época, el tiempo es un bien muy preciado. En este caso, al agente se le asignaría un perfil similar a la personalidad del usuario humano. Está claro que serían necesarios complejos mecanismos de inferencia, ya que modelar la personalidad humana es una ardua tarea, pero estos "avatares"podrían permanecer dentro de una o varias plataformas en las que interactuar. Como posibles ejemplos tendríamos:
  - *Realización de tareas*. El agente detecta (se le informa) que para mañana no hay leche, ni azúcar, ni café, ni pan de molde. Entonces, nos informa que va a hacer

un pedido de estas cosas y tras nuestra confirmación, hace el pedido. En este caso, el agente, además de saber los productos que faltan, tiene que conocer el tipo de leche que consumo, si soy alérgico a la lactosa o no, si me importaría cambiar una marca por una marca blanca, etc y así con todos los productos. Después tiene que comparar entre los distintos supermercados, para buscar los posibles productos y así hacer los pedidos de manera que el ahorro sea el máximo, sin descuidar nuestras preferencias.

- *Relaciones sociales*. Introducimos a nuestro "avatar" en una plataforma y mientras nosotros estamos haciendo otra tarea, él se encarga de entablar relaciones sociales. Así, pasado un tiempo tendremos un listado de posibles candidatos, los cuales, una vez recomendados por un sistema de recomendación, se han probado virtualmente y el grado de afinidad es alto.

- *Negocio*. En este caso, situamos a nuestro "avatar" en una plataforma donde haya agentes de tiendas. De entre todas las ofertas que le hacen a nuestro agente, él nos hará llegar sólo las de nuestro interés. De esta manera se consigue una publicidad selectiva.

- **Agente de Bolsa**. Con la potencia de los SMAs, así como con la inferencia adquirida, de las correlaciones entre los valores de los distintos mercados, se podrían formar un grupo de expertos inversores, los cuales de manera autónoma realizarían comprar y ventas de acciones para optimizar la ganancia.

- **Minería de datos**. Mediante el uso de SMA se pueden realizar procesos de minería de datos, aprovechando los recursos de máquinas distribuidas, posteriormente se negociarían las mejores dependencias de datos para tomar medidas al respecto.

Se han enumerado posibles mejoras y aplicaciones que se le podrían dar a este prototipo, algunas de ellas pareciendo sacadas del cine de ciencia ficción, no obstante quisiera acabar con unas reflexiones realizadas en el vídeo promocional del nuevo MacBook Pro Retina:

*“ Para crear algo verdaderamente nuevo, tienes que volver a empezar. Y si te esfuerzas de verdad, consigues desconectar del pasado.”*

**Jony Ive** Vice-Presidente Senior de Diseño de Apple.

*“Si nunca cambias nada, sólo podrás avanzar poco a poco. Pero si estás dispuesto a cambiar, las posibilidades en el mundo del diseño son infinitas.”*

**Bob Mansfield**. Vice-Presidente Senior Ingeniería Hardware de Apple.

## Apéndice A

# Procesos de Alcance de Consenso

## A.1. Introducción

Tomar una decisión consiste en elegir la mejor opción entre un conjunto de alternativas posibles [34, 74, 120, 126] . A menudo nos enfrentamos a situaciones en las que debemos decidir qué alternativa tomar en función del entorno en el que nos encontramos.

Los problemas clásicos de decisión presentan los siguientes elementos básicos:

1. Uno o varios objetivos por resolver.
2. Un conjunto de alternativas o decisiones posibles para alcanzar dichos objetivos.
3. Un conjunto de factores o estados de la naturaleza que definen el contexto en el que se plantea el problema de decisión.
4. Un conjunto de valores de utilidad o consecuencias asociados a los pares formados por cada alternativa y estado de la naturaleza.

Dado que en la vida real se puede presentar una enorme variedad de problemas de decisión, la Teoría de la Decisión ha establecido una serie de criterios para clasificarlos bajo diferentes puntos de vista:

1. Según el número de criterios o atributos que se han de valorar para cada alternativa. Podemos tener problemas de toma de decisión de un solo criterio y multi-criterio [120, 140, 146].

2. Según el ambiente de decisión en el que se han de tomar las decisiones. Se definen problemas de toma de decisión en ambiente de certidumbre, riesgo e incertidumbre [89, 93].

3. Según el número de expertos. Tenemos problemas de toma de decisión individual y de toma de decisión en grupo (TDG) [79, 81, 93].

En este proyecto fin de carrera, nos centraremos en problemas de decisión bajo incertidumbre donde participan varios expertos, más concretamente, en problemas de TDG. Por ello, en el siguiente apartado, revisaremos las principales características de los problemas de TDG. Por ello, en el siguiente apartado, revisaremos las principales características de los problemas de TDG.

## A.2. Problemas de Toma de Decisión en Grupo (TDG)

La toma de decisiones en grupo implica la participación de varios decisores que han de tomar decisiones de manera colectiva, de cara a alcanzar una solución común a un problema. Un proceso de toma de decisión en el que participen varios individuos o expertos, donde cada uno de ellos aporta sus propios conocimientos y experiencia, dará como resultado, en ciertos contextos, una decisión de mayor calidad que aquella aportada por un único experto.

Formalmente, un problema de TDG se caracteriza por:

- La existencia de un problema o cuestión común a resolver.
- Un conjunto de *alternativas* o posibles soluciones entre las que escoger.

Figura A.1: Fases del proceso de selección en problemas de TDG

$$X = \{x_1, x_2, ..., x_n\}(n \geq 2)$$

- Un conjunto de individuos o *expertos* que expresan sus juicios, opiniones o preferencias sobre el conjunto de alternativas y que tienen la intención de alcanzar una solución en común al problema planteado.

$$E = \{e_1, e_2, ..., e_m\}(m \geq 2)$$

La solución a un problema de TDG se puede obtener aplicando un *enfoque directo* o bien un *enfoque indirecto*. En un enfoque directo, la solución se obtiene a partir de las preferencias individuales de los expertos (sin obtener una opinión social o general antes de la resolución del problema), mientras que en un enfoque indirecto dicha solución se consigue determinando *a priori* una opinión social, y empleando dicha opinión para la obtención de la solución [74]. Tal y como se observa en la figura A.1, en ambos enfoques el proceso general para alcanzar una solución al problema de TDG se compone de dos fases [96]:

- *Fase de Agregación*: Se combinan las preferencias de los expertos.
- *Fase de Explotación*: Consiste en obtener una alternativa o un subconjunto de alternativas que den solución al problema de toma de decisión.

Dependiendo del problema al que nos enfrentamos, existen situaciones en las que todos los individuos participan para tomar la decisión, y otras en las que el proceso de decisión sólo debe concernir a un decisor o un pequeño grupo de éstos, de ahí que puedan surgir inconvenientes cuando un grupo de individuos participa en un proceso de decisión. Así, podemos encontrarnos en situaciones de colaboración entre expertos, de competitividad entre expertos, propuestas compatibles y propuestas incompatibles con uno o más expertos, e incluso propuestas que involucren a diferentes entornos (por ejemplo, entre compañías, gobiernos, etc). Por esta razón, existen diferentes criterios que ayudan a resolver problemas de toma de decisión en grupo, basados en diferentes reglas para obtener la solución [41]:

- *Regla de la Mayoría*: Se toma la decisión teniendo en cuenta la opinión de la mayoría de individuos que componen el grupo envuelto en el problema de toma de decisión. Una vez adoptada la decisión de la mayoría, ésta debe ser respetada por las minorías del grupo, por lo que éstas no deben oponerse a la misma, ya que se asume que todos aceptan el uso de la regla. La noción de mayoría admite dos grandes modalidades de aplicación de la regla.

  1. *Mayoría absoluta*, cuando la opinión mayoritaria ha sido tenida en cuenta por más de la mitad del total de expertos.
  2. *Mayoría relativa o simple*, cuando solamente se requiere que la opinión mayoritaria haya sido la más numerosa en cuanto a expertos se refiere, aunque la suma del resto de expertos la supere.

- *Regla de la Minoría*: Se delega la toma de la decisión a un subgrupo de personas, ya que el problema requiere un nivel de experiencia que solamente presentan dichas personas. Es necesario que todos los expertos participantes acepten la regla y, por consiguiente, estén de acuerdo con delegar la toma de decisión al subgrupo acordado.

- *Individual*: Esta situación se presenta cuando el grupo recurre a un experto para tomar la decisión o cuando existe un líder en el grupo.

- *Unanimidad*: Todos los miembros deben estar de acuerdo con la decisión tomada.

En todas estas situaciones se presenta el problema de que algunos expertos consideren que sus opiniones no han sido tenidas en cuenta lo suficiente, lo que en ocasiones implica que no estén de acuerdo con la decisión tomada. Además, existen situaciones en las que es necesario un alto nivel de acuerdo entre los expertos participantes. Por esta razón, surge la necesidad de emplear enfoques basados en *consenso*, que añaden una nueva fase al proceso de TDG con el objetivo de alcanzar un acuerdo global entre todos los expertos. En el siguiente apartado se abordará el concepto de consenso, dando una visión general del mismo.

## A.3. El Consenso en Problemas de TDG

La Real Academia de la Lengua Española [15], define el término consenso como el *acuerdo producido por consentimiento mutuo entre todos los miembros de un grupo o entre varios grupos*. Esta definición supone la idea de un proceso de TDG en el que ningún experto está en desacuerdo sobre las decisiones tomadas, aunque algunos expertos pueden seguir opinando que su solución individual fuera mejor que la finalmente tomada. Para conseguir el acuerdo es necesario, pues, que todos los expertos cambien sus opiniones iniciales, tendiendo a aproximarlas hacia una opinión que consideren satisfactoria.

A pesar de que la fase de consenso se introduce en los procesos de TDG para incrementar el nivel de acuerdo entre los expertos que participan en el problema de decisión, en ocasiones el concepto de consenso causa cierta controversia, ya que puede ser interpretado de distintas formas, desde un total acuerdo (unanimidad) a una interpretación más flexible. El consenso por unanimidad a veces puede haberse alcanzado mediante intimidación u otras circunstancias en las que el acuerdo alcanzado no es verdadero (*consenso normativo*), razón por lo cual se ha mostrado a menudo como un método poco realista para tomar decisiones reales. El consenso normativo implica en la práctica una imposición *a priori* desde el exterior del grupo que los miembros del mismo asumen sin discutir.

El consenso no debe entenderse como un acuerdo unánime, sino como un proceso en el que la decisión finalmente tomada posiblemente no coincida totalmente con las posiciones iniciales de los expertos. Esta concepción del consenso se conoce como *Consenso Cognitivo* [98], e implica que los expertos modifican sus opiniones iniciales tras una serie de rondas de discusión y negociación. En la literatura podemos encontrar diferentes ideas de *consenso* que intentan "suavizar" la versión rígida de consenso como unanimidad. Algunas de ellas son [18, 74, 131]:

- Todos los miembros del grupo apoyan la decisión tomada.
- Ningún miembro del grupo se opone a la decisión tomada.
- Todos los miembros del grupo asumen la decisión tomada, aunque no la apoyen.
- Estado de mutuo acuerdo entre los miembros del grupo, y donde las opiniones cruciales de los individuos han sido consideradas para satisfacción de estos.

Uno de los enfoques más aceptados para "suavizar" la noción rígida de consenso como unanimidad es el de "*Soft Consensus*", propuesto por Kacprzyk [18, 74, 76, 77], y que profundiza en el estudio del concepto de mayoría en problemas de TDG, proponiendo suavizarlo mediante otro más flexible, denominado "mayoría difusa", que introduce el uso de cuantificadores lingüísticos difusos, y donde el concepto de "soft consensus" se entiende por: *La mayor parte de los expertos están de acuerdo en las alternativas importantes.* Nótese que la definición de "*Soft Consensus*" está basada en la teoría de conjuntos difusos [83]. Este enfoque ha proporcionado muy buenos resultados en diferentes entornos de TDG [53, 63, 145], y supondrá la visión del consenso considerada en el modelo de consenso propuesto en este anexo.

En definitiva, el consenso es un área de investigación de gran importancia en el campo de la toma de decisión en grupo, y son muchos los autores que han propuesto modelos que justifican la necesidad de llevar a cabo un proceso de consenso previo a la selección de alternativas en este tipo de problemas [32, 33, 37, 53, 63, 64, 78, 99, 131].

### A.3.1. Procesos de Consenso

El consenso en sí es un objetivo, y para alcanzar dicho objetivo es fundamental realizar un proceso que nos conduzca a él. En esta sección estudiaremos en qué consiste un proceso de consenso, y conoceremos las ideas generales de aquellos procesos basados en la noción de "*Soft Consensus*", indicando las fases y esquema de actuación que caracteriza a dichos procesos.

El principal propósito de los procesos de consenso consiste en alcanzar un nivel de acuerdo mínimo antes de iniciar el proceso de selección de alternativas, mediante discusión de las preferencias de los expertos, durante una o varias rondas [131]. Se trata de un proceso dinámico e iterativo, que suele estar coordinado o dirigido por una figura humana: el *moderador*. El moderador es una figura clave en procesos de consenso, y sus funciones fundamentales son:

- Evaluar el consenso alcanzado en cada ronda de consenso.
- Identificar las alternativas que impiden alcanzar el consenso deseado.
- Informar a los expertos sobre los cambios que éstos deben considerar sobre las preferencias en dichas alternativas.

En el siguiente apartado describimos el esquema general de actuación para procesos de consenso, en el que se basa el modelo seguido en este proyecto.

### A.3.2. Esquema General de los Procesos de Consenso

En esta subsección presentamos un esquema general del proceso de consenso, el cual se ajusta a enfoques suavizados del consenso, tales como el de "soft consensus" basado en mayoría difusa. Las principales fases llevadas a cabo en este proceso son las siguientes [74, 76, 78, 145]:

1. Descripción del problema de toma de decisión.

2. Identificación del formato empleado para representar las preferencias, junto con las medidas para calcular el nivel de consenso alcanzado a partir de dichas preferencias [37, 45, 48, 50, 61, 63, 121].

3. Cada experto proporciona sus preferencias individuales al moderador.

4. Comprobación del nivel de consenso actual, de acuerdo a las medidas de consenso elegidas. Diferentes modelos se han propuesto para obtener el grado de consenso empleando diversas medidas y tipos de información [85, 33]. Si el nivel de consenso es suficiente, el proceso finaliza y el grupo pasa a la fase de selección de alternativas; de lo contrario, continuar con el paso 5.

5. Realimentación por parte del moderador a los expertos. Cuando existe una cierta discrepancia entre las preferencias de los expertos, el moderador identifica a los expertos y/o preferencias que impiden alcanzar el nivel de consenso deseado. A continuación, el moderador aconsejará a los expertos que modifiquen dichas preferencias, con el objeto de acercar sus opiniones y mejorar el grado de consenso en la siguiente ronda. Como puede verse, el moderador juega un papel clave en esta etapa, siendo responsable de guiar, controlar y finalizar el proceso de consenso.

6. Volver al paso 3. Dado que el número de rondas de consenso debe estar limitado, en los casos en que este número se haya sobrepasado sin llegar a un acuerdo, deben buscarse estrategias alternativas o finalizar el proceso sin éxito [131].

En la figura A.2 se muestra un esquema de este proceso.

## A.4. Modelo de Consenso Propuesto

A lo largo de la literatura podemos encontrar un gran número de modelos de Consenso empleados en Problemas de TDG [62, 81, 101, 131, 145]. De entre todos ellos podemos

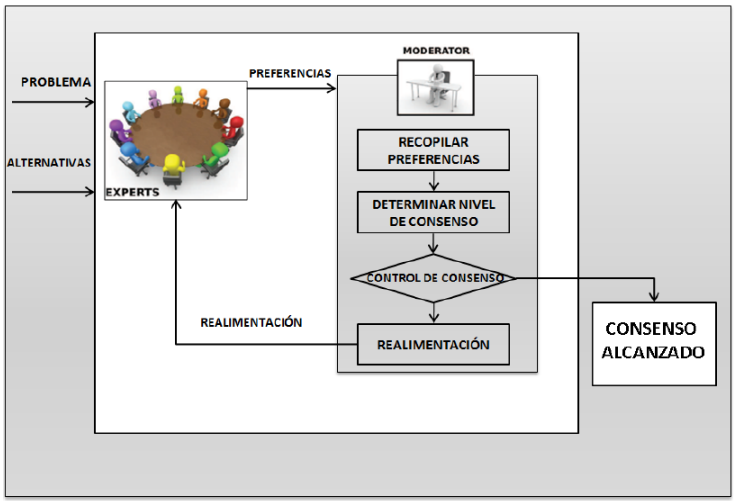


Figura A.2: Esquema General de "Soft Consensus" para TDG

destacar:

- Modelo teórico del proceso de consenso, propuesto por Saint y Lawson [131].
- Modelo propuesto por Zadrozny con Información Difusa [145].
- Modelo de Bryson con vectores de preferencia [101].
- Modelo de Kim para problemas de TDG multicriterio [81].
- Modelo de Herrera con valoraciones ligüísticas [62].

Una vez citados algunos de los modelos de consenso más relevantes, vamos a presentar en mayor detalle el modelo de sistema de apoyo al consenso para problemas de TDG bajo incertidumbre que se utilizará en este proyecto fin de carrera.

Para estudiar las fases de las que se compone nuestro modelo de consenso, es necesario conocer en primer lugar la definición y contexto en el que se situarán los problemas de TDG a abordar. Formalmente, el problema de TDG estará definido de esta forma: dado un conjunto de expertos $E = \{e_1, ..., e_m\}$ , cada $e_i$ debe expresar sus preferencias sobre un conjunto de alternativas $X = \{x_1, ..., x_n\}$ mediante una relación de preferencia $P_i$, donde $p_i^{lk} \in [0, 1]$.

