UNIVERSIDAD DE MURCIA

Departamento de Ingeniería de la Información y las Comunicaciones

SISTEMA BASADO EN TECNOLOGÍAS DEL CONOCIMIENTO PARA ENTORNOS DE SERVICIOS WEB SEMÁNTICOS

Francisco García Sánchez

JULIO 2007

Directores:

Dr. Rodrigo Martínez Béjar
Dr. Rafael Valencia García

Agradecimientos

A Loli.
A toda mi familia y amigos.
A los compañeros de investigación con los
que he trabajado durante estos años.

ÍNDICES

ÍNDICE DE CONTENIDO

ÍNDICES v
Índice de Contenido vi
Índice de Figuras xiii
Índice de Tablas xx
INTRODUCCIÓN 1
CAPÍTULO I. ONTOLOGÍAS, SERVICIOS WEB SEMÁNTICOS Y AGENTES INTELIGENTES – ESTADO DEL ARTE 5
I.1. Introducción 5
I.2. Ontologías 6
I.2.1. Definición 6
I.2.2. Tipos de Ontologías 9
I.2.2.1. Clasificación por el conocimiento que contienen 9
I.2.2.2. Clasificación por motivación 10
I.2.3. Formalismos lógicos y razonamiento con ontologías 10
I.2.3.1. Lógica de primer orden 11
I.2.3.2. Lógica descriptiva 11
I.2.3.3. Programación lógica 13
I.2.4. Lenguajes para la representación de ontologías 14
I.2.4.1. Resource Description Framework (RDF) 15
I.2.4.2. RDF Schema (RDFS) 17
I.2.4.3. Web Ontology Language (OWL) 18
I.2.4.4. Web Services Modeling Language (WSML) 21
I.3. Tecnología de Agentes 23
I.3.1. Agentes Inteligentes 23
I.3.2. Sistemas Multi-Agente 25
I.3.3. Negociación en Sistemas Multi-Agente 27
I.3.4. Metodologías para el diseño de Sistemas Multi-Agentes 31
I.3.4.1. INGENIAS 32
I.3.4.2. Otras 34

I.3.5. Plataformas Multi-Agente 37
I.3.5.1. Modelo de Referencia FIPA 37
I.3.5.2. Jade 41
I.3.5.3. Otras 46
I.3.6. Problemas y Carencias de los Sistemas Multi-Agente 48
I.4. Servicios Web Semánticos 50
I.4.1. La Web Semántica 51
1.4.1.1. Antecedentes 51
1.4.1.2. Fundamentos de la Web Semántica 52
1.4.1.3. Arquitectura de la Web Semántica 54
1.4.1.4. Ventajas e Inconvenientes 55
I.4.2. Servicios Web 57
1.4.2.1. Antecedentes 58
1.4.2.2. Componentes de la Arquitectura 61
1.4.2.3. Arquitectura Orientada a Servicios (SOA) 69
1.4.2.4. Ventajas e Inconvenientes 71
I.4.3. Servicios Web Semánticos 73
I.4.3.1. Aproximaciones a los Servicios Web Semánticos 73
I.4.3.2. Entornos de Ejecución de Servicios Web Semánticos 89
I.4.4. Problemas y Carencias de los Servicios Web Semánticos 98
I.5. Problema a Resolver 99
I.6. Resumen 100
CAPÍTULO II. INTEGRACIÓN DE AGENTES INTELIGENTES Y SERVICIOS WEB SEMÁNTICOS – TRABAJO RELACIONADO 105
II.1. Introducción 105
II.2. GODO (Goal-Oriented DiscOvery for Semantic Web Services) 106
II.3. SWF (Semantic Web Fred) 109
II.4. Servicios Web y Comportamientos de Agente 112
II.5. Task Computing 115
II.6. Otras metodologías 118
II.6.1. Interoperabilidad entre Agentes Software y Servicios Web 118

II.6.2. Servicios Web para Implementar Sistemas Multi-Agente 122
II.6.3. Agentes Inteligentes y la Web Semántica 126
II.6.4. Entornos de Ejecución de Servicios Web Semánticos 128
II.7. Problemas de las soluciones disponibles actualmente 129
II.8. Resumen 130
CAPÍTULO III. SISTEMA BASADO EN TECNOLOGÍAS DEL CONOCIMIENTO PARA ENTORNOS DE SERVICIOS WEB SEMÁNTICOS 133
III.1. Introducción 133
III.2. Objetivos de Diseño 136
III.3. Diseño y especificación 138
III.3.1. Casos de Uso 139
III.3.2. Modelo de Organización 143
III.3.3. Modelo de Agentes 145
III.3.4. Modelo de Tareas y Objetivos 147
III.3.5. Modelo de Interacción 150
III.3.6. Modelo de Entorno 152
III.4. Arquitectura 154
III.4.1. Sistema Multi-Agente 155
III.4.1.1. Agentes Inteligentes involucrados 156
III.4.1.2. Roles 163
III.4.1.3. Bases de Conocimiento 167
III.4.2. Integración con los Servicios 169
III.4.2.1. Descubrimiento 171
III.4.2.2. Composición 175
III.4.2.3. Selección 176
III.4.2.4. Invocación 179
III.4.2.5. Monitorización 180
III.4.2.6. Negociación 181
III.4.3. Mediación 182
III.4.3.1. Mediación de datos 183
III.4.3.2. Mediación de protocolos 185