Una vez formalizada la notación básica del problema, procedemos a describir las etapas seguidas en nuestro modelo, representado en la figura A.3:

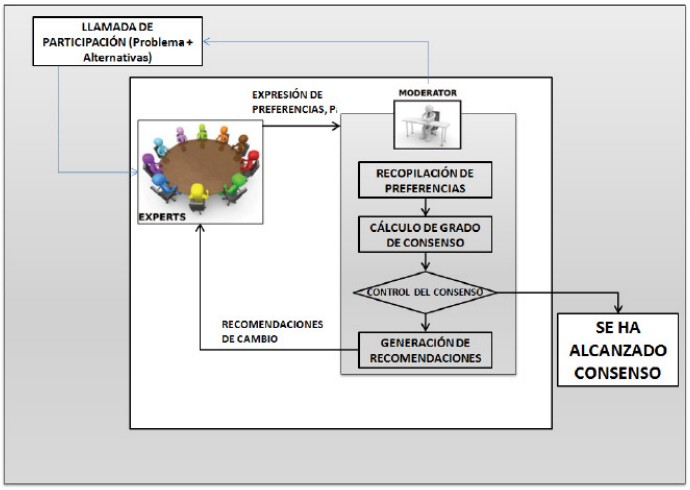


Figura A.3: Modelo de Consenso

### A.4.1. Llamada de participación en el Proceso de Consenso

En esta primera etapa, el moderador informa al conjunto $E$ de expertos sobre el problema, proporcionándoles el conjunto $X$ de alternativas que componen el problema de TDG e invitándoles a participar en el mismo. Cada experto decide si participa o no en el problema. Una vez transcurrido un período de tiempo previamente fijado por el moderador, dará comienzo el problema, (siempre que al menos hayan aceptado participar 2 expertos en él). Antes de comenzar el proceso de consenso, el moderador fija los parámetros del problema, tales como el umbral de consenso $\mu \in [0, 1]$ (mínimo nivel de acuerdo deseado) y el número máximo de rondas permitido, $Maxrounds \in \mathbb{N}$.

### A.4.2. Expresión de Preferencias de los Expertos

Cada experto debe utilizar una estructura de preferencia para representar su opinión sobre un conjunto de alternativas, de modo que éstas se las pueda proporcionar al moderador. Una de las estructuras más habituales en problemas de TDG es la *relación de preferencia difusa*, que será la que utilizaremos en nuestro modelo de consenso. Dado un conjunto finito de alternativas $X$, una relación de preferencia difusa $P_i$ asociada al experto $e_i$ es una matriz cuadrada de dimensión $n$:

$$P_i = \begin{pmatrix} p_i^{11} & \cdots & p_i^{1n} \\ \vdots & \ddots & \vdots \\ p_i^{n1} & \cdots & p_i^{nn} \end{pmatrix}$$

donde cada elemento $p_i^{lk} = \mu_{P_i}(x_l, x_k) \in [0, 1]$ representa la preferencia de la alternativa $x_l$ sobre la alternativa $x_k$ del experto $e_i$, cumpliéndose:

- $p_i^{lk} > 0.5$ indica que el experto $e_i$ prefiere la alternativa $x_l$ respecto de $x_k$.
- $p_i^{lk} < 0.5$ indica que el experto $e_i$ prefiere la alternativa $x_k$ respecto de $x_l$.
- $p_i^{lk} = 0.5$ indica indiferencia del experto $e_i$ entre las alternativas $x_l$ y $x_l$.

Los elementos $p_i^{ll}$ situados en la diagonal de esta matriz carecen de relevancia durante la resolución de un problema de TDG, ya que no es adecuado hablar de la preferencia de una alternativa sobre sí misma.

Es deseable que las valoraciones cumplan la propiedad de reciprocidad, de manera que si $p_i^{lk} = x(x \in [0, 1])$, entonces $p_i^{kl} = 1 - x$. De esta forma se mantendrá la consistencia entre valoraciones sobre pares de alternativas recíprocos.

### A.4.3. Cálculo del Grado de Consenso

Una vez recibidas las preferencias de los expertos, el moderador procede al cálculo del nivel de acuerdo entre estos. Para ello, se realizarán los siguientes pasos:

- En primer lugar, para cada par de expertos $e_i, e_j (i < j)$, se calcula una matriz de similitud $SM_{ij}^{lk} = (sm_{ij}^{lk})$ utilizando la medida de similitud propuesta en [23], donde se emplea la *distancia Euclídea*,

$$sm_{ij}^{lk} = 1 - |p_i^{lk} - p_j^{lk}|$$

  donde $sm_{ij}^{lk} \in [0, 1]$ es la similitud entre los expertos $e_i$ y $e_j$ sobre el par de alternativas $(x_l, x_k)$.

- Se calcula una matriz de consenso, $CM = (cm^{lk})$, agregando a nivel de pares todas las matrices de similitud previamente calculadas. Cada elemento de esta matriz $cm \in [0, 1]$ se calcula como,

$$cm^{lk} = \phi(sm_{12}^{lk}, sm_{13}^{lk}, \cdots, sm_{1m}^{lk}, sm_{23}^{lk}, \cdots, sm_{2m}^{lk}, \cdots, sm_{(m-1)m}^{lk})$$

  donde $\phi$ es el operador de agregación empleado, y $l, k \in \{1, \cdots, n\}$.

- Por último, se procede a calcular el nivel de acuerdo. Esto se hace a tres niveles:

  1. Consenso a nivel de pares de alternativas. Se obtiene como,

$$cp^{lk} = cm^{lk}, \forall l, k = 1, \cdots, n \wedge l \neq k,$$

     donde $cp^{lk}$ representa el acuerdo alcanzado sobre el par de alternativas $(x_l, x_k)$, obtenido directamente a partir de la matriz de consenso $CM$.

  2. Consenso a nivel de alternativas,

$$ca^l = \frac{\sum_{k=1, l \neq k}^{n} cp^{lk}}{(n-1)},$$

donde $cp^{lk}$ representa el acuerdo sobre la alternativa $x_l$.

3. Consenso a nivel de relación de preferencia,

$$cr = \frac{\sum_{l=1}^{n} ca^l}{n},$$

donde $cr$ representa el grado de consenso global alcanzado entre los expertos en la ronda actual.

### A.4.4. Control de Consenso

Se comprueba el nivel de consenso $cr$ obtenido en la anterior fase. Si $cr$ es mayor o igual que el umbral de consenso inicialmente fijado, $\mu$, se entiende que se ha alcanzado el nivel de acuerdo deseado y el proceso finaliza; de lo contrario, el proceso requiere mayor discusión y debe continuar , a menos que se haya alcanzado el número máximo de rondas permitido (*Maxrounds*), en cuyo caso el proceso finaliza sin haber llegado a un acuerdo y se deberán emplear estrategias alternativas para tomar la decisión.

### A.4.5. Generación de Recomendaciones

Esta fase se da si en la etapa de control de consenso el grado obtenido era menor que el umbral $\mu$, y en ella se sugiere a los expertos la manera en que deberían modificar sus preferencias de cara a incrementar el nivel de acuerdo en rondas siguientes. Para obtener estas *recomendaciones de cambio*, se llevan a cabo tres tareas:

- *Calcular la preferencia colectiva y las matrices de proximidad de los expertos.* La preferencia colectiva $P_c = (p_c^{lk})$ se calcula agregando todas las preferencias de los expertos $\{P_1, \cdots, P_m\}$ a nivel de pares de alternativas:

$$p_c^{lk} = \phi(p_1^{lk}, \ldots, p_m^{lk})$$

A continuación, se determinan las matrices de proximidad, que indican cuán próximas están las preferencias de cada experto a la preferencia colectiva. Cada experto $e_i$ tiene una matriz de proximidad, y cada elemento de dicha matriz se calcula midiendo la distancia entre la preferencia de cada experto y la preferencia colectiva,

$$pp_i^{lk} = 1 - |p_i^{lk} - p_c^{lk}|$$

Con estos valores de proximidad, es posible identificar las preferencias más alejadas de la preferencia colectiva, y por consiguiente, los expertos que deberían cambiar sus opiniones.

- *Identificación de las preferencias que deben ser cambiadas.* Existen diferentes criterios para elegir las preferencias que deben modificarse, considerando tanto el grado de consenso como los valores de proximidad [43, 99]. Nuestro modelo propone hacer los siguientes cambios en aquellos pares de alternativas cuyo grado de consenso a nivel de alternativas, $ca^l$ no sea suficiente, obteniendo el conjunto de cambios, $CC$,

$$CC = \{(x_l, x_k) | ca^l < cr \wedge cp^{lk} < cr\} \forall l, k \in \{1, \cdots, n\}$$

Una vez identificados los pares a cambiar, nuestro modelo debe identificar a los expertos que deberían hacer cambios en cada uno de estos pares. De este modo, los expertos cuya preferencia esté más alejada de la preferencia colectiva para el par $(x_l, x_k) \in CC$ deberán de modificar su valoración sobre dicho par. Para poder identificar a estos expertos, definimos un umbral de proximidad o *proximidad media* $\overline{pp^{lk}}$ para cada par, que se calcula agregando todos los valores o matrices de proximidad de los expertos a nivel de pares:

$$\overline{pp^{lk}} = \phi(pp_1^{lk}, \ldots, pp_m^{lk})$$

Nuestro modelo aconsejará cambiar el par $(x_l, x_k) \in CC$ a todo experto $e_i$ cuyo $pp_i^{lk} < pp^{lk}$.

- *Establecer las direcciones de cambio.* El modelo utiliza una serie de reglas de dirección para sugerir la dirección correcta en las recomendaciones de cambio, y mejorar el acuerdo en siguientes rondas. Para cada recomendación de cambio $((x_l, x_k), e_i)$ asociada a un experto y par de alternativas determinados, se sugerirá incrementar o decrementar la valoración del experto sobre dicho par. Para ello se tendrá en cuenta además, la preferencia colectiva $P_c$.

  - DIR.1: Si $((p_i^{lk} - p_c^{lk}) < 0)$, entonces se recomienda al experto $e_i$ incrementar la valoración asociada al par $(x_l, x_k)$.
  - DIR.2: Si $((p_i^{lk} - p_c^{lk}) > 0)$, entonces se recomienda al experto $e_i$ decrementar la valoración asociada al par $(x_l, x_k)$.
  - DIR.3: Si $((p_i^{lk} - p_c^{lk}) = 0)$,entonces el experto $e_i$ no tiene que modificar la valoración asociada al par $(x_l, x_k)$.

Una vez obtenidas todas las recomendaciones de cambio, el moderador se las proporciona a los expertos. La siguiente ronda comienza con una nueva expresión de preferencias por parte de los expertos, basándose en las recomendaciones de cambio recibidas.

# Apéndice B

# Manual de Instalación

Este anexo está dedicado a realizar las configuraciones necesarias para poner en marcha nuestra aplicación.

Hemos de decir que nuestro sistema trabajará con un servidor basado en plataforma Microsoft Windows, por lo que todas estas instalaciones y configuraciones hay que realizarlas teniendo en cuenta dicho Sistema Operativo.

## B.1. Instalación del Servidor

Todo el material necesario para instalar y dejar operativo el servidor se encuentra disponible en el DVD que acompaña a esta memoria. Inserte el DVD en la unidad D, sitúese en D:\MInstalacion y compruebe que se encuentran los siguientes archivos:

- JADE-all-4.3.0.zip
- xampp-win32-1.8.0-VC9-installer.exe
- ServletCOMAS.war
- Fichero BDComas.*sql*

Si es así podemos proceder a la instalación de nuestro servidor inmediatamente. Si falta algún archivo o alguno de ellos se encuentra dañado, póngase en contacto con el responsable de la aplicación para subsanar el problema.

### B.1.1. Instalación de XAMPP

En primer lugar, localizamos en el disco de instalación el archivo xampp-win32-1.8.0-VC9-installer.exe, el cual lo podremos descargar de la página oficial de XAMPP[1], y hacemos doble clic en él. Tras hacer clic, nos aparecerá una ventana indicando si permitimos que éste programa realice cambios en el equipo, pulsamos la opción "Sí".

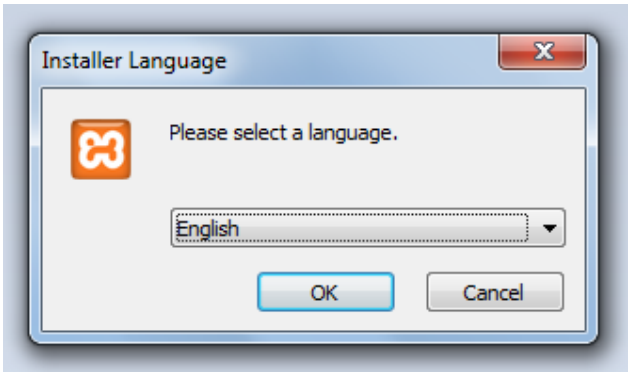


Figura B.1: Ventana de elección de idioma de instalación

Después de permitir que el programa pueda hacer cambios, aparecerá una ventana similar a la figura B.1 para elegir el idioma de instalación de XAMPP. En nuestro caso, dado que sólo están disponibles los idiomas inglés y alemán, elegimos "inglés" y pulsamos el botón "OK" tal y como se muestra en la figura B.2.

Acto seguido comenzará la instalación, con la aparición de la siguiente pantalla (figuraB.3). En nuestro caso, pulsaremos el botón "Next".

Una vez pulsado el botón "Next" no aparecerá una ventana en la que podremos seleccionar

[1] www.apachefriends.org

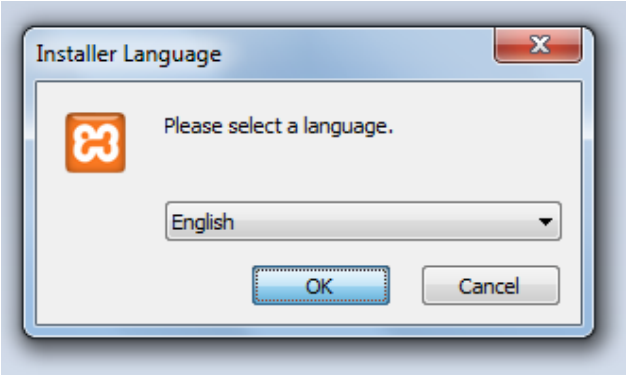


Figura B.2: Elección inglés como idioma de instalación

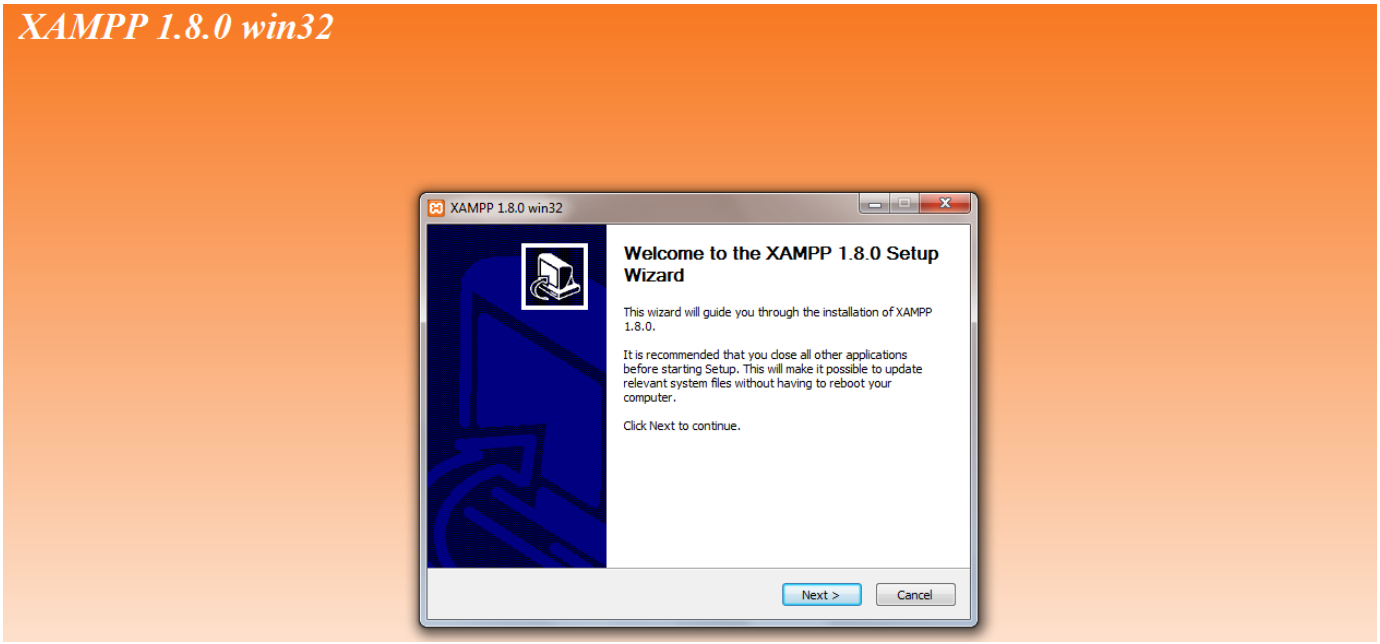


Figura B.3: Pantalla de inicio de instalación

el lugar de instalación de XAMPP. En nuestro caso, optamos por la instalación por defecto en la unidad C: y pulsamos nuevamente el botón “Next” (figura B.4)

Después de hacer clic en el botón “Next” nos aparecerá una pantalla para elegir las opciones de instalación. En este caso, seleccionaremos todas las opciones que nos aparecen, marcando las tres opciones pertenecientes a SERVICE SECTION, tal y como se muestra en

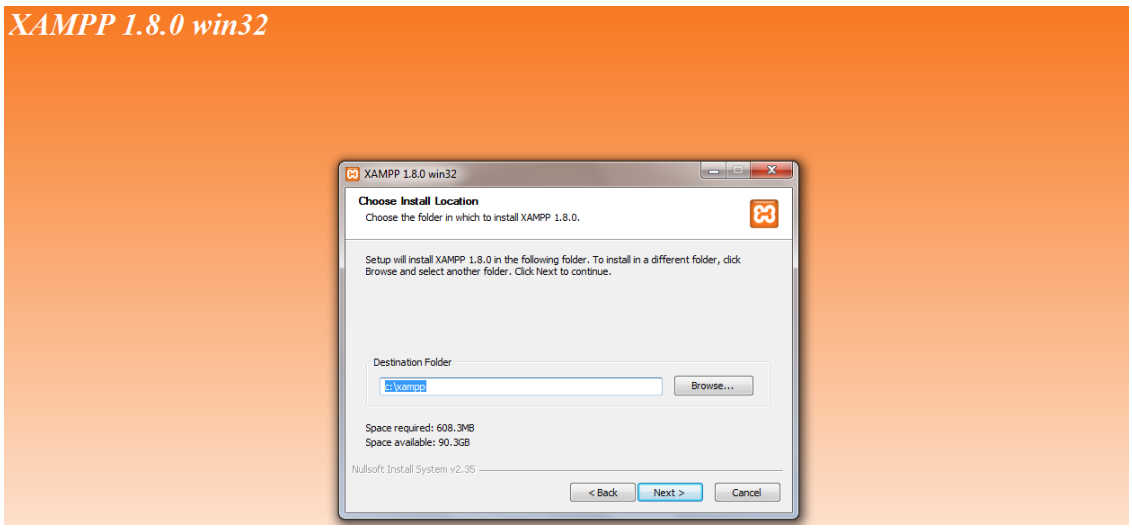


Figura B.4: Pantalla de elección del lugar de instalación

la figura B.5.

- Install Apache as service.
- Install MySQL as service.
- Install Filezilla as service.

Finalmente pulsamos el botón “Install”.

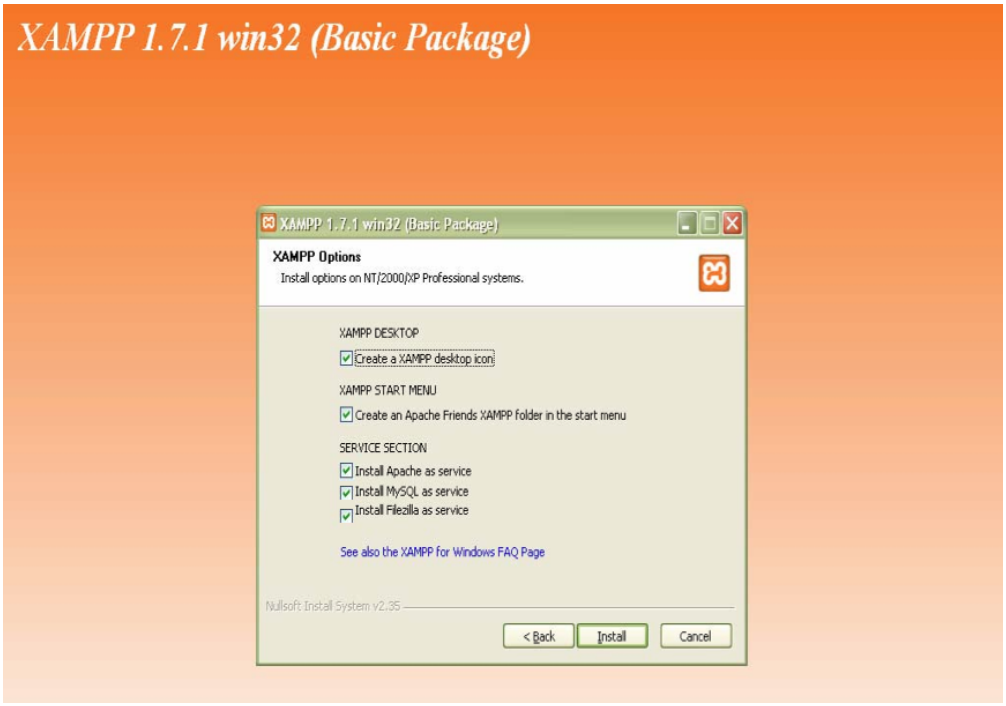


Figura B.5: Pantalla de elección de opciones de instalación

Por último aparecerá la pantalla que indica que la instalación se ha completado (ver figura B.6). Para salir pulsaremos el botón “Finish”.

Figura B.6: Pantalla de instalación completada

Tras la instalación, XAMPP comprueba los puertos necesarios por las distintas herramientas que aglutina. En primer lugar tendremos que permitir el acceso de apache (ver figuraB.7).

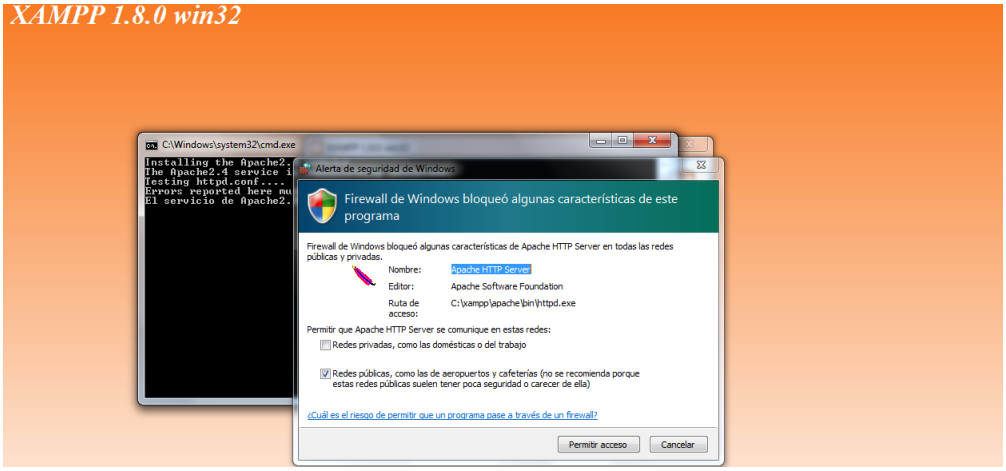


Figura B.7: Permitir acceso de Apache HTTP Server

Después de haber habilitado el acceso de apache, aparecerá una ventana indicando la finalización del servicio de instalación (ver figura B.8). Finalmente pulsamos el botón “Aceptar” para abandonar dicha ventana.

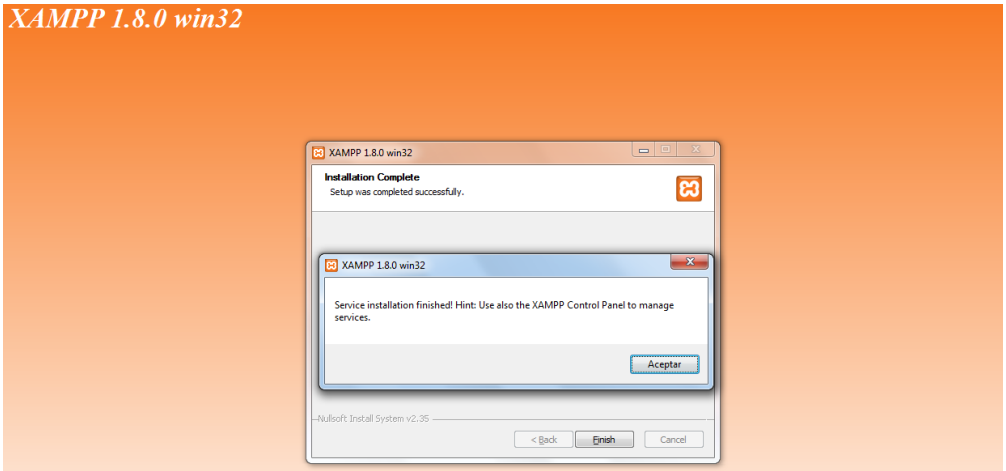


Figura B.8: Ventana de finalización del servicio de instalación

### B.1.2. Configuración Base de datos

Para configurar la Base de datos, utilizaremos la herramienta phpMyAdmin, la cual se encuentra dentro de la herramienta XAMPP.

Una vez que hemos arrancado apache en el panel de control de XAMPP, abriremos el navegador e introduciremos en la barra de direcciones la siguiente dirección:

http:\\localhost

Después de realizar lo descrito anteriormente nos aparecerá en el navegador la pantalla de bienvenida de XAMPP, tal y como se muestra en la figura B.9

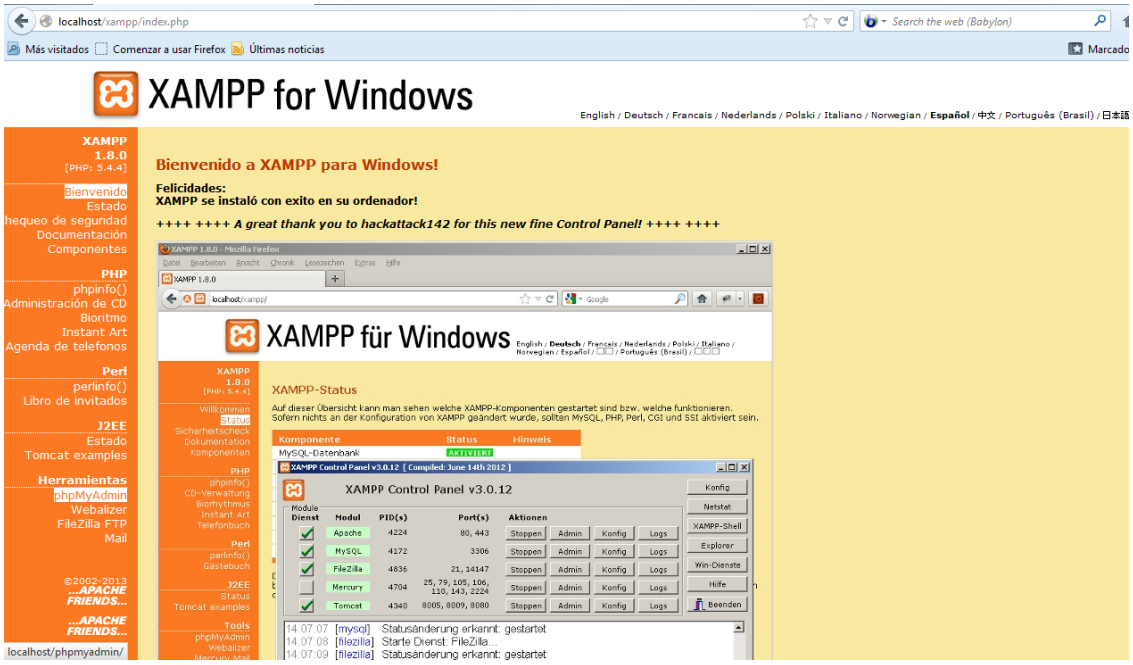


Figura B.9: Pantalla de Bienvenida de XAMPP

Tras haber accedido a la pantalla inicial, seleccionamos haciendo clic en el apartado denominado “phpMyAdmin”, situado en el menú de la izquierda de la pantalla. A continuación se mostrará la pantalla inicial de “phpMyAdmin” (ver figura B.10).

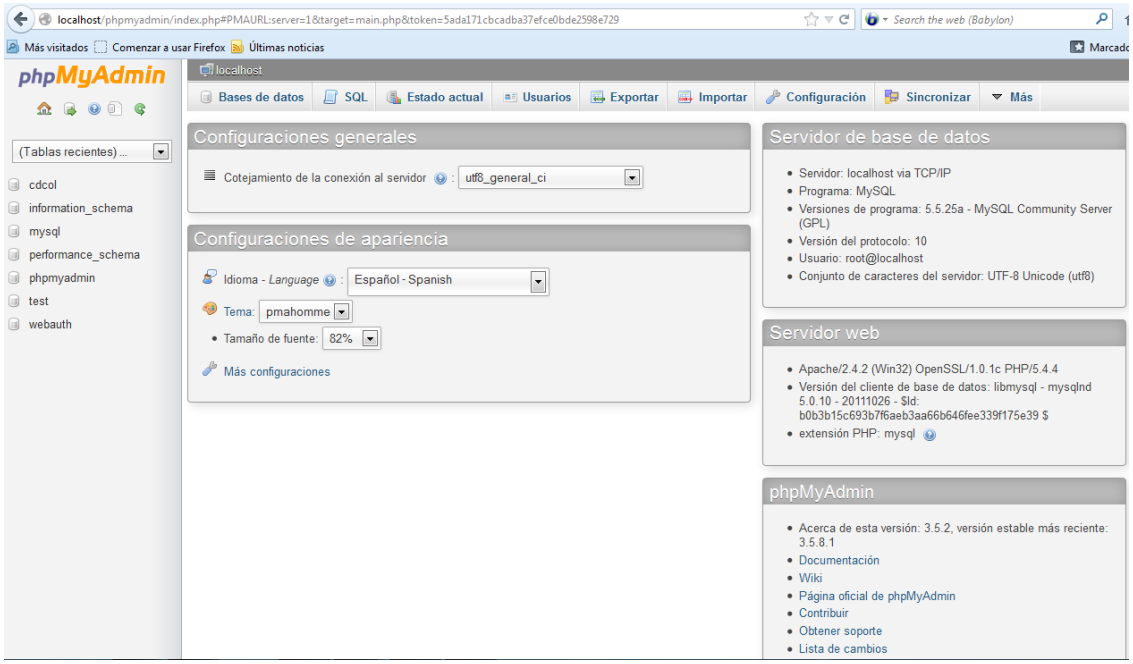


Figura B.10: Pantalla de inicial de phpMyAdmin

Ya, en la pantalla inicial de “phpMyAdmin” seleccionamos en el menú superior el apartado denominado “Importar”. A continuación se abrirá la pantalla para realizar la importación de la base de datos.

Dentro de la sección “Archivo a importar”, haremos clic en el botón “Examinar” y seleccionaremos el archivo BDComas.sql que hay en el disco de instalación (ver figura B.11).

Finalmente vamos a la parte inferior de la página y pulsamos el botón “Continuar”, como podemos observar en la imagen B.12.

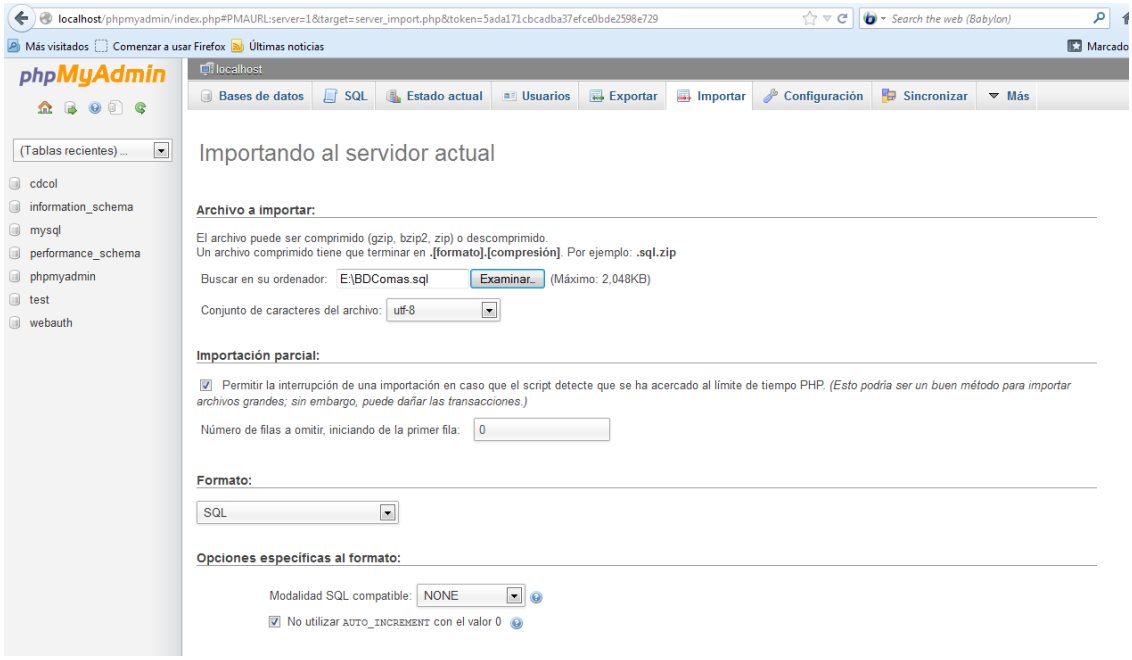


Figura B.11: Seleccionar archivo a importar

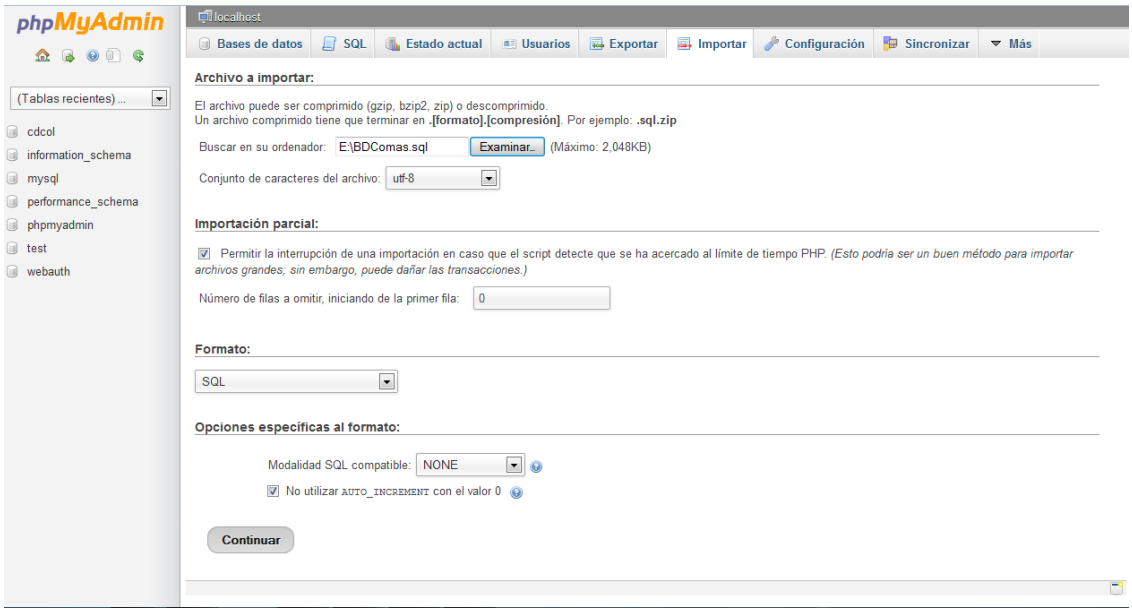


Figura B.12: Finalizar importación

Si todo ha ido bien, se habrá creado una base de datos llamada “BDComas” con sus

respectivas tablas y datos (ver figura B.13).