III.4.3.3. Mediación de procesos 187
III.4.3.4. Mediación de funcionalidad 190
III.4.4. Usuarios Externos 190
III.4.4.1. Desarrolladores 191
III.4.4.2. Proveedores de servicios 194
III.4.4.3. Consumidores de servicios 196
III.5. Resumen 199
CAPÍTULO IV. CASO DE USO – EJEMPLOS DEL SISTEMA 203
IV.1. Introducción 203
IV.2. Tarea de Ejemplo 1 – descubrimiento, selección e invocación de servicios 203
IV.2.1. Escenario 204
IV.2.2. Aplicación del marco de trabajo 205
IV.2.3. Diferencias con otras plataformas 211
IV.3. Tarea de Ejemplo 2 – composición de servicios 212
IV.3.1. Escenario 212
IV.3.2. Aplicación del marco de trabajo 214
IV.3.3. Diferencias con otras plataformas 219
IV.4. Tarea de Ejemplo 3 – acceso integrado a fuentes de datos heterogéneas 220
IV.4.1. Escenario 220
IV.4.2. Aplicación del marco de trabajo 222
IV.4.3. Diferencias con otras plataformas 226
IV.5. Resumen 227
CAPÍTULO V. UNA APLICACIÓN SOFTWARE PARA LA EJECUCIÓN DINÁMICA DE SWS .. 231
V.1. Introducción 231
V.2. Fundamentos tecnológicos 231
V.2.1. Herramientas 231
V.2.1.1. JADE 232
V.2.1.2. Tomcat 232
V.2.1.3. Sesame 232
V.2.1.4. MySQL 233
V.2.1.5. Java y Eclipse 234

V.2.2. Librerías 234
V.2.2.1. JadeGateway 235
V.2.2.2. Jena 236
V.2.2.3. Axis2 237
V.2.2.4. KAText 238
V.2.2.5. Planificador 240
V.3. Framework Multi-Agente para la Ejecución Dinámica de SWS 241
V.3.1. Agentes, comportamientos y roles 242
V.3.2. Tareas básicas 246
V.3.2.1. Procesar las consultas de los usuarios 246
V.3.2.2. Descubrimiento y composición 248
V.3.2.3. Selección 252
V.3.2.4. Invocación 257
V.3.3. Arquitectura de la aplicación Web 260
V.4. Uso de la Aplicación 263
V.4.1. Interfaz con el Desarrollador 264
V.4.1.1. Gestión de repositorios: añadir y eliminar 264
V.4.1.2. Gestión de ontologías: añadir y eliminar 265
V.4.1.3. Gestión de roles: implementación 267
V.4.1.4. Gestión de agentes: instanciación 268
V.4.2. Interfaz con el Usuario: Proveedor de Servicios 269
V.4.2.1. Gestión de proveedores: altas, bajas y modificaciones 270
V.4.2.2. Gestión de servicios: altas, bajas y modificaciones 272
V.4.3. Interfaz con el Usuario: Consumidor de Servicios 275
V.4.3.1. Gestión de consumidores: altas, bajas y modificaciones 276
V.4.3.2. Gestión de consultas: tipos de entrada 278
V.4.3.3. Selección y ejecución de servicios 280
V.4.3.4. Presentación de resultados 281
V.4.3.5. Servicio Web 281
V.5. Resumen 282
CAPÍTULO VI. APLICACIÓN DEL PROTOTIPO 285

VI.1. Introducción 285
VI.2. Aplicación en Gobierno Electrónico (eGovernment) 286
VI.2.1. Descripción del Problema y Objetivos 286
VI.2.1.1. Introducción al eGovernment 286
VI.2.1.2. Life Events – aproximación semántica 288
VI.2.1.3. Escenario de ejemplo 290
VI.2.2. Adaptación del Prototipo a los Requisitos del Dominio 292
VI.2.2.1. Ontología del dominio 293
VI.2.2.2. Servicios Web y anotación semántica 293
VI.2.2.3. Ejemplo de aplicación 297
VI.3. Aplicación en Bioinformática 302
VI.3.1. Descripción del Problema y Objetivos 303
VI.3.1.1. Introducción a la Bioinformática 303
VI.3.1.2. Fuentes de Información Biológica 305
VI.3.1.3. Escenario de ejemplo 308
VI.3.2. Adaptación del Prototipo a los Requisitos del Dominio 309
VI.3.2.1. Ontología del dominio 310
VI.3.2.2. Servicios Web y anotación semántica 313
VI.3.2.3. Ejemplo de aplicación 316
VI.4. Aplicación en Comercio Electrónico (eCommerce) 319
VI.4.1. Descripción del Problema y Objetivos 320
VI.4.1.1. Introducción al Comercio Electrónico 320
VI.4.1.2. Agentes en Comercio Electrónico (B2C y B2B) 324
VI.4.1.3. Escenario de ejemplo 328
VI.4.2. Adaptación del Prototipo a los Requisitos del Dominio 329
VI.4.2.1. Ontología del dominio 330
VI.4.2.2. Servicios Web y anotación semántica 333
VI.4.2.3. Ejemplo de aplicación 336
VI.5. Resumen 341
CAPÍTULO VII. CONCLUSIONES Y TRABAJO FUTURO 345
VII.1. Conclusiones 345

VII.2. Líneas Futuras 352
VII.2.1. Grid Computing 352
VII.2.2. Integración de datos 353
VII.2.3. Composición de servicios 354
VII.2.4. Mejora del prototipo 355
CAPÍTULO VIII. RESUMEN EN INGLÉS / SUMMARY IN ENGLISH 359
REFERENCIAS 391