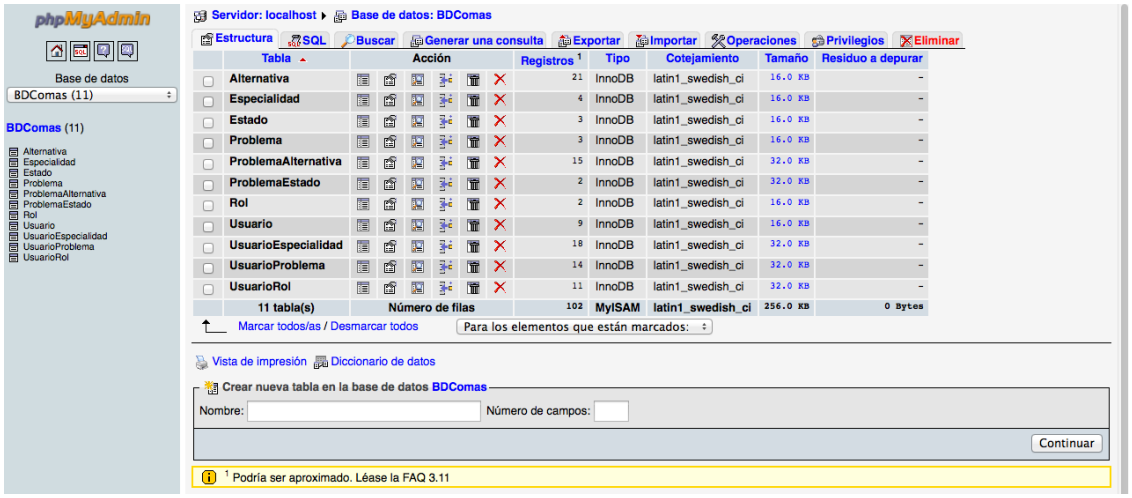


Figura B.13: Base de Datos que se acaba de importar

### B.1.3. Configuración de Tomcat

En primer lugar tendremos que acceder al panel de control de XAMPP, haciendo doble clic en el icono de XAMPP que hay en el escritorio.

Una vez que estamos dentro, hacemos clic en el icono que hay a la izquierda del módulo de Tomcat, el cual tiene una cruz, tal y como podemos ver en la figura B.14.

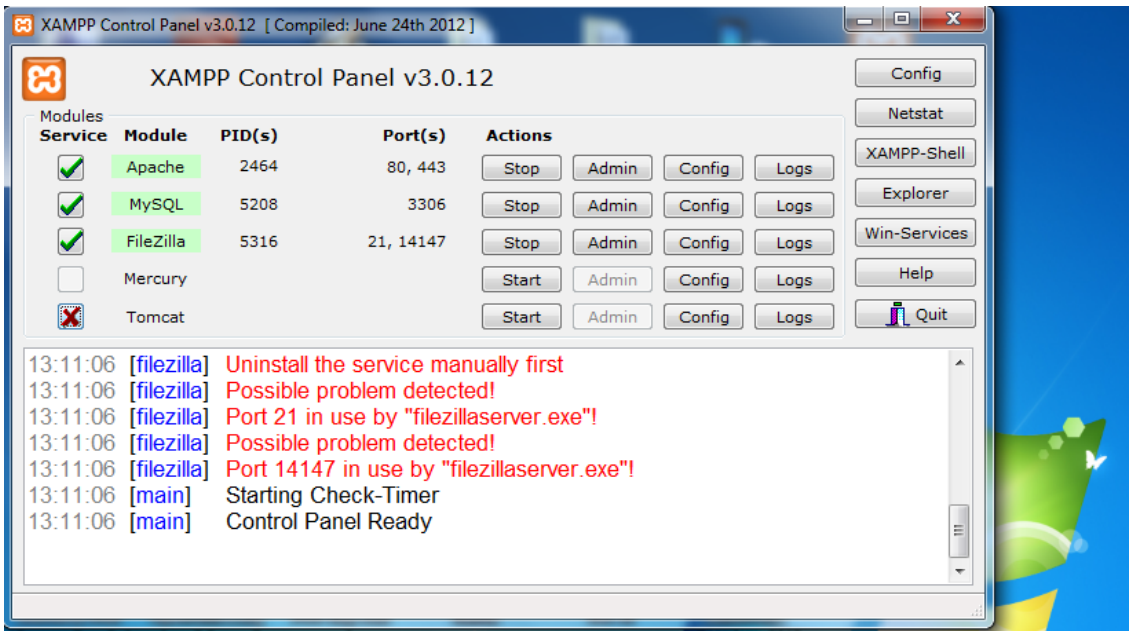


Figura B.14: Panel de control de XAMPP

Tras hacer clic, nos aparecerá una ventana que nos indicará si deseamos instalar el servicio de Tomcat. En este caso, pulsaremos sobre el botón “Yes” (ver figura B.15).

Después de haber instalado el servicio, pulsamos el botón “Start” correspondiente a las acciones de Tomcat (ver figura B.16).

A continuación nos aparecerá una ventana para que permitamos el acceso de java al firewall. Esta vez pulsaremos el botón “Permitir acceso”, tal y como se puede apreciar en la figura B.17.

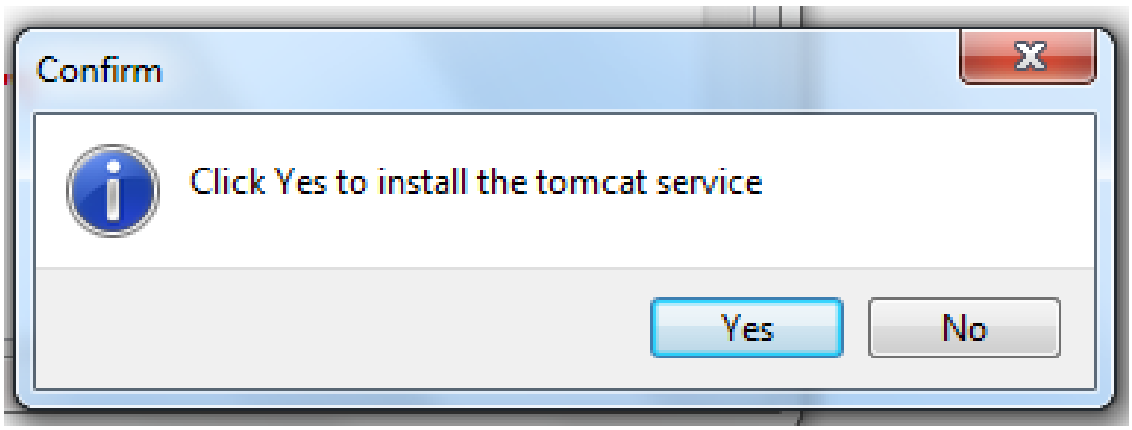


Figura B.15: Ventana para instalar servicio tomcat

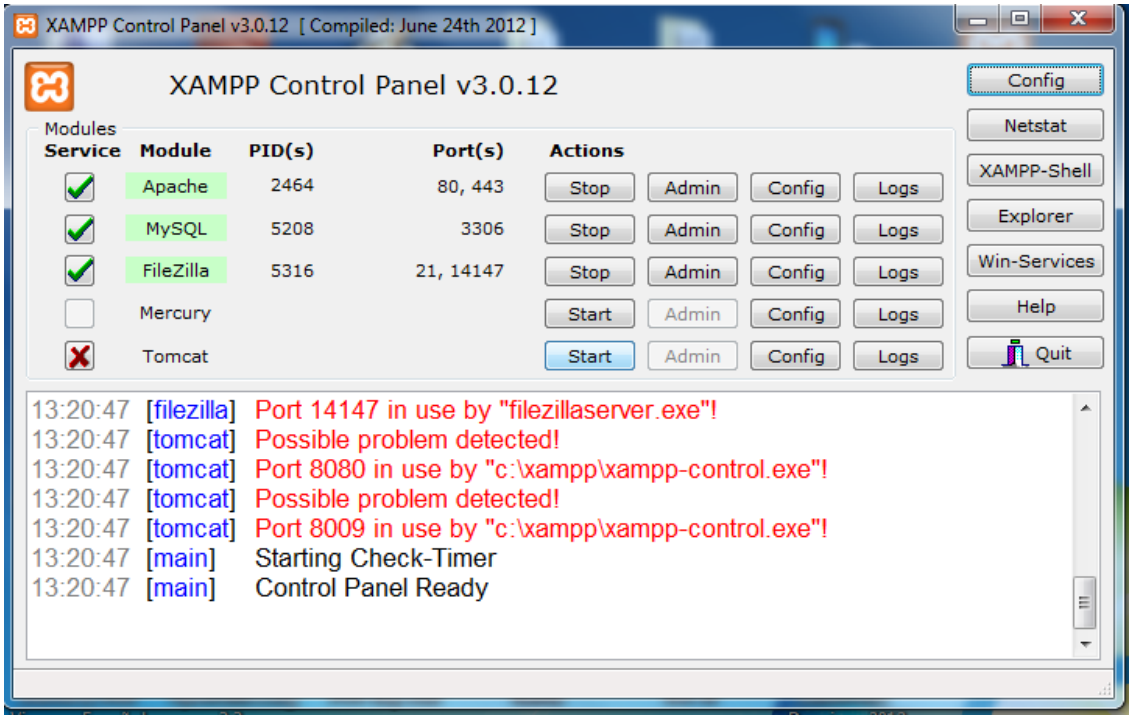


Figura B.16: Iniciar tomcat

Finalmente, Tomcat aparecerá como que está ejecutándose (ver figura B.18).

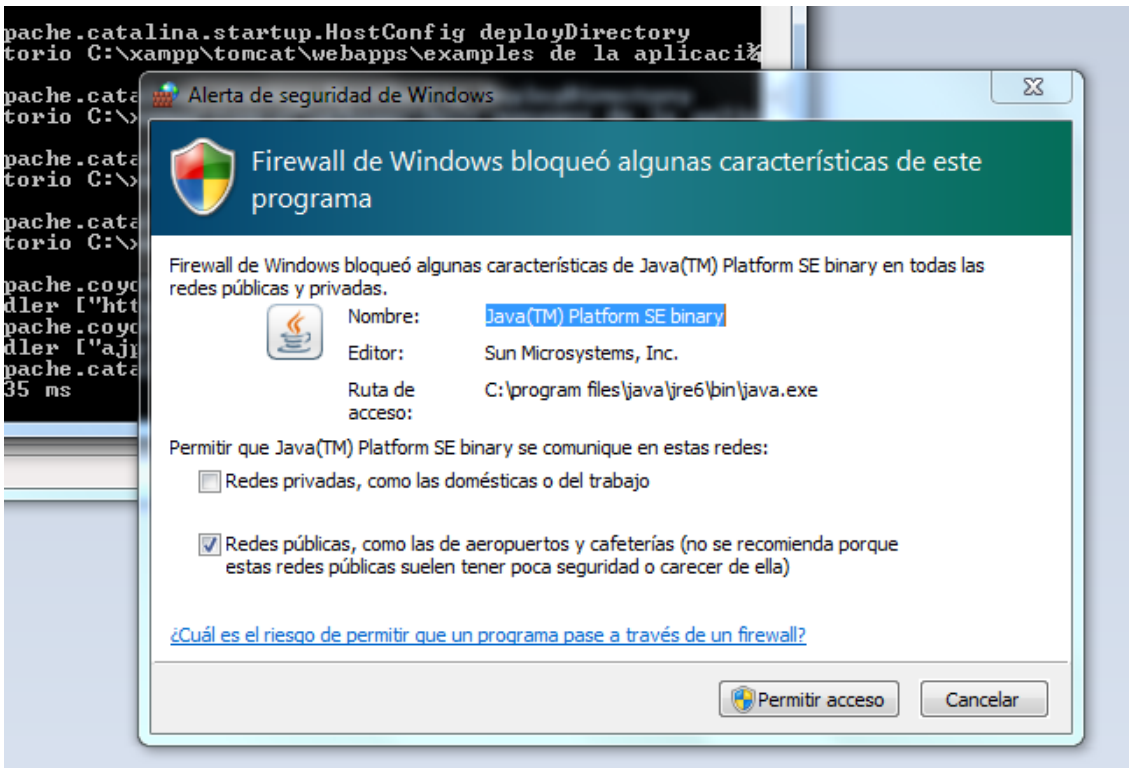


Figura B.17: Permitir acceso al firewall

### B.1.4. Instalación JADE

En primer lugar localizamos en el disco de instalación el fichero JADE-all-4.3.0.zip y lo descomprimimos. Posteriormente descomprimiremos el fichero JADE-bin-4.3.0.zip

El contenido de la carpeta JADE que se encuentra en al descomprimir los archivos lo podemos copiar en el directorio raíz C: . Quedando de la siguiente forma (ver figura B.20)

Una vez hecho esto, accedemos a la capeta "lib", dentro del directorio "JADE" y encontraremos las librerías que permiten interacturar y utilizar la plataforma (ver figura B.21).

Para poder ejecutar la plataforma JADE, se debe crear un archivo, al que llamaremos *runjade*, con extensión .bat (Runjade.bat). Este archivo lo crearemos en el directorio JADE.