ÍNDICE DE FIGURAS

Fig. 1. Arquitectura en capas de la Web Semántica 15
Fig. 2. Grafo RDF 16
Fig. 3. Versiones del lenguaje OWL 20
Fig. 4. Variantes de WSML (WSML, 2005) 22
Fig. 5. Formalismos lógicos en las variantes de WSML (WSML, 2005) 23
Fig. 6. Modelo de Referencia de FIPA (FIPA, 2004) 38
Fig. 7. Plataformas y contenedores en JADE (Caire, 2003) 43
Fig. 8. Página HTML en la Web actual 52
Fig. 9. Página Web con metadatos 53
Fig. 10. Evolución de la estructura de la Web tradicional a la estructura de la Web Semántica (W3C Oficina Española, 2006) 53
Fig. 11. Estructura en forma de pastel de la Web Semántica para el W3C 55
Fig. 12. Modelo de funcionamiento de RPC 59
Fig. 13. Arquitectura en capas de RMI 60
Fig. 14. Arquitectura de los Servicios Web 61
Fig. 15. Modelo de componentes de WSDL 2.0 (Booth & Liu, 2007) 63
Fig. 16. Fichero WSDL del Servicio Web de un hotel (Booth & Liu, 2007) 65
Fig. 17. Estructuras de datos centrales de UDDI (Clement et al., 2004) 66
Fig. 18. Estructura de un mensaje SOAP (adaptado de Miltra & Lafon, 2006) 68
Fig. 19. Mensaje SOAP para la reserva de un viaje (Miltra & Lafon, 2006) 69
Fig. 20. Evolución de la Web (adaptado de Fensel & Bussler, 2002) 73
Fig. 21. Ontología de alto nivel de OWL-S (OWL-S, 2004) 75
Fig. 22. Algunos de los conceptos y relaciones asociados al perfil del servicio (OWL-S, 2004) 76
Fig. 23. Ontología del proceso de alto nivel (OWL-S, 2004) 77
Fig. 24. Correspondencia entre OWL-S y WSDL (OWL-S, 2004) 78
Fig. 25. Elementos de alto nivel de WSMO (WSMO, 2005) 79
Fig. 26. Diagrama UML de los conceptos de nivel superior de WSMO (WSMO, 2005) 79

Fig. 27. Elementos relacionados con el concepto Ontología en WSMO (WSMO, 2005)..... 80
Fig. 28. Elementos relacionados con los conceptos Servicio Web y Objetivo de WSMO (WSMO, 2005)..... 81
Fig. 29. Mediadores que se tienen en cuenta en WSMO (WSMO, 2005)..... 82
Fig. 30. Anotación semántica de los elementos de WSDL (WSDL-S, 2005)..... 86
Fig. 31. Arquitectura de la OWL-S VM (Paolucci et al., 2003a)..... 91
Fig. 32. Arquitectura de WSMX (WSMX, 2005) 92
Fig. 33. Framework IRS 95
Fig. 34. Arquitectura del framework IRS-III (Cabral et al., 2006)..... 96
Fig. 35. Arquitectura de alto nivel de METEOR-S (Verma et al., 2005)..... 97
Fig. 36. "*Execution Environment*" (Verma et al., 2005) 98
Fig. 37. GODO entre los entornos de ejecución y los usuarios (Gómez et al., 2006) 107
Fig. 38. Arquitectura de GODO (Gómez et al., 2006) 107
Fig. 39. Arquitectura general de SWF (Stollberg et al., 2004a)..... 110
Fig. 40. Arquitectura del SMA para un flujo de trabajo de ejemplo (Buhler & Vidal, 2005)..... 114
Fig. 41. Arquitectura del entorno de trabajo Task Computing (Song et al., 2004) .. 116
Fig. 42. Task Computing en computación ubicua (Masuoka et al., 2003)..... 117
Fig. 43. Middleware que sirve de puente entre agentes y Servicios Web (Shafiq et al., 2005; Shafiq et al., 2006) 119
Fig. 44. Detalle de la estructura interna de "AgentWeb" (Shafiq et al., 2005; Shafiq et al., 2006)..... 120
Fig. 45. Interacción Agente-Servicio Web a distinto nivel 122
Fig. 46. Agentes compuestos por "*stub*" y "*body*" (Walton, 2005)..... 123
Fig. 47. Estructura de un "Servicio Web Semántico autónomo" (Paolucci & Sycara, 2003)..... 125
Fig. 48. Entorno de trabajo independiente de la aplicación y del dominio 138
Fig. 49. Caso de uso (nivel 0)..... 139
Fig. 50. Caso de uso (nivel 1) – publicación de servicios 140

Fig. 51. Caso de uso (nivel 1) – modificación de repositorios 140
Fig. 52. Caso de uso (nivel 1) – descubrimiento, selección y composición de servicios 141
Fig. 53. Caso de uso (nivel 1) – invocación de servicios 141
Fig. 54. Caso de uso (nivel 1) – monitorización 142
Fig. 55. Caso de uso (nivel 1) – mediación 142
Fig. 56. Caso de uso (nivel 1) – monitor del sistema 143
Fig. 57. Caso de uso (nivel 1) – gestión integral de la plataforma 143
Fig. 58. Diagrama de Organización – entorno 144
Fig. 59. Diagrama de organización – facilitador 145
Fig. 60. Modelo de agentes – agentes consumidor y proveedor 146
Fig. 61. Modelo de agentes – agentes descubridor y selector 146
Fig. 62. Modelo de agentes – agentes de servicio y mediador 147
Fig. 63. Modelo de agentes – agente de gestión y monitorización de la plataforma 147
Fig. 64. Modelo de Tareas y Objetivos (nivel 0) 148
Fig. 65. Modelo de Tareas y Objetivos (nivel 1) – tareas de gestión 149
Fig. 66. Modelo de Tareas y Objetivos (nivel 1) – provisión de servicios 149
Fig. 67. Gestión de servicios – Caso de Uso & Interacción 150
Fig. 68. Gestión de proveedores – Caso de Uso & Interacción 150
Fig. 69. Gestión de consumidores – Caso de Uso & Interacción 151
Fig. 70. Modelo de Interacción – Establecer necesidades 151
Fig. 71. Modelo de Colaboración – Establecer necesidades 152
Fig. 72. Modelo de Entorno – recursos externos a la plataforma 153
Fig. 73. Modelo de Entorno – recursos internos a la plataforma 153
Fig. 74. Modelo de Entorno – bases de conocimiento 154
Fig. 75. Arquitectura general del sistema 155
Fig. 76. Interacciones de los agentes 163
Fig. 77. Jerarquía de los roles que juegan los agentes 164
Fig. 78. Ontologías como núcleo de la arquitectura 168
Fig. 79. Infraestructura multi-capa. 170
Fig. 80. Niveles de descubrimiento (adaptado de Stollberg & Haller, 2005) 173

Fig. 81. Correspondencias posibles entre objetivos y Servicios Web (adaptado de Stollberg & Haller, 2005) 174
Fig. 82. Modelo ontológico de los objetivos 175
Fig. 83. Algoritmo para el descubrimiento y la composición 176
Fig. 84. Heterogeneidad de datos en la Web Semántica 184
Fig. 85. Mecanismo para la mediación de datos..... 185
Fig. 86. Modelo conceptual de la mediación de protocolos (Williams et al., 2005) 186
Fig. 87. Comunicación a nivel de protocolos en Sistemas Multi-Agente 186
Fig. 88. Problema que requiere mediación de procesos 187
Fig. 89. Protocolo de interacción *Contract Net* 189
Fig. 90. Mediación de procesos en el entorno de trabajo 190
Fig. 91. Envoltura de Servicio Web que permite a sistemas informáticos actuar como consumidores de servicios 199
Fig. 92. Escenario para la tarea de ejemplo 1 204
Fig. 93. Ontología del objetivo para tarea de ejemplo 1..... 207
Fig. 94. Diagrama de colaboración para la tarea de ejemplo 1 – fases de descubrimiento y selección..... 209
Fig. 95. Diagrama de colaboración para la tarea de ejemplo 1 – fase de invocación210
Fig. 96. Escenario para la tarea de ejemplo 2 213
Fig. 97. Ontología del objetivo para tarea de ejemplo 2..... 215
Fig. 98. Lista de subobjetivos para tarea de ejemplo 2 217
Fig. 99. Diagrama de colaboración para la tarea de ejemplo 2 – fase de composición 217
Fig. 100. Diagrama de colaboración para la tarea de ejemplo 2 – fases de invocación e integración 219
Fig. 101. Escenario para la tarea de ejemplo 3 221
Fig. 102. Ontología del objetivo para tarea de ejemplo 3..... 224
Fig. 103. Diagrama de colaboración para la tarea de ejemplo 3 – fases de descubrimiento y selección..... 224
Fig. 104. Diagrama de colaboración para la tarea de ejemplo 3 – fases de invocación e integración 225

Fig. 105. Mecanismo de interacción SMA-Aplicación Web (Kelemen, 2006) 235
Fig. 106. SEMMAS – Aplicación Web para ejecución de Servicios Web Semánticos 242
Fig. 107. Representación gráfica del modelo ontológico de objetivo 248
Fig. 108. Diagrama de secuencia para las tareas de descubrimiento y composición 249
Fig. 109. Descomposición de objetivos y descubrimiento de servicios 250
Fig. 110. Consulta SPARQL para obtener los elementos '*Output*' de un objetivo .. 251
Fig. 111. Consulta SPARQL de ejemplo para obtener los elementos que se producen como salida de un servicio 251
Fig. 112. Esquema DTD de los modelos de preferencia de los consumidores..... 253
Fig. 113. Esquema DTD de los modelos de preferencia de los proveedores 254
Fig. 114. Lista de servicios asociadas a los (sub)objetivos 254
Fig. 115. Cálculo del valor de utilidad 255
Fig. 116. 1ª Fase Selección: creación de los agentes implicados 256
Fig. 117. 2ª Fase Selección: negociación y evaluación de propuestas 257
Fig. 118. Consulta SPARQL de ejemplo para obtener los parámetros de entrada de un servicio 258
Fig. 119. Consulta SPARQL para obtener los nombres y valores de los elementos '*Input*' de un objetivo 258
Fig. 120. Diagrama de secuencia para la tarea de invocación 259
Fig. 121. Estructura de la aplicación Web conforme al patrón Struts 261
Fig. 122. Implementación del patrón DAO 263
Fig. 123. Interfaz desarrollador – gestión de repositorios 265
Fig. 124. Interfaz desarrollador – gestión de ontologías 266
Fig. 125. Interfaz desarrollador – gestión de roles 267
Fig. 126. Interfaz desarrollador – gestión de agentes 269
Fig. 127. Interfaz proveedor – registro en el sistema 271
Fig. 128. Interfaz proveedor – acceso al sistema..... 271
Fig. 129. Interfaz proveedor – modificación datos de registro..... 272
Fig. 130. Interfaz proveedor – página principal & listado de servicios aportados ... 273
Fig. 131. Interfaz proveedor – añadir nuevo servicio al sistema 274

Fig. 132. Interfaz proveedor – modificación datos del servicio 275
Fig. 133. Interfaz consumidor – registro en el sistema..... 277
Fig. 134. Interfaz consumidor – acceso al sistema 277
Fig. 135. Interfaz consumidor – modificación datos de registro 278
Fig. 136. Interfaz consumidor – realizar consulta 280
Fig. 137. Interfaz consumidor – listado de servicios por subobjetivos para su selección 281
Fig. 138. Interfaz consumidor – presentación de resultados 281
Fig. 139. Escenario de ejemplo para eGovernment..... 291
Fig. 140. Extracto del fichero WSDL del servicio ALOWS 295
Fig. 141. Extracto de la descripción semántica del servicio AOLWS 296
Fig. 142. eGovernment – ontología del objetivo resultante 299
Fig. 143. Plan para la descomposición del objetivo 300
Fig. 144. eGovernment – servicios encontrados tras la descomposición 301
Fig. 145. eGovernment – selección de servicios 301
Fig. 146. eGovernment – resultado de la consulta 302
Fig. 147. Escenario de ejemplo para Bioinformática 309
Fig. 148. Jerarquía de alto nivel de la ontología '*Oncogene*' 312
Fig. 149. El concepto Cáncer en la ontología '*Oncogene*' 312
Fig. 150. Fichero WSDL del servicio GOWS 314
Fig. 151. Extracto de la descripción semántica del servicio GOWS 315
Fig. 152. Bioinformática – ontología del objetivo resultante 317
Fig. 153. Bioinformática – selección de servicios..... 318
Fig. 154. Bioinformática – resultado de la consulta 319
Fig. 155. Flujo de información y material en un entorno integrado de B2C (The USHER Project, 2002a)..... 322
Fig. 156. Flujo de información y material en un entorno integrado de B2B (The USHER Project, 2002b) 323
Fig. 157. Modelo de comportamiento del consumidor al comprar (He et al., 2003) 326
Fig. 158. Modelo del ciclo de vida Business-to-Business (He et al., 2003)..... 327
Fig. 159. Escenario de ejemplo para Comercio Electrónico 328