Para crear el archivo .bat haremos lo siguiente:

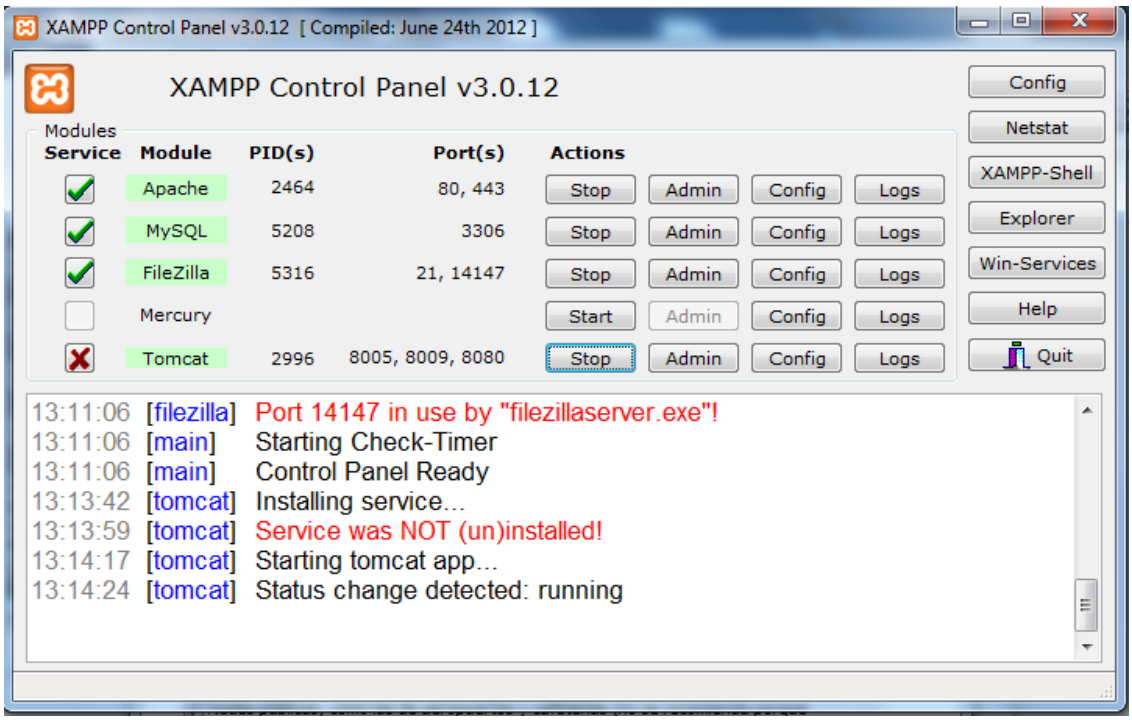


Figura B.18: Ejecución de tomcat

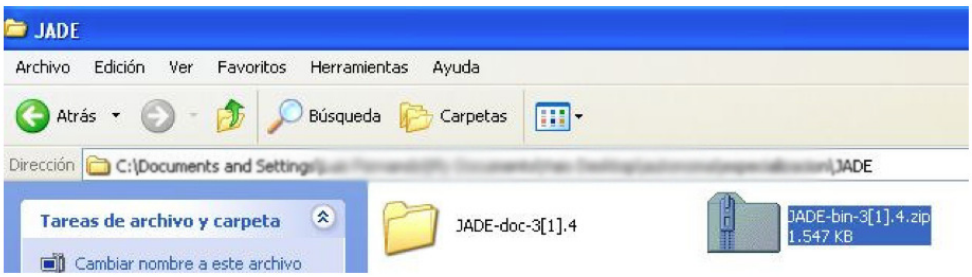


Figura B.19: Archivos incluidos en JADE-all-4.3.0.zip

1. Nos situamos dentro de la carpeta de JADE.

2. Hacemos clic con el botón derecho en el fondo de la ventana y nos aparecerá un menú.

3. En el menú seleccionamos la opción “Nuevo” y posteriormente se desplegará un submenú.

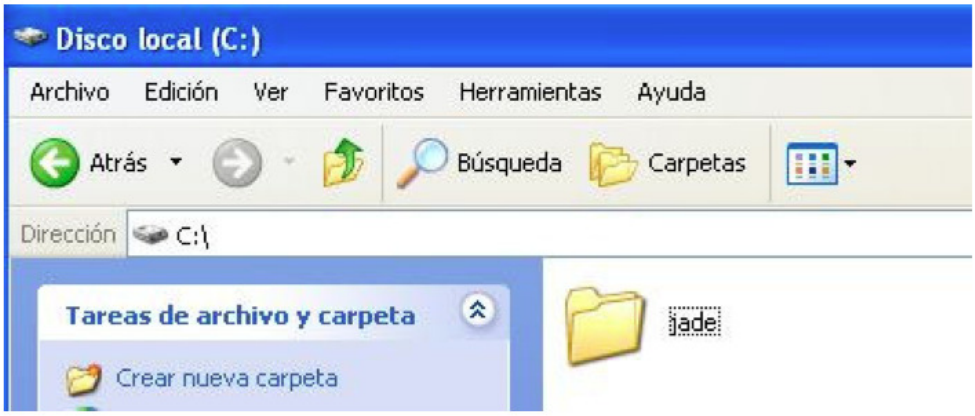


Figura B.20: Copiar la carpeta JADE en C:

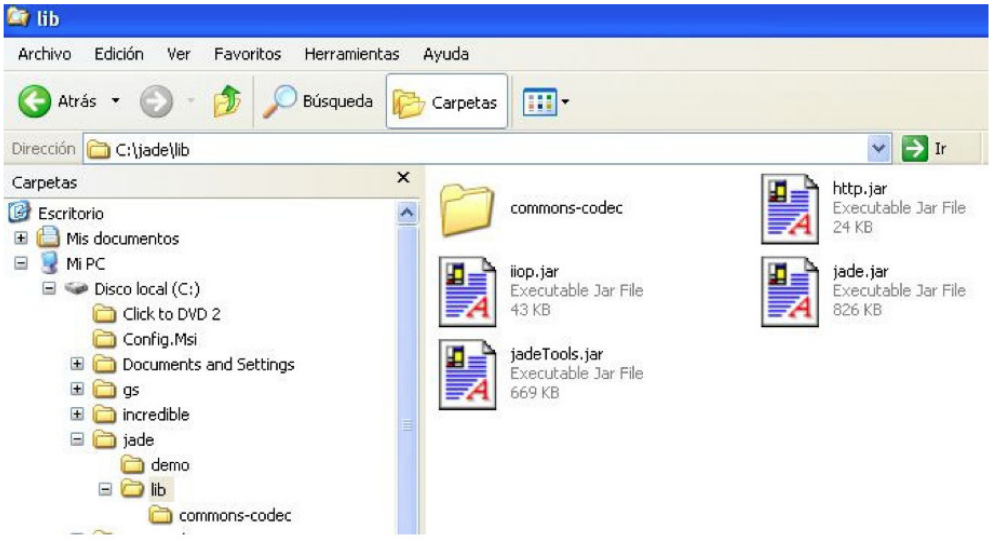


Figura B.21: Librerías de JADE

4. Dentro del submenú escogeremos la opción “Documento de texto”

Estos pasos se pueden ver en la figura B.22

Después de seleccionar el documento de texto se debe cambiar el nombre y la extensión del documento, tal y como se indica en la figura B.23

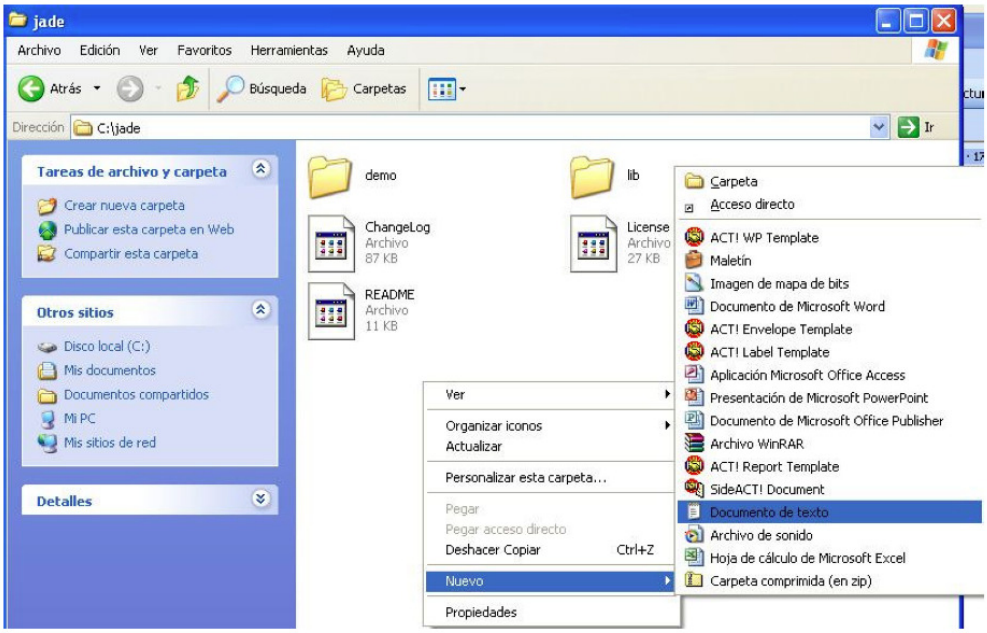


Figura B.22: Creación de un Documento de texto en el directo JADE

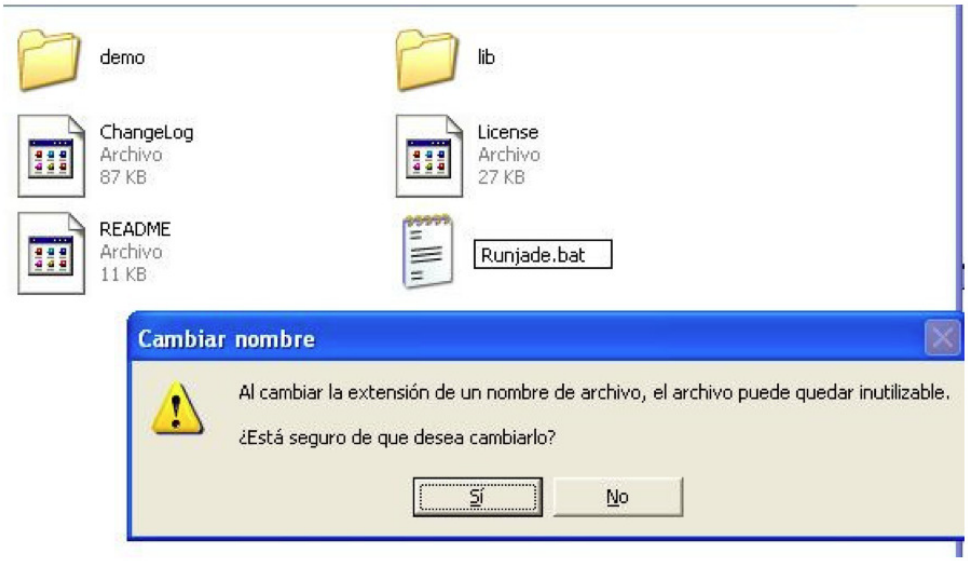


Figura B.23: Cambiar extensión al Documento de texto

Una vez hecho esto, la imagen del icono cambiará.

A continuación tendremos que editar el archivo Runjade.bat y añadir las líneas que se muestran en la figura B.24

```
java -classpath .;.\lib\jade.jar;.\lib\jadeTools.jar;
.\lib\iiop.jar;.\lib\http.jar; jade.Boot %1 %2 %3 %4
%5 %6 %7 %8 %9
```

Figura B.24: Dato a añadir en el archivo Runjade.bat

Por último, para ejecutar la plataforma JADE podría hacerlo desde la línea de comandos DOS poniendo lo que muestra la figura B.25

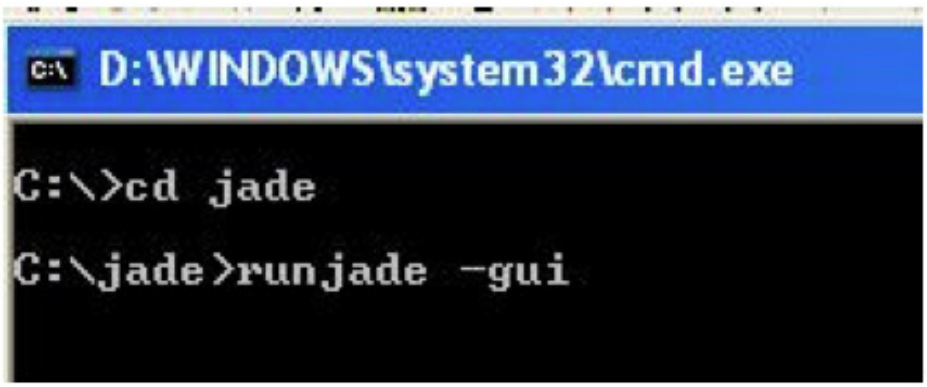


Figura B.25: Sentencias de ejecución de la plataforma JADE

Una vez introducidas estas sentencias, la plataforma se ejecutará tal y como se puede visualizar en la figura B.26

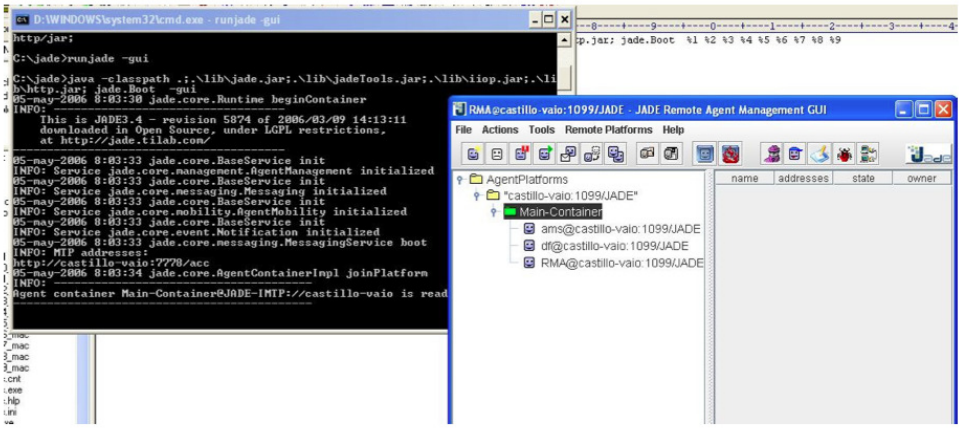


Figura B.26: Interfaz de la plataforma JADE

# Apéndice C

# Manual del Usuario "Administrador"

Este manual está desarrollado para que sirva como guía visual, al usuario administrador. De manera que se recogerán las pantallas y los pasos a seguir a la hora de realizar las distintas acciones propias del usuario Administrador.

## C.1. Acceso

En primer lugar, lo que deberemos hacer es acceder al sistema como usuario administrador. Para ello, abriremos un navegador e introduciremos la siguiente dirección:

http://localhost:8080/ServletCOMAS.

Un vez que hemos realizado el paso anterior, nos aparecerá la página de la figura C.1

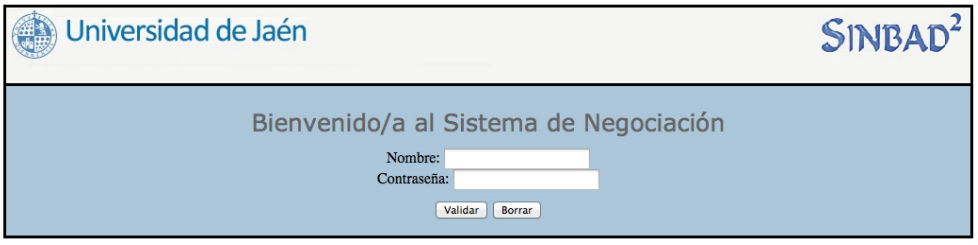


Figura C.1: Página Principal

Seguidamente, rellenamos los campos *Nombre* y *Contraseña* con las credenciales del usuario Administrador. En este caso, las credenciales son: (*admin, admin*) y pulsamos el botón *Validar*.

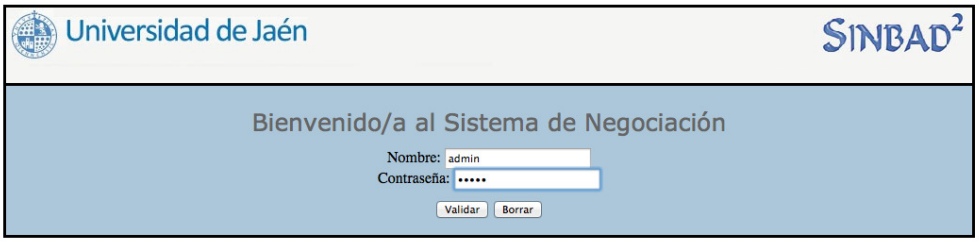


Figura C.2: Acceso como Usuario Administrador

Tras haber pulsado el botón validar y si hemos introducidos correctamente las credenciales, accederemos al menú principal de usuario Administrador.

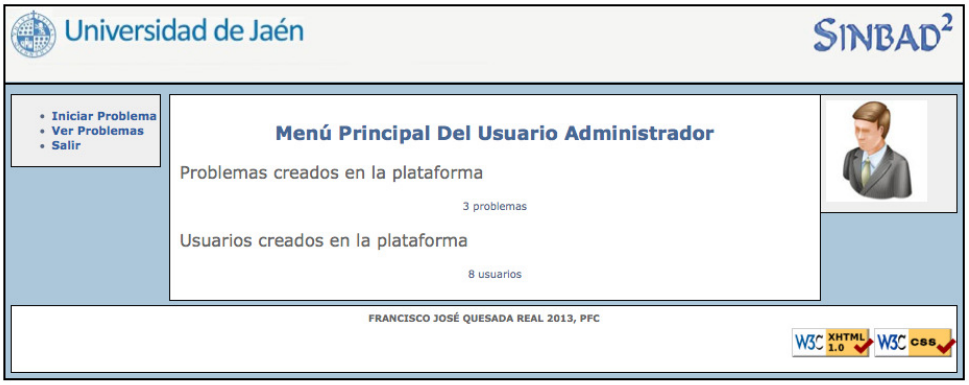


Figura C.3: Menú Principal Usuario Administrador

## C.2. Iniciar Problema

Una vez que ya hemos accedido al sistema como usuario de tipo Administrador (ver apartado C.1), para iniciar un problema, hacemos clic al vínculo que hay en el menú de la

izquierda denominado *"Iniciar Problema"*, tal y como se muestra en la figura C.4

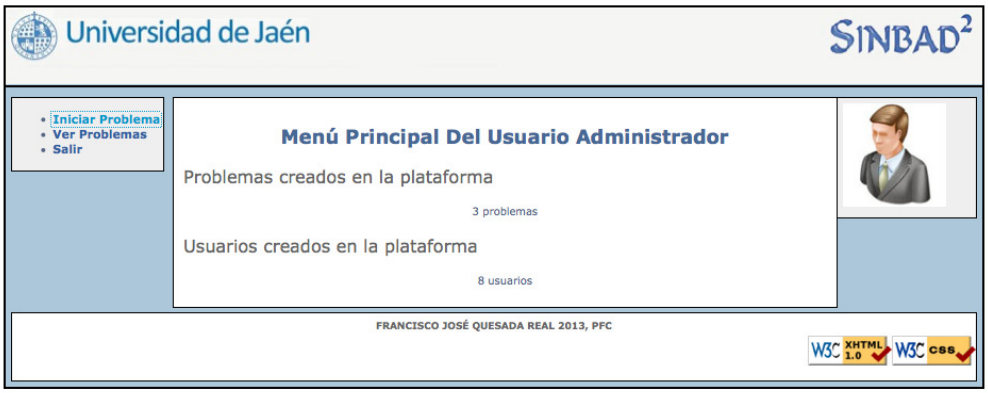


Figura C.4: Acceder a Iniciar Problema

Tras hacer clic en el vínculo indicado, nos aparecerá una pantalla como la que se muestra en la figura C.5 en la que seleccionaremos un problema de la lista desplegable y finalmente pulsaremos el botón "*Iniciar*".