Fig. 160. Conceptos de primer nivel de la ontología de componentes informáticos 331
Fig. 161. Jerarquía de componentes para el almacenamiento de información 331
Fig. 162. Relaciones mereológicas: uniendo las partes al todo 332
Fig. 163. Extracto de fichero WSDL del servicio AWS 334
Fig. 164. Definición de los mensajes de entrada y salida de la operación *compraComp_A* 335
Fig. 165. Extracto de la descripción semántica del servicio AWS 336
Fig. 166. Comercio Electrónico – ontología del objetivo resultante 337
Fig. 167. Comercio Electrónico – selección de servicios 340
Fig. 168. Comercio Electrónico – resultado de la consulta 341

ÍNDICE DE TABLAS

Tabla 1. Capacidades de razonamiento con RDF/RDFS 18

Tabla 2. Capacidades de razonamiento con OWL 21

Tabla 3. Actividades a realizar en las etapas de inicio, elaboración y construcción (Pavón y Gómez-Sanz, 2003) 34

Tabla 4. Descripción de los elementos obligatorios de la Arquitectura Abstracta de FIPA (FIPA, 2002a) 41

Tabla 5. Objetivos que persigue cada tipo de agente 159

Tabla 6. Entradas, salidas e interacciones de los agentes 162

Tabla 7. Descripción de los roles jugados por agentes 166

Tabla 8. Asociación inicial entre agentes y los roles que estos asumen 167

Tabla 9. Listado de propiedades no funcionales (O'Sullivan et al., 2005; Toma & Foxvog, 2006) 177

Tabla 10. Asociación en tiempo de diseño entre Comportamientos y Roles 243

Tabla 11. Relación de comportamientos que muestran los agentes, necesarios y permitidos 244

Tabla 12. API de los roles en el sistema 245

Tabla 13. Servicios considerados relevantes en el dominio 289

INTRODUCCIÓN

Interconectar varios sistemas software para ofrecer servicios o funcionalidades de un modo remoto se ha planteado como objetivo desde antes de la aparición de Internet. Con el nacimiento de la World Wide Web (WWW) en 1989 se dio un paso más hacia la consecución de este objetivo. En sus orígenes, la Web fue concebida como un medio donde compartir información. En la actualidad, sin embargo, gracias a los *Servicios Web* (herederos de otras soluciones para computación distribuida como CORBA, RMI o DCOM) la Web ha pasado de ser un mero repositorio de información a una fuente de servicios accesibles desde cualquier punto del planeta.

El incremento exponencial de la cantidad de datos publicados en la Web y el consecuente aumento de la complejidad y el tiempo necesario por parte de usuarios humanos para encontrar la información que precisan en un determinado momento es lo que dio lugar a lo que hoy se conoce como *Web Semántica*. En el mundo de los Servicios Web, y de forma similar a lo ocurrido en la Web, la cantidad cada vez mayor de servicios disponibles hace inviable en tiempo y eficiencia que sea un usuario humano el que determine el servicio o servicios necesarios para satisfacer una necesidad concreta, surgiendo de este modo los *Servicios Web Semánticos*. En ambos casos, el problema es el mismo: la inexistencia de información procesable automáticamente por máquinas. Y, en ambos casos, la solución es equivalente, a saber, incluir información adicional expresada formalmente y que permita a los sistemas informáticos "entender", en cierto modo, el contenido de los documentos. Con este propósito se hace uso de la *ingeniería ontológica*, disciplina en la que *ontología* se entiende como una especificación explícita y formal de una conceptualización compartida (Gruber, 1993; Borst, 1997; Studer et al., 1998).

Una vez la información acerca de las funcionalidades ofrecidas por los servicios ha sido descrita por medio de ontologías, los sistemas informáticos quedan habilitados para acceder a esta información de forma automática sin intervención humana. Es en este punto donde aparecen en escena los *Agentes Inteligentes*, los cuales pueden definirse

como entidades software encargadas de acceder a la información de los servicios y ejecutarlos en representación de usuarios humanos.

Diversas y contradictorias son las propuestas establecidas hasta el momento acerca de cómo ha de tener lugar esta interacción entre tecnologías de agentes y de Servicios Web Semánticos. En la mayor parte de los casos, estas propuestas implican la modificación de los estándares pre-establecidos en alguna de las dos tecnologías o la limitación de las propiedades que éstas ofrecen. Esta es la razón primordial que ha motivado la realización de la investigación descrita en esta tesis doctoral. La solución propuesta en este trabajo se basa en el desarrollo de un marco de trabajo donde agentes y servicios interactúen de forma cooperativa aprovechando al máximo las posibilidades que estas tecnologías ofrecen y sin necesitar para esto ningún cambio sustancial dentro de las propuestas y estándares desarrollados hasta el momento en el ámbito de las mismas. El componente central y estratégico de la propuesta son las ontologías, por medio de las cuales los agentes se comunican y pueden entablar conversaciones al tiempo que hacen uso de los servicios disponibles sin precisar intervención humana. Este trabajo aporta por tanto un nuevo modelo para relacionar Agentes Inteligentes y Servicios Web Semánticos de forma no intrusiva.