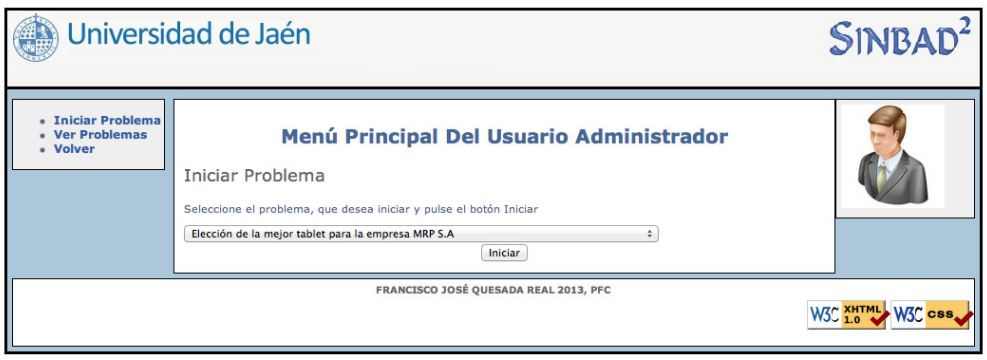


Figura C.5: Iniciar Problema

Finalmente, tras iniciar el problema indicado, el sistema nos llevará a la página principal del usuario administrador (Figura C.6)

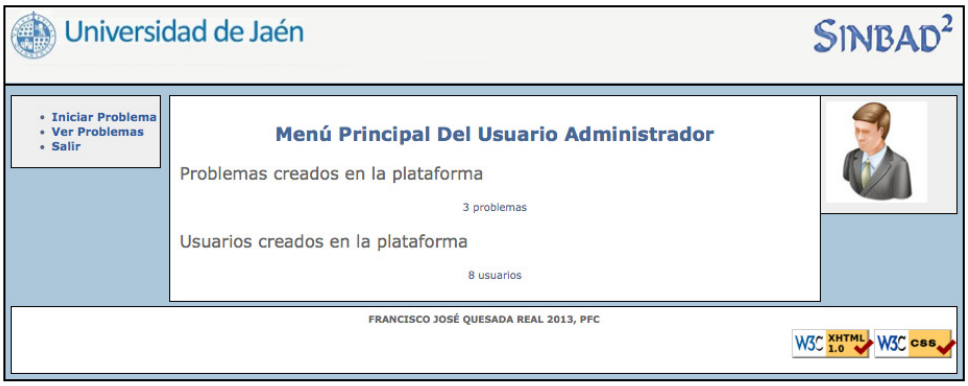


Figura C.6: Menú Principal Usuario Administrador

## C.3. Ver Problemas

Para ver los problemas que hay creados en el sistema, así como los parámetros de los mismos, necesitaremos acceder como usuario Administrador (ver apartado C.1).

Una vez en el menú principal del usuario Administrador, hacemos clic en el apartado *"Ver Problemas"* situado en el menú de la izquierda, tal y como se muestra en la figura C.10 .

Figura C.7: Selección de "Ver Problema"

Tras haber accedido al apartado "Ver Problemas", tendremos que seleccionar en la lista desplegable el problema del que deseemos ver sus parámetros. Después de seleccionar el problema elegido, pulsaremos el botón *"Aceptar"*, como podemos ver en la figura C.11.

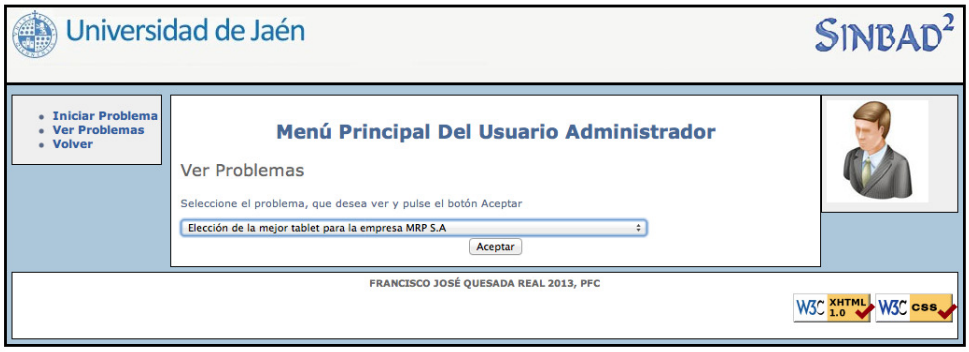


Figura C.8: Selección del problema para ver su parámetros

Una vez que se ha pulsado el botón ' *"Aceptar"*, en la pantalla se visualizarán los parámetros del problema que hayamos seleccionado, tal y como aparece en la figura C.9.

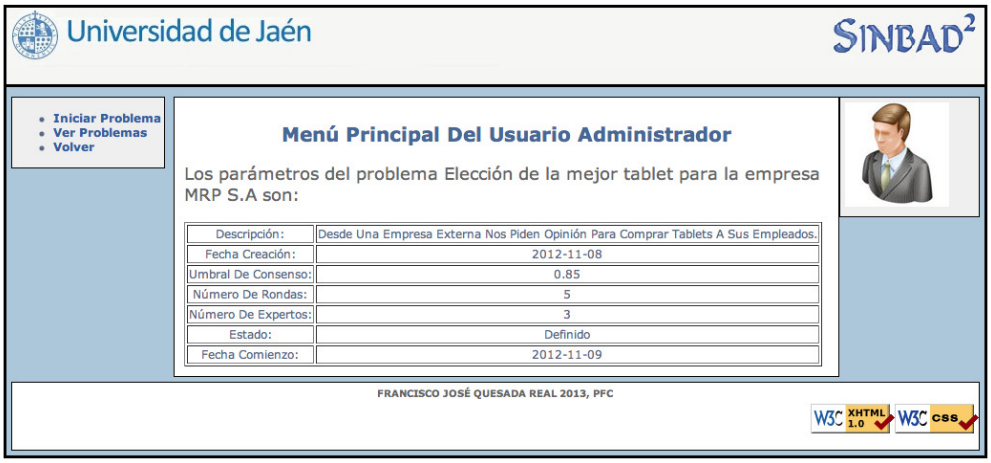


Figura C.9: Parámetros del problema seleccionado

## C.4. Ver Resultados de un Problema

Para ver los resultados de un problema, éste deberá de haberse iniciado previamente (ver apartado C.2).Después, accederemos como usuario Administrador (ver apartado C.1).

Una vez en el menú principal del usuario Administrador, hacemos clic en el apartado *"Ver Problemas"* situado en el menú de la izquierda, tal y como se muestra en la figura C.10.

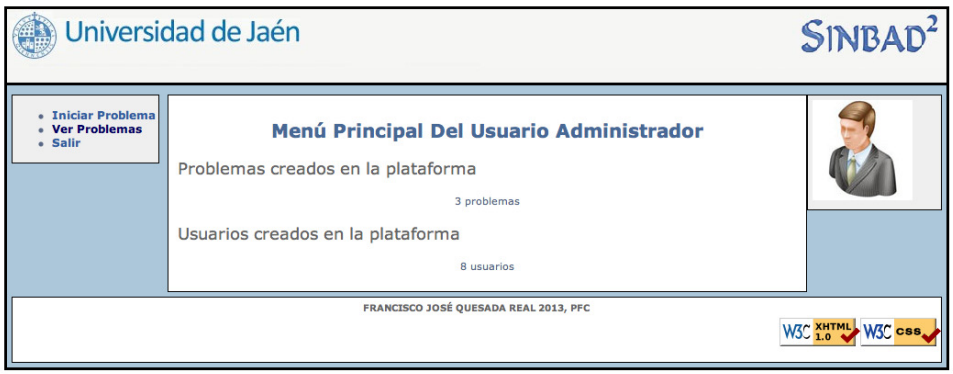


Figura C.10: Selección de "Ver Problema"

Tras haber accedido al apartado "Ver Problemas", tendremos que seleccionar en la lista desplegable el problema del que deseemos ver sus parámetros. Después de seleccionar el problema elegido, pulsaremos el botón *"Aceptar"*, como podemos ver en la figura C.11.

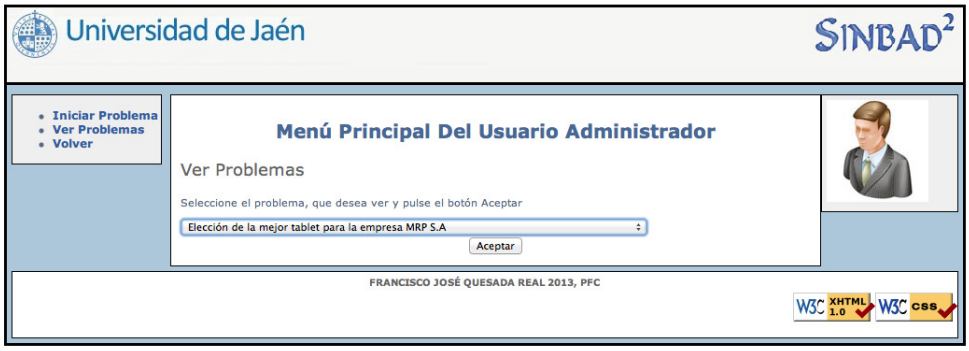


Figura C.11: Selección del problema para ver su parámetros

Una vez que se ha pulsado el botón ' *"Aceptar"*, en la pantalla se visualizarán los parámetros del problema que hayamos seleccionado junto con los resultados del mismo, tal y como aparece en la figura C.12. Destacando el estado del problema y en caso de que esté finalizado se mostrará también el grado de consenso alcanzado y el número de rondas que se han llevado a cabo.

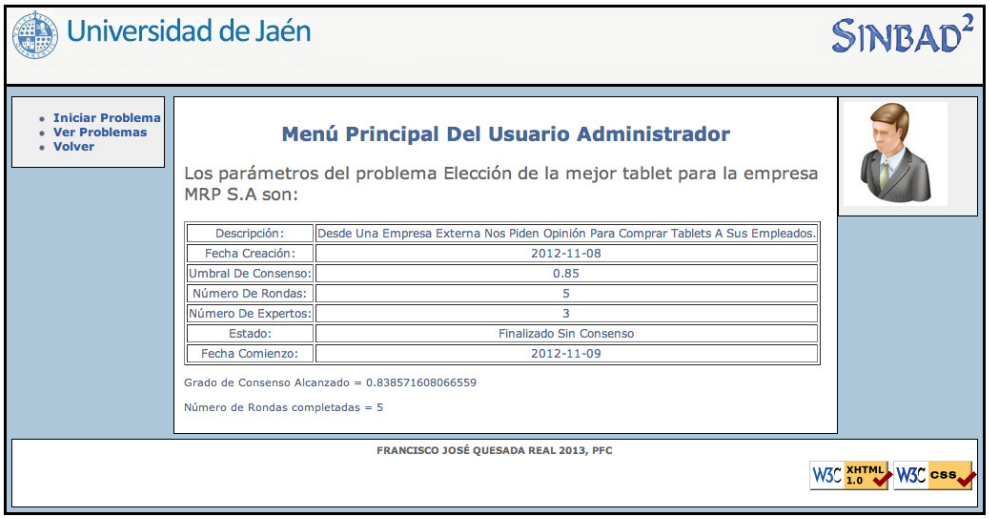


Figura C.12: Resultados del problema seleccionado

## C.5. Salir

El apartado *"Salir"* servirá para abandonar la aplicación e ir a la página de inicio. Por lo tanto, para poder seleccionar esta utilidad es necesario estar dentro de la aplicación (ver apartado C.1).

Únicamente tendremos que localizar, en el menú de la izquierda el item *"Salir"* y hacer clic, como se indica en la figura C.13.

Figura C.13: Seleccionar "Salir"

Tras haber hecho clic en la opción *"Salir"*, estaremos en la página principal, que se muestra en la figura D.12.

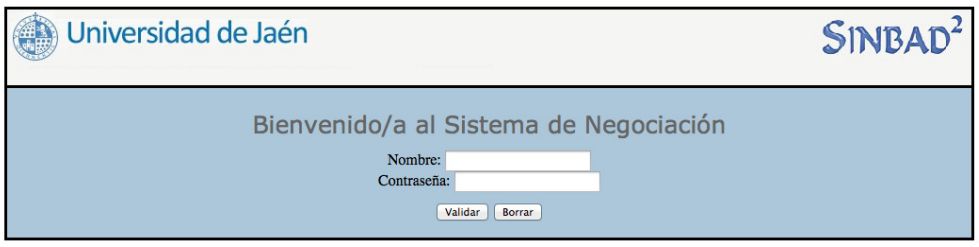


Figura C.14: Página Principal

# Apéndice D

# Manual del Usuario "Experto"

## D.1. Acceso

En primer lugar, lo que deberemos hacer es acceder al sistema como usuario experto. Para ello, abriremos un navegador y pondremos la dirección:

http://localhost:8080/ServletCOMAS

Un vez que hemos realizado el paso anterior, nos aparecerá la página de la figura D.1

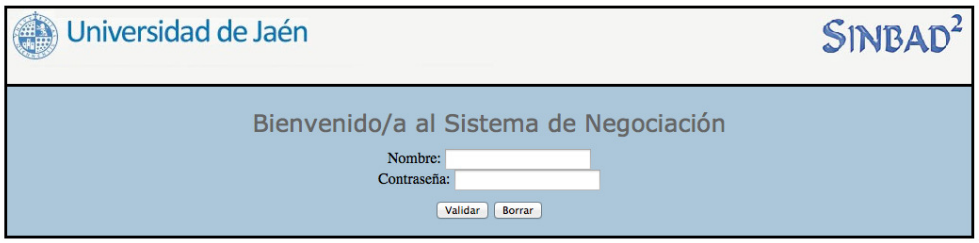


Figura D.1: Página Principal

Seguidamente, rellenamos los campos *Nombre* y *Contraseña* con las credenciales del usuario Experto. En este caso, vamos a entrar con las credenciales del usuario Experto *francis* cuyas credenciales son (*francis, francis*) y pulsamos el botón *Validar*.

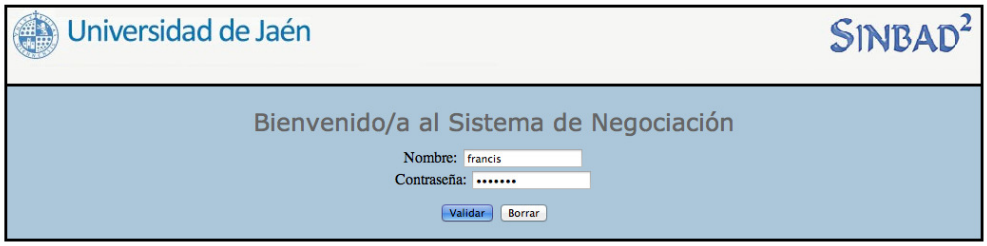


Figura D.2: Acceso como Usuario Experto

Tras haber pulsado el botón validar y si hemos introducidos correctamente las credenciales, accederemos al menú principal del usuario Experto.

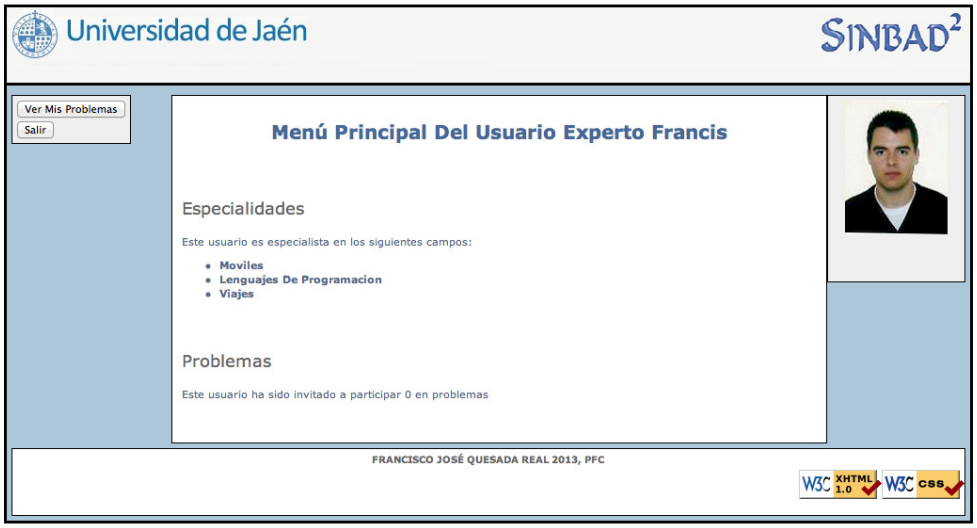


Figura D.3: Menú Principal Usuario Experto

## D.2. Ver Mis Problemas

Para ver los problemas en los que ha participado el usuario Experto, así como los datos de los mismos, necesitaremos acceder al sistema (ver apartado C.1).

Una vez en el menú principal del usuario Experto, hacemos clic en el apartado *"Ver Mis*

*Problemas"* situado en el menú de la izquierda, tal y como se muestra en la figura D.8 .