Para lograr este objetivo se ha seguido la siguiente metodología:

1. Análisis del estado del arte en Ingeniería Ontológica, Agentes Inteligentes y Servicios Web Semánticos. Dada su relación con alguna de estas tecnologías, también se realizó un estudio no exhaustivo del estado del arte acerca de la Web Semántica, los formalismos lógicos, el razonamiento lógico y los motores de inferencia.
2. Análisis de las metodologías actuales para diseñar y construir *Sistemas Multi-Agente*.
3. Análisis de las propuestas actuales para la interacción de Agentes Inteligentes y Servicios Web Semánticos.
4. Definición y formalización de un entorno multi-agente basado en tecnologías del conocimiento para el acceso dinámico y ejecución automática de Servicios Web Semánticos. Este entorno se ha concebido para que sea independiente del

dominio (el conocimiento del dominio se representa por medio de ontologías) y de la aplicación concreta que se desee desarrollar sobre este dominio.

5. Diseño e implementación de una aplicación software para la ejecución dinámica de servicios a partir de objetivos abstractos establecidos por usuarios (humanos). Utilizando dicha aplicación en el marco de trabajo adecuado, se pretende que cualquier desarrollador pueda construir diversas aplicaciones sobre dominios concretos.
6. Testing y validación de la aplicación para comprobar que satisface los requisitos establecidos en la etapa de definición y formalización. Para ello, se desarrollan los componentes necesarios para la aplicación del software a desarrollar en tres dominios diferentes y se evalúan los resultados obtenidos.

Las cuestiones planteadas en la metodología anterior se han abordado con éxito, y los resultados obtenidos se presentan en esta memoria. La organización de la misma es como sigue.

En el capítulo I se ofrece un detallado estado del arte en aquellas tecnologías involucradas en la investigación, esto es, Ontologías, Agentes Inteligentes y Servicios Web Semánticos. En primer lugar, se define el concepto de "Ontología", se indican los distintos tipos de ontologías existentes, los lenguajes ontológicos y los formalismos lógicos en que éstos están basados, además de ofrecer una breve descripción de los sistemas de inferencia desarrollados hasta la fecha. A continuación, se enumeran diversos aspectos relacionados con el mundo de los agentes, se revisan las metodologías para el diseño de Sistemas Multi-Agente y las plataformas que permiten el desarrollo de este tipo de sistemas de acuerdo con los estándares establecidos. Finalmente, se realiza un recorrido histórico que lleva al surgimiento de los Servicios Web Semánticos como unión lógica de los Servicios Web y la Web Semántica. En este capítulo, además, se indican los problemas y carencias que padecen estas tecnologías cuando se aplican por separado.

En el capítulo II se muestran diversas propuestas desarrolladas hasta el momento para conseguir una integración efectiva de las tecnologías de Agentes (Inteligentes) y de Servicios Web (Semánticos). Asimismo, se destacan los inconvenientes de cada una de

estas herramientas. Esto enlaza con la definición de los objetivos y requisitos a satisfacer por el sistema, elementos que se detallan en el capítulo III. En este capítulo, además de definir los objetivos de diseño, se presenta la arquitectura del marco de trabajo donde entran en juego los Agentes Inteligentes, los Servicios Web Semánticos y, para habilitar la interoperabilidad entre ellos, las Ontologías. Finalmente, se indica de qué forma los usuarios externos a la aplicación (ya sean desarrolladores, proveedores de servicios o consumidores de servicios) pueden hacer uso del sistema.

En el capítulo IV se muestran varios ejemplos que describen el funcionamiento del marco de trabajo descrito en el capítulo III y se compara con lo que hubiera sido necesario hacer y los resultados que se hubieran obtenido en caso de haber utilizado agentes y Servicios Web por separado.

La aplicación software desarrollada, que sigue las pautas marcadas por la arquitectura presentada en el capítulo III, se describe en el capítulo V. En este capítulo, se detalla el proceso de diseño de la aplicación tal y como viene especificado en la metodología para el desarrollo de Sistemas Multi-Agente utilizada. Para terminar el capítulo V, se muestran las interfaces a través de las cuales los tres tipos de usuarios mencionados anteriormente pueden acceder al sistema. Los resultados de la validación de este prototipo en tres dominios diferentes, como son el *Gobierno Electrónico* (e-Gobierno), la *Bioinformática* y el *Comercio Electrónico* (e-Comercio) se muestran en el capítulo VI.

Para finalizar, en el capítulo VII se presentan las conclusiones finales, así como líneas de trabajo futuro, para mejorar los resultados obtenidos a través de esta tesis, su aplicación en otros dominios y la integración de otras tecnologías, como el GRID, en la plataforma.

CAPÍTULO I. ONTOLOGÍAS, SERVICIOS WEB SEMÁNTICOS Y AGENTES INTELIGENTES – ESTADO DEL ARTE

I.1. Introducción

En este capítulo se describe el estado actual de las tres tecnologías que forman el núcleo central de esta tesis doctoral, a saber, Ingeniería Ontológica, Agentes Inteligentes y Servicios Web Semánticos.

En primer lugar, se trata el concepto de "ontología" a través de algunas de las definiciones más aceptadas del término en el ámbito informático y, más concretamente, de la Inteligencia Artificial y la Representación del Conocimiento. Seguidamente se enumeran los distintos tipos de ontologías, clasificándolas tanto por el conocimiento que contienen como por la motivación de las mismas. Después se describen brevemente los principales formalismos lógicos que constituyen la base de los lenguajes ontológicos actuales y, para finalizar, se introducen algunos de los lenguajes ontológicos definidos hasta la fecha poniendo especial énfasis en el lenguaje OWL, que se ha erigido como un estándar para la comunidad.