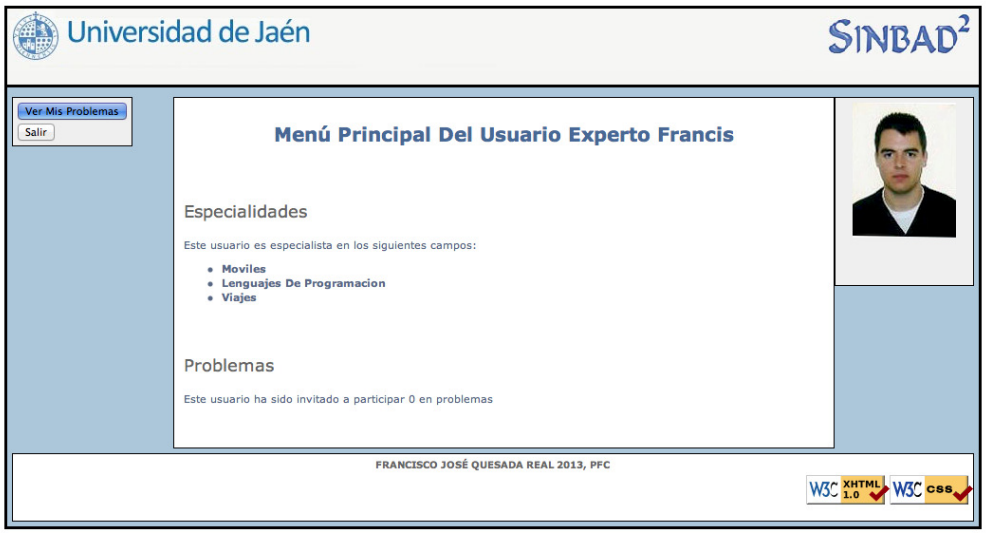


Figura D.4: Selección de "Ver Mis Problemas"

Tras haber accedido al apartado "Ver Mis Problemas", tendremos que seleccionar en la lista desplegable el problema del que deseemos ver sus datos. Después de seleccionar el problema elegido, pulsaremos el botón *"Aceptar"*, como podemos ver en la figura D.9.

Figura D.5: Selección de un problema en el apartado "Ver Mis Problemas"

Después de haber aceptado, veremos los datos del problema seleccionado.

## D.3. Participar en el Proceso de Negociación

El usuario "experto" en primer lugar tendrá que acceder a la página inicial del usuario "Experto", como se describió en el apartado C.1.

Una vez dentro de la página inicial del usuario "Experto", llegarán las invitaciones para participar en procesos de consenso para problemas de TDG pertenecientes a alguna de las especialidades del usuario en cuestión, si los hubiese (ej. gastronomía, viajes,...). Así, se abrirá una ventana, como la que aparece en la figura D.6.

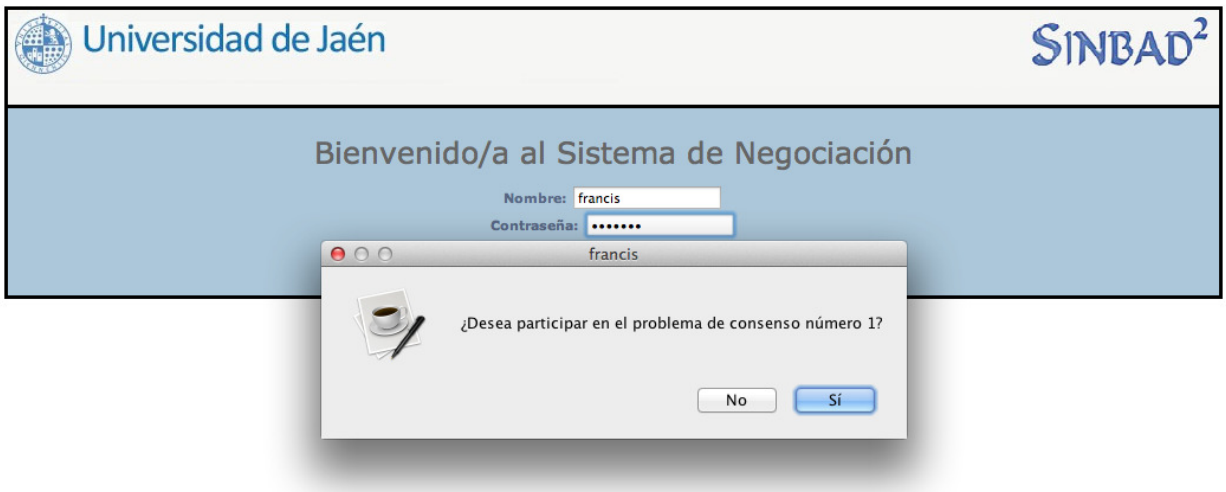


Figura D.6: Ventana Participar en el problema

En este caso podremos aceptar o rechazar la invitación para participar en el problema de consenso. Si rechazamos la invitación nos iremos a la pantalla principal del usuario "Experto". En el caso de aceptar la invitación, nos aparecerá la siguiente pantalla en la que podremos introducir las valoraciones del para el problema indicado, así como elegir la estrategia de negociación que deseemos (ver figura D.7).

Figura D.7: Pantalla Introducir Valoraciones y Seleccionar Estrategia

Finalmente, una vez introducidas las valoraciones y seleccionada la estrategia de negociación, pulsaremos el botón "Enviar", para mandar la información. Posteriormente pulsaremos en "Volver" lo que nos redirigirá a la pantalla principal del usuario "Experto".

## D.4. Ver Resultados de Mis Problemas

Para ver los resultados de los problemas en los que ha participado el usuario Experto, así como los datos de los mismos, necesitaremos acceder al sistema (ver apartado C.1). Éstos se mostrarán sólo cuando se haya iniciado el problema y después de que el usuario Experto haya decido participar en el proceso de negociación (ver apartado D.3 )

Una vez en el menú principal del usuario Experto, hacemos clic en el apartado *"Ver Mis Problemas"* situado en el menú de la izquierda, tal y como se muestra en la figura D.8 .

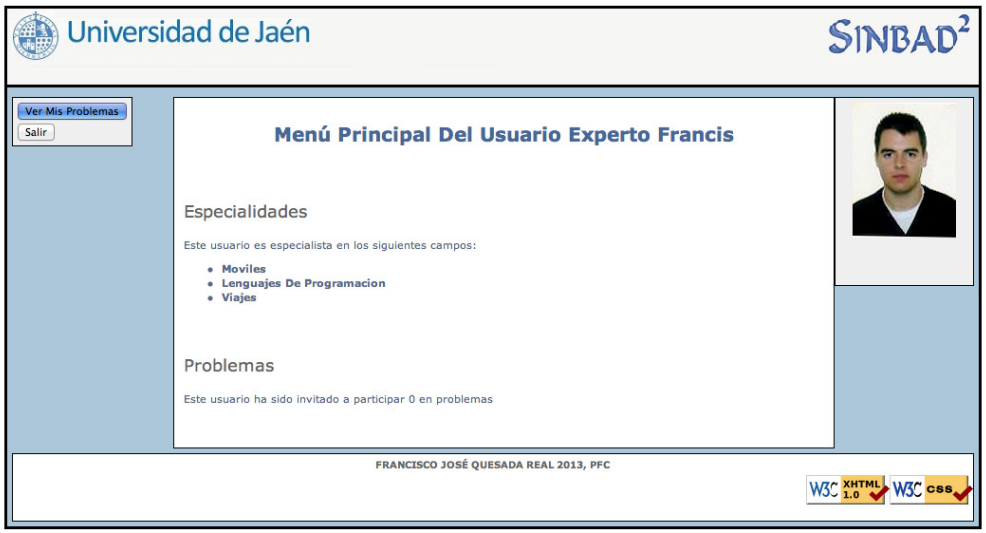


Figura D.8: Selección de "Ver Mis Problemas"

Tras haber accedido al apartado "Ver Mis Problemas", tendremos que seleccionar en la lista desplegable el problema del que deseemos ver sus datos. Después de seleccionar el problema elegido, pulsaremos el botón *"Aceptar"*, como podemos ver en la figura D.9.

Figura D.9: Selección de un problema en el apartado "Ver Mis Problemas"

Después de haber aceptado, veremos los datos del problema seleccionado tal y como muestra la figura D.10. Destacar el valor del parámetro "*estado*", el grado de consenso y el

número de rondas que se han llevado a cabo.

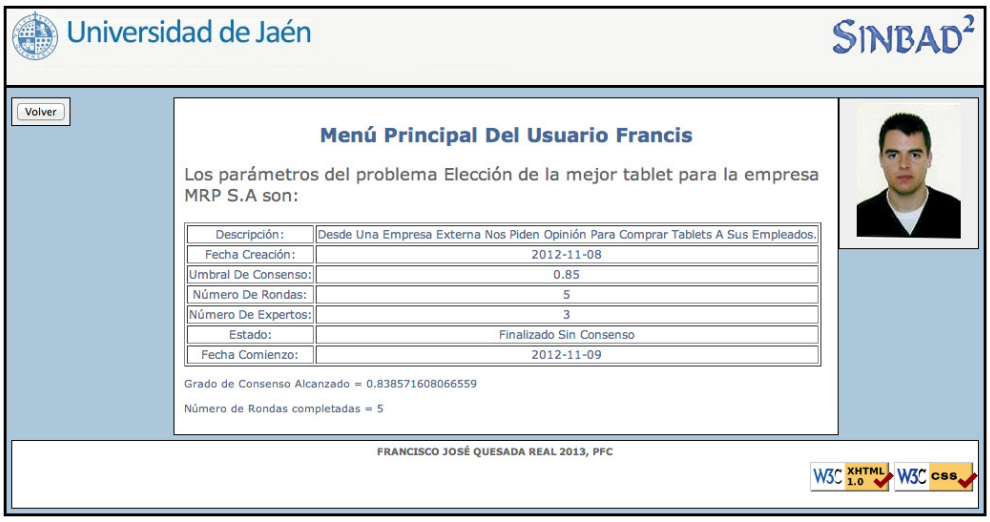


Figura D.10: Resultados de un problema

## D.5. Salir

El apartado *"Salir"* servirá para abandonar la aplicación e ir a la página de inicio. Por lo tanto, para poder seleccionar esta utilidad es necesario estar dentro de la aplicación (ver apartado D.1).

Únicamente tendremos que localizar, en el menú de la izquierda el item *"Salir"* y hacer clic, como se indica en la figura D.11.

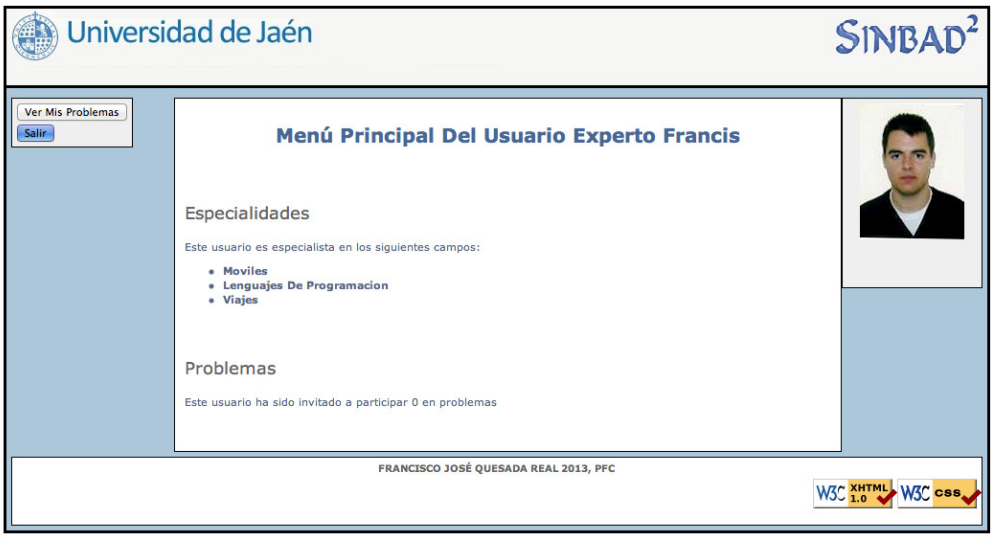


Figura D.11: Seleccionar "Salir"

Tras haber hecho clic en la opción *"Salir"*, estaremos en la página principal, que se muestra en la figura D.12.

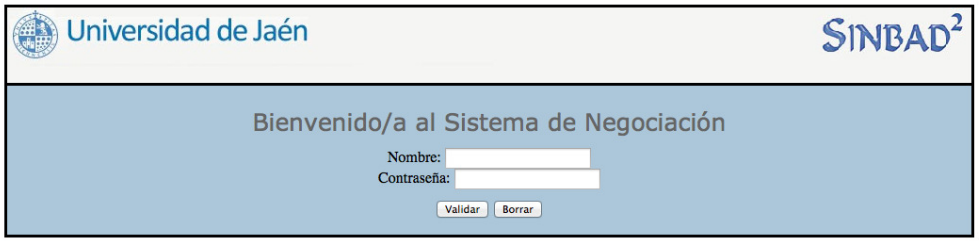


Figura D.12: Página Principal

# Bibliografía


[1] Auml. http://www.auml.org. Revisado en enero 2011.

[2] Daml web site: www.daml.org. Consultado en agosto 2012.

[3] Fipa. http://www.fipa.org. Revisada enero 2011.

[4] Foaf website: http://xmlns.com/foaf/spec. Consultado en agosto 2012.

[5] Health on the net foundation. http://www.idshealthcare.com. Revisado noviembre 2010.

[6] Jade. http://jade.tilab.com. Revisado en enero 2011.

[7] Kif website: http://logic.stanford.edu/kif/kif.html. Consultado en agosto 2012.

[8] Lenguaje de progamación java. http://www.java.com/es/. Revisado enero 2010.

[9] Mercapesca. http://www.mercapesca.net. Revisada noviembre 2010.

[10] Multi-agent development kit. http://www.madkit.org. Revisado noviembre 2010.

[11] Owl website: www.w3.org/tr/2002/wd-owl-ref-20020729. Consultado en agosto 2012.

[12] Página oficial proyecto masfit. www.masfit.net. Revisada noviembre 2010.

[13] Portal hipertexto. www.hipertexto.info/documentos/ontologias.htm. Consultado en agosto 2012.

[14] Rdf-schema. www.w3.org/tr/rdf-schema. Consultado en agosto 2012.

[15] Real academia española. http://www.rae.es. revisada enero 2011.

[16] Web services. www.w3.org/2001/sw/. Consultado en agosto 2012.

[17] Webmate. http://www.cs.cmu.edu/ softagents/webmate/introduction.html. Revisada noviembre 2010.

[18] *The Analysis of Fuzzy Information, in On Some Fuzzy Cores and "SoftÇonsensus Measures in Group Decision Making*, 1987.

[19] *Multiagent meeting scheduling: Preliminary experimental results*, volume 95-102. In Proceedings of the Second International Conference on Multiagent Systems, 1996.

[20] *A meta-model for the analysis and design of organizations in multi-agent systems*, number 128-135. IEEE computer soc, 10662 Los vaqueros circle, po box 3014, Los alamitos, ca 90720-1264 USA, 1998.

[21] *Agent Oriented Analysis using MESSAGE/UML*. Springer-verlag Berlin, Heidelberger platz 3, d-14197 Berlin, Germany, 2001.

[22] *Decision-Making Embedded Devices using FIPA-ACL Communication*, number 696-701. IEEE, 345 e 47th st, New York, ny 10017 USA, 2009.

[23] Basilevsky A. Applied matrix algebra in the statistical sciences. *Elsevier*, 1993.

[24] Flores-Mendez Roberto A. Towards a standardization of multi-agent system frameworks. Technical report, ACM Crossroads, Special Issue on Intelligent Agents,Association for Computer Machinery, 1999.

[25] Mas A. *Agentes Software y sistemas multi-agente: Conceptos, arquitecturas y aplicaciones*. Pearson Education, 2004.

[26] Newell A. The knowledge level. *Artificial Intelligence*, (18):87–127, 1982.

[27] Elamy A.H. Perspectives in agent-based technology. *AgentLink News*, 18:19–22, 2005.

[28] Mulet Luis Alberola Juan M, Such Jose M. Diseño de magentix. Technical report, Grupo de Tecnología Informática- Inteligencia Artificial. UPV, 2007.

[29] Hart S Aumman R J. *Handbook of Game Theroy. With Economy Applications, Volumen 2*. Netherlands: North Holland, 1994.

[30] Odell J Bauer B, Müller J P. Agent uml: A formalism for specifying multianget interaction. *International Journal of Software Engineering and Knoeledge*, 11(3), 2001.

[31] Greenwood D Bellifemine F, Caire G. *Developing Multi-agent systems with JADE*. Wiley, 2007.

[32] Chen Z Ben-Arieh D. Linguistic labels aggregation and consensus measure for automatic decision-making using group recommendations. *IEEE transactions on systems man and cybernetics Part A: Systems and Humans*, 36(1):558–568, 2006.

[33] Evans B Ben-Arieh D, Easton T. Minimum cost consensus with quadratic cost functions. *IEEE transactions on systems man and cybernetics Part A: Systems and Humans*, 39(1):210–217, 2009.

[34] Spillman R Bezdek J C, Spillman B. A fuzzy relation space for group decision theory. *Fuzzy sets and Systems*, 1(25), 1978.

[35] Pilgrim JR Mills WN Diao Y Bigus JP, Schlosnagle DA. Able: A toolkit for building multiagent autonomic systems. *IBM systems journal*, 41(3):350–371, 2002.

[36] A. H. Bond and L. Gasser. *Readings in Distributed Artificial Intelligence*. Morgan Kauffmann, 1988.

[37] Pasi G Bordogna G, Fedrizzi M. A linguistic modeling of consensus in group decision making based on owa operators. *IEEE transactions on systems man and cybernetics Part A: Systems and Humans*, 27(7), 1997.

[38] V Botti, V; Julián. Agentes inteligentes: el siguiente paso en la inteligencia artificial. *Novatica*, pages 95–99, 2000.