En segundo lugar, se introducen diversos elementos relacionados con el mundo de los Agentes Inteligentes, indicando algunas de las propiedades que debe satisfacer un agente para poder denominarse inteligente. Además, se justifica la necesidad de disponer de diversos agentes actuando sobre un mismo entorno para llevar a cabo tareas ya sea de un modo cooperativo o competitivo, lo que enlaza de forma natural con la idea de Sistema Multi-Agente. Se señalan posteriormente algunas pistas que nos pueden conducir a la elaboración de mecanismos de negociación eficaces en entornos multi-agente colaborativos. Para terminar esta sección, se mencionan diversas metodologías para el desarrollo de Sistemas Multi-Agente y plataformas para su implementación, haciendo especial hincapié en la metodología INGENIAS y la plataforma JADE, que han sido las utilizadas en este trabajo.

En tercer lugar, se presentan los principios básicos de los Servicios Web Semánticos. Se parte de la definición de Web Semántica y Servicios Web para concluir lógicamente con el surgimiento de la tecnología de los Servicios Web Semánticos. De esta manera,

se presentan las principales aproximaciones (dado que no existe a día de hoy un estándar único) y varios entornos de ejecución que se encuentran ligados a uno u otro estándar. Además, a lo largo del capítulo se indican los principales inconvenientes asociados al uso de las tecnologías descritas cuando se aplican por separado, lo cual conduce a la necesidad de desarrollar sistemas que superen estas adversidades a partir de la integración de las tecnologías mencionadas.

I.2. Ontologías

I.2.1. Definición

En el diccionario de la Real Academia de la Lengua Española se define el concepto '*ontología*' como la "parte de la metafísica que trata del ser en general y de sus propiedades transcendentales". 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 Inteligencia Artificial (en particular en Ingeniería del Conocimiento) 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 Willard Van Orman Quine (Quine, 1961) quien 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 en informática con más aceptación sobre las que se basa este trabajo de investigación.

Una de las definiciones más citadas es la enunciada por Gruber (1993):

"Una ontología es una especificación explícita 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 (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 sus colegas (1998) se encargaron de fusionar las definiciones de Gruber y Borst, estableciendo la definición que ha sido adoptada en esta investigación:

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

Este enunciado fue, a su vez, examinado en profundidad en este mismo trabajo 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 un 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 Tom Gruber, el cual responde en un boletín publicado en Octubre de 2004 (Gruber, 2004) lo siguiente, cuando le preguntan si cambiaría de alguna forma la definición que dio de ontología y que ha sido muchas veces referenciada:

> *"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. Retrospectivamente, 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 mas 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 (Gruber, 1993):

- 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 (Studer et al., 1998). 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: $R: C_1 \times C_2 \times \ldots \times C_n$. Entre los distintos tipos de relaciones

posibles, se encuentran las relaciones taxonómicas ("*es_un*") y las mereológicas o partonómicas ("*parte_de*") como relaciones binarias más destacadas.

- 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: $F: C_1 \times C_2 \times \ldots \times C_{n-1} \rightarrow C_n$. 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, las *instancias* son las ocurrencias en el mundo real de los conceptos. En una instancia todos los atributos del concepto tienen asignado un valor concreto.

I.2.2. Tipos de Ontologías

Existen diferentes clasificaciones de tipos de ontologías. Se siguen principalmente dos criterios para estas clasificaciones: (a) el tipo de conocimiento que contienen; y (b) la motivación de la ontología. A continuación, se enumeran los distintos tipos de ontologías para cada una de estas posibles clasificaciones.

I.2.2.1. Clasificación por el conocimiento que contienen

Gertjan van Heijst y sus colegas propusieron los siguientes tres tipos de ontologías atendiendo al conocimiento que contienen (van Heijst et al., 1997):

- 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 tienen una estructura interna mucho más rica que los anteriores y son las que interesan a los desarrolladores de sistemas basados en conocimiento.

I.2.2.2. Clasificación por motivación

La segunda clasificación se hace dependiendo del motivo por el que se elabora de la ontología. Tradicionalmente se distinguen cuatro tipos principales de acuerdo con su motivación (Steve et al., 1997):

- 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* (*meta-level* o *top-level ontologies*).
- 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, la cardiología, etc.
- Ontologías de aplicación (Guarino, 1998): 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 reutilizadas.

I.2.3. Formalismos lógicos y razonamiento con ontologías

Como ya se comentó previamente, para que los programas software sean capaces de realizar procesos de inferencia sobre la información almacenada en las ontologías, éstas deben de fundamentarse en algún formalismo lógico. La utilización de estos formalismos lógicos, además de proporcionar la base necesaria para el uso automatizado de la información, ofrece la posibilidad de realizar procesos de razonamiento e inferencia que conducen a la adquisición de nuevo conocimiento a partir del ya existente. Así, se puede decir que una lógica define un lenguaje formal para expresar conocimiento acerca de algún dominio y las proposiciones a ser derivadas de este conocimiento, es decir, fija el conjunto de primitivas que pueden emplearse para modelar el mundo, esto es, la *sintaxis*. Asimismo, una lógica establece la *semántica* formal de cada proposición en el lenguaje correspondiente (valor de verdad de cada

sentencia respecto a cada mundo posible) y está equipada con un '*cálculo*', es decir, un procedimiento formal (computable) para derivar nuevos hechos (descritos en el lenguaje correspondiente) a partir de un conjunto de proposiciones dado en el lenguaje.

Dada su influencia en los lenguajes ontológicos de la Web Semántica que se estudiarán en la siguiente sección, describiremos brevemente a continuación los tres formalismos lógicos más relevantes en este ámbito: *lógica de primer orden* o *lógica de predicados*, *lógica descriptiva* y *programación lógica*.