[39] Pollack M E Bratman M E, Israel D J. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4):1–22, 1988.

[40] Pollack M E Bratman M E, Israel D J. Intention, practical rationaly and selt-governance. *Ethics*, 119(1):411–443, 2998.

[41] Rothstein A Butler C T L. On conflict and consensus: A handbook on formal consensus decision making. *Takoma Park*, 2006.

[42] Tordera J C. *El abecé de la Lingüística computacional*. Arco Libros, 2012.

[43] Herrera-Viedma E Cabrerizo F J, Pérez I J. Managing the consensus in group decision making in an unbalanced fuzzy linguistic context incomplete information. *Knowledge-Based Systems*, 23:169–181, 2010.

[44] Ceballos H G Cantu F J. A multiagent knowledge and information network approach for managing research assets. *Expert Systems with Applications*, 37(7):5272–5284, 2010.

[45] Eklund P Fedrizzi M Carlsson C, Ehrenberg D. Consensus in distribuited soft enviorments. *European Journal of Operational Research*, 61:165–185, 1992.

[46] Alvarez E Castillo E. Expert systems: Uncertainty and learning. *Computational Mechanics Publications and Elsevier Applied Science*, 1991.

[47] Maes P. Chavez A. Kasbah: An agent marketplace for buying and selling goods. *MIT Media Lab*, 1996.

[48] Seiford L Cook W. Priority ranking and consensus formation. *Management Science*, 24:1721–1732, 1978.

[49] Larman Craig. *Uml y Patrones*. Prentice Hall, 2006.

[50] Vila M A Delgado M, Verdegay J L. A model for linguistic partial information in decision making problem. *International Journal of Intelligent Systems*, 9(5):365–378, 1994.

[51] Sparkman CH Deloach SA, Wood MF. Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, JUN 2001.

[52] V. R. Lesser E. H. Durfee and D. D. Corkill. Trends in cooperative distributed problem solving. Technical report, IEEE Transactions on Knowledge and Data Engineering, 1989.

[53] Pereira R A M Fedrizzi M. Soft consensus and network dynamics in group decision making. *International Journal of Intelligent Systems*, 14(1):63–77, 1999.

[54] Garijo F.J. Tecnología de agentes: Experiencias y perspectivas para el desarrollo de nuevos servicios y aplicaciones. *Bole.tic*, (24):1–9, Noviembre-Diciembre 2002.

[55] Francisco García Sánchez. Sistema basado en tecnología del conocimiento para entornos de servicios web semánticos. Master's thesis, Universidad de Murcia, 2007.

[56] Martínez R Fernández J T García Sánchez F, Valencia R. An ontology, intelligent agent-based framework for the provision of semantic web services. *Expert Systems with Applications*, 36(2):3167–3187, 2009.

[57] Garijo F Gomez-Sanz J J, Pavón J. Meta-modelling of multi-agent systems. In *Simposium on Applied Computing*. ACM, 2002.

[58] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.

[59] T. R. Gruber. Interview to tom gruber. *AIS SIGSEMIS Bulletin*, 1(3):4, October 2004.

[60] N Guarino. Formal ontology and information systems. In *1st International Conference on Formal Ontology in Information Systems*, 1998.

[61] Herweyers G Janssen M Hamer L, Hemercyck Y. Similarity measures in scientometric research: The jaccard index versus salton's cosine formula. *Information Processing and Management*, 25(3):315–318, 1989.

[62] Herrera-Viedma E Herrera F. Linguistic decision analysis: Steps for solving decision problems under linguistic information. Technical report, Department of Computer Science and Artificial Intelligence, 1998.

[63] Verdegay J Herrera F, Herrera-Viedma E. A model of consensus in group decision making under linguistic assessments. *Fuzzy sets and Systems*, 78(1):73–87, 1996.

[64] Mata F Chiclana F Herrera-Viedma E, Martínez L. A consensus support system model for group decision making problems with multigranular linguistic preference relations. *IEEE Transactions on Fuzzy Systems*, 13(5):644–658, 2005.

[65] Singh M P Huhns M N. Cognitive agents. *Internet Computing*, 2(6):87–89, 1998.

[66] Sommerville Ian. *Ingeniería del Software*. Prentice - Hall, 2010.

[67] IEEE, editor. *Optimizing KQML for usage in power distribution network simulator*, number 846-849. 2008 3rd International symposium on communications, control and signal processing, vols 1-3, IEEE, 345 e 47th st, New York, ny 10017 USA, 2008.

[68] IEEE Proc. on Software Engineering. *Agent-based software engineering*, volume 144, 1997.

[69] Intelligent Agents: Theories, Architectures and Languages (LNAI 890). *Modelling reactive behaviour in vertically layered agent architecture*, 1995.

[70] International Conference on Intelligent Systems Design and Applications (ISDA 2010). *COMAS: A Consensus Multi-Agent Based System*, El Cairo, 2010.

[71] Durkin J. Expert systems: Design and development. In *Maxwell Macmillan, New York*, 1994.

[72] Ferber J. *Multiagent Systems - An Introduction to Ditributed Artificial Intelligence*. Addison Wesley, 1999.

[73] Garcia A; Lucena C; Zambonelli F; Omicini A; Castro J, editor. *The RETSINA MAS, a case study*, volume 2603. Software engineering for large-scale multi-agent systems - research issues and practical applications, Springer-verlag Berlin, Heidelberger platz 3, d-14197 Berlin, Germany, 2003.

[74] Kacprzyk J. Group decision making with a fuzzy linguistic majority. *Fuzzy sets and Systems*, 18(14), 1986.

[75] Lind J. Massive: Software engineering for multiagent systems. Master's thesis, University of the Saarland, 2000.

[76] Fedrizzi M Kacprzyk J. A "soft"measure of consensus in the setting if partial (fuzzy) preferences. *European Journal of Operational Research*, 34(1):316–325, 1988.

[77] Nurmi H Kacprzyk J, Fedrizzi M. Group decision making and consensus under fuzzy preferences and fuzzy majority. *Fuzzy sets and Systems*, 49(1):21–31, 1992.

[78] Zadrozny S Kacprzyk J. On a concept of a consensus reaching process support system based on the use of soft computing and web techniques. *Computational Intelligence and Decision Control*, 1:859–864, 2008.

[79] Zadrozny S Kacprzyk J. Towards a general and unified characterization of individual and collective choice functions under fuzzy and nonfuzzy preferences and majority via the ordered weighted average operators. *International Journal of Intelligent Systems*, 24(1):4–26, 2009.

[80] Zadrozny S Kacprzyk J. Soft computing and web intelligence for supporting consensus reaching. *Soft Computing*, 14(8):833–846, JUN 2010.

[81] Han C H Kim S H Kim J K, Choi S H. An interactive procedure for multiple criteria group decision making with incomplete information. *Computers and Industrial Engineering*, 35(1,2):295–298, 1998.

[82] Rao A Kinny D, Georgeff M. A methodology and modelling technique for systems of bdi agents. *Australian Artificial Intelligence Institute*, 1997.

[83] Yuan B Klir G J. *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice Hall, 1995.

[84] Evenchik A Kraus S, Sycara K. Reaching agreements through argumentation: a logical model and implementation. *Artificial Intelligence*, 104(1-2):1–69, SEP 1998.

[85] García-Lapresta J L. *Some consensus measures and their applications in group decision making*. 2008.

[86] Luck M; Padgham L, editor. *An agent framework for processing FIPA-ACL messages based on interaction models*, volume 4951. Springer-verlag Berlin, Heidelberger platz 3, d-14197 Berlin, Germany, 2008.

[87] Stevens L. *Artificial Intelligence. The earch for the Perfect Machine*. Hayden Book Company, Hasbrouck Heights, N.J, 1984.

[88] C. M. Lewis and K. Sycara. Reaching informed agreement in multispecialist cooperation. *Group Decision and Negotiation*, 2(3):279–300, 1993.

[89] Wang Y C Li D F. Mathematical programming approach to multiattribute decision making under intuitionistic fuzzy environments. *International Journal of Uncertainty Fuzziness and Knowledge-Based System*, 16(4):557–577, 2008.

[90] Martinez T Louis V. The jade semantic agent: Towards agent communication oriented middleware. *AgentLink News*, 18:16–18, 2005.

[91] J. Luke, S. Heflin. Shoe 1.01. propose specification. *SHOE Project*, 2000.

[92] Gervais M. Odac: An agent-oriented methodology based on odp. *Autonomous agents and Multi-Agents systems*, 7(3):199–228, 2003.

[93] Lee H M. Generalization of the group decision making using fuzzy sets theory for evaluating the rate of aggregative risk in software development. *Information Sciences*, 1999.

[94] Morales M. *Ontologías. En Representación del conocimiento.* Alianza Editorial, 2000.

[95] Ricordel P M. Programmation orientée multi-agents, développement et déploiement de systèmes multi-agents voyelles. Master's thesis, Institut National PolyTechnique de Grenoble, 2001.

[96] Roubens M. Fuzzy sets and decision analysis. *Fuzzy sets and Systems*, 90(2):199–206, 1997.

[97] Man and Cybernetics Conference. *A bdi agent architecture for reasoning about reputation*, volume 2(1). IEEE Systems, 2001.

[98] Montero J Martínez L. Challenges for improving consensus reaching process in collective decisions. *New Mathematics and Natural Computation*, 3(2):203–217, 2007.

[99] Herrera-Viedma E Mata F, Martinez L. An adaptative consensus support model for group decision-making problems in a multigranular fuzzy linguistic context. *IEEE Transactions on Fuzzy Systems*, 17(2):279–290, 2009.

[100] Roure de D C Middleton S E, Shadbolt N R. Ontological user profiling in recommender systems. *ACM Transactions on Information Systems*, 22(1):54–88, 2004.

[101] Bryson N. Group decision-making and the analytic hierarchy process exploring the consensus-relevant information content. *Computers and Operational Research*, 1(23):9, 1996.

[102] A. R. Lomuscio S. Parsons C. Sierra N. R. Jennings, P. Faratin and M. Wooldridge. Automated negotiation: Prospects, methods and challenges. *International Journal of Group Decision and Negotiation*, 10(2):199–215, 2001.

[103] K. Sycara N. R. Jennings and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems Journal*, 1(1):7–38, 1998.

[104] Jennings N. R. Narayanan V. An adaptative bilateral negotiation model for e-commerce settings. In *7 th IEEE International Conference on E-Commerce Technology*, pages 34–41, 2005.

[105] Netherlands Energy Research Foundation ECN and Frree University of Brussels. *CommonKADS: A comprehensive methodology for KBS development*, 1994.

[106] Maes P. Situated agents can have goals. *Robotic and Autonomus Systems*, 6(1-2):49–70, 1990.

[107] Maes P. Agents that reduce work and information overload. *Communication of the ACM*, 37(7):31–40, 1994.

[108] Sycara K P. Multiagent systems. *AI magazine*, 19(2):79–92, SEP 1998.

[109] Ivan Palomares Carrascosa. *Desarrollo de un Sistema Multi-Agente para automatizar Procesos de Consenso en Problemas de Toma de Decisión en Grupo.* DEC 2010.

[110] Jennings N Parsons S, Sierra C. Agents that reason and negotiate by arguing. *Journal of logic and computation*, 8(3):261–292, JUN 1998.

[111] Odell J Parunak H V D. *Representing social structures in UML.* 2002.

[112] Gómez-Sanz J J Pavón J. Agent oriented software engineering with ingenias. In *In Multi-Agent Systems and Applications III*, pages 394–403. 3er International Central and Easter Conference on Multi-Agent Systems, Springer-verlag Berlin, Heidelberger platz 3, d-14197 Berlin, Germany, 2003.

[113] Principles of knowledge representation and reasoning - proceedings of the second international conference ( kr91 ). *Modeling Rational Agents within a BDI-architecture*, number 473-484. Morgan Kauffmann, 1991.

[114] Proccedings of the Third European Workshop Autonomus Agents and Multi-Agents World. *Towards an architecture for adaptative, rational, mobile agents*, number 249-262, 1991.

[115] Meyer JJC; Schobbens PY Meyer JJC; Schobbens PY, editor. *Modelling internal dynamic behaviour of BDI agents*, volume 1760. Springer-verlag Berlin, Heidelberger platz 3, d-14197 Berlin, Germany, 1999.

[116] Brooks R. Intelligence without representation. *Artificial Intelligence*, 47(1):139–159, 2001.

[117] Dignum F; VanEijk RM; Flores R, editor. *ACL semantics between social commitments and mental attitudes*, volume 3859. Springer-verlag Berlin, Heidelberger platz 3, d-14197 Berlin, Germany, 2006.

[118] Dignum F; VanEijk RM; Flores R, editor. *An operational model for the FIPA-ACL semantics*, volume 3859. International Workshop on Agent Communication Hakodate, JAPAN, MAY 09, 2006, Springer-verlag Berlin, Heidelberger platz 3, d-14197 Berlin, Germany, MAY 2006.

[119] Lindemann G; Denzinger J; Timm IJ; Unland R, editor. *Towards an approach for debugging MAS through the analysis of ACL messages*, volume 3187. Multiagent System Technologies, Proceedings, Springer-verlag Berlin, Heidelberger platz 3, d-14197 Berlin, Germany, 2004.

[120] Yager R R. Non-numeric multi-criteria multi-person decision making. *Group Decision and Negotiation*, 2(1):81–93, 1993.

[121] Yager R R. Fusion of ordinal information using weigthed mean aggregation. *International Journal of Approximate Reasoning*, 12(1-2):35–52, 1998.

[122] Jennings N R McBurney P Parsons S Sonenberg L Rahwan I, Ramchurn S D. Argumentation-based negotiation. *The knowledge Engineering Review*, 18(4):343–375, 2004.

[123] Georgeff M P Rao A S, editor. *BDI Agents: From theory to practice*. Proceeding of the First International Conference on Multiagent Systems, 1995.

[124] Georgeff M P Rao A S. Decision procedures for bdi logics. *Journal of logic and computation*, 8(3):293–343, JUN 1998.

[125] Julian Vicente Rebollo M, Carrascosa C. Modelo de agente magentix. Technical report, Grupo de Tecnología Informática- Inteligencia Artificial. UPV, 2006.

[126] Mateos A Rios S, Bielza C. *Fundamentos de los sistemas de ayuda a la decisión*. RA-MA, 2001.

[127] S Russell. *Inteligencia Artificial: un enfoque moderno*. Prentice - Hall, Mexico, 1996.

[128] Black M Russell B. *La Justificación Del Razonamiento Inductivo*. Alianza Editorial, 1974.

[129] Nolfi S. Power and the limits of reactive agents. *Neurocomputing*, 42(1):119–145, 2002.

[130] Moore SA. Kqml and flbc: Contrasting agent communication languages. *International Journal of Electronic Commerce*, 5(1):109–124, 2000.

[131] Lawson J R Saint S. *Rules for Reaching Consensus. A Modern Approach to Decision Making*. Jossey-Bass, 1988.

[132] Wooldridge M J Singh M P, Rao A, editor. *Analysis and design of multiagent systems using MAS-CommonKADS*. Springer-verlag Berlin, Heidelberger platz 3, d-14197 Berlin, Germany, 1998.

[133] Pisanelli D. M Steve G, Gangemi A. Integrating medical terminalogies with onions methodology. Technical report, ITBM-CNR, 1997.

[134] Fensel D Studer R, Benjamins R. Knowledge engineering: Principles and methods. *Data and Knowledge Engineering*, 25(1-2):161–197, 1998.

[135] Van Velsen M Giampapa J Sycara K, Paolucci M. The retsina mas infrastructure. *Autonomous agents and Multi-Agents systems*, 7(1-2):29–48, JUL-SEP 2003.

[136] Quine W. V. From a logical point of view, nine logico-philosophical essays. Technical report, Cambridge, Massachusetts: Hardvard University Press, 1961.

[137] Wielinga B. J van Heijst G, Schreiber A.T. Using explicit ontologies in kbs development. *International Journal of Human-Computer Studies*, 46(2):183–292, 1997.

[138] Sandholm T W. Distributed rational decision making. *Multiagent Systems- A Modern Apporoach to Distributed Artificial Intelligence*, pages 201–258, 1999.

[139] Kinny D Wooldridge M, Jennings N R. The gaia methodology for agent-oriented analysis and design. *Autonomous agents and Multi-Agents systems*, 3(3):285–312, SEP 2000.

[140] Bui T X. A group decision support system for cooperative multiple criteria decision making. *Springer-Verlang*, 1987.

[141] Fan XC Xu DX, Zheng GL. A logic based language for networked agents. *Information and Software Technology*, 40(8):435–442, AUG 1998.

[142] Demazeau Y. *La méthode VOYELLES dans Systèmes Multi-Agents: Des Theories Organisationnelles aux Applications Industrielles*. 2001.

[143] Wooldridge M Zambonelli F, Jennings N R. Developing multiagent systems: The gaia methodology. *ACM Trans. Softw. Eng. Methodol*, 12(3):317–370, 2003.

[144] A Zapata García. *Desarrollo de una Ontología para un Porta Inmobiliario*. PhD thesis, Universidad de Jaén, 2009.

[145] Lu W F Zha X F, Lim S Y E. A knowledge intensive multi-agent framework for cooperative/colaborative design modeling and decision support of assemblies. *Journal of Integrated Design and Process Science*, 7(1):99–122, 2002.

[146] Chong P P Zhang Q, Chen J C H. Decision consolidation criteria weight determination using multiple formats. *Decision Support Systems*, 38(2):247–258, 2004.