I.2.3.1. Lógica de primer orden

La lógica de primer orden (LPO) es una extensión de la lógica proposicional que fue ideada con el propósito de ser un lenguaje capaz de representar la mayor parte de las situaciones que pueden suceder en el mundo real. En LPO, se pueden representar entidades individuales (*objetos*), entre las que puede haber *relaciones* (las relaciones unarias se denominan *propiedades* y representan características distintivas de los objetos). Las relaciones en las que, dada una "entrada", se obtiene un solo "valor", se denominan *funciones*.

Entre las ventajas de la LPO se encuentran las siguientes:

- Estructura el mundo en objetos y relaciones, lo que facilita el razonamiento.
- Ofrece libertad para describir el mundo de la manera que el diseñador considere apropiada.
- Permite expresar sentencias sobre todos los objetos del universo.

La gran capacidad expresiva de la LPO implica, desgraciadamente, que algunas tareas de razonamiento en este lenguaje son, en general, indecidibles, esto es, no se puede obtener una respuesta en un tiempo finito (de Bruijn, 2006). Es por esta razón por la que se han estudiado numerosos subconjuntos de la LPO obteniéndose lenguajes como la lógica descriptiva y la programación lógica.

I.2.3.2. Lógica descriptiva

La lógica descriptiva es un formalismo para representar conocimiento. Más concretamente, la lógica descriptiva es un conjunto de lenguajes de representación de conocimiento que pueden ser usados para representar el conocimiento terminológico de

un dominio de aplicación de una forma estructurada y con una semántica formal bien definida (Baader et al., 2003).

El nombre de 'lógica descriptiva' se refiere, por un lado, a la descripción de los conceptos que se usan para definir el dominio y, por el otro lado, a la semántica basada en lógica (semántica formal bien definida) que viene dada por su traducción en predicados de LPO (esto distingue a la lógica descriptiva de sus predecesores: redes semánticas y marcos).

La lógica descriptiva es un subconjunto de lógica de predicados (LPO) y constituye la base de RDF(S) y de OWL. Este formalismo se ideó buscando un compromiso entre la capacidad de ser computable y capacidad expresiva. El lenguaje más simple basado en lógica descriptiva es el $\mathcal{ALC}$ (Attributive Language with Complement) que permite representar conceptos, jerarquías de conceptos, restricciones de roles y combinaciones booleanas de descripciones de conceptos. Lenguajes populares y más expresivos en lógica descriptiva, como $\mathcal{SHIQ}$ y $\mathcal{SHOIQ}$, son extensiones de $\mathcal{ALC}$.

La sintaxis de la lógica descriptiva consiste en:

- Conceptos atómicos: equivalentes a los predicados unarios en LPO.
- Roles atómicos: equivalentes a los predicados binarios en LPO.
- Operadores: establecen relaciones entre conceptos y restricciones sobre los mismos.

En general, un concepto denota un conjunto de individuos que pertenecen a una clase, y un rol denota una relación entre conceptos. Los lenguajes en lógica descriptiva se basan, generalmente, en constructores provenientes de LPO como *intersección*, *conjunción*, *unión*, *disyunción*, *negación*, *complemento*, *cuantificador* (restricción) *universal*, *cuantificador* (restricción) *existencial*, etc. Otros constructores pueden incluir restricciones en los roles (típicos de relaciones binarias) como la *transitividad*, *inverso*, *funcionalidad*, etc.

La lógica descriptiva se puede utilizar tanto para definir las bases de conocimiento como para hacer inferencias sobre ellas. En una base de conocimiento se pueden distinguir el conocimiento '*intensional*' (conocimiento general acerca del dominio de un problema) y el conocimiento '*extensional*' (conocimiento específico de un problema particular). Del mismo modo, en lógica descriptiva, una base de conocimiento se

compone de *TBox* y *ABox*. La TBox contiene conocimiento intensional en forma de una terminología y está construida por declaraciones que describen propiedades generales de los conceptos. En otras palabras, la TBox contiene un conjunto de axiomas que describen la estructura del dominio (esquema de clases). Por su parte, la ABox contiene conocimiento extensional (conocimiento de aserciones), o sea, conocimiento específico de los individuos del dominio de discurso representable por medio de un conjunto de axiomas que describen una situación concreta.

Entre las tareas de inferencia más comunes en los sistemas basados en lógica descriptiva, podemos encontrar la *subsunción* (probar formalmente que un concepto es más específico que otro concepto; p.ej. $C \subseteq D$.) y *satisfacibilidad* (determinar si una base de conocimiento es satisfacible, esto es, no contiene proposiciones contradictorias). El razonamiento en la mayor parte de los lenguajes de lógica descriptiva es decidible y, para algunos de éstos, existen algoritmos optimizados para razonamiento.

I.2.3.3. Programación lógica

La Programación Lógica, junto con la funcional, forma parte de lo que se conoce como *programación declarativa*. En los lenguajes tradicionales, la programación consiste en indicar cómo resolver un problema mediante sentencias. En la programación lógica, se trabaja de una forma descriptiva, estableciendo relaciones entre entidades e indicando no el cómo se deben de hacer las cosas, sino el qué hacer. La programación lógica se basa en un subconjunto de LPO denominada '*Lógica de Horn*' (Lloyd, 1987). Sin embargo, la semántica de la programación lógica es diferente de la LPO. En este caso, la semántica está basada en los modelos mínimos de Herbrand.

Una cláusula de Horn es una fórmula clausular en donde aparece, como mucho, un literal afirmado:

$$\neg A(x) \vee \neg D(x) \vee \neg E(x) \vee F(x, y)$$

Por analogía con la formulación implicativa y deductiva, la notación utilizada para este tipo de cláusulas es una lista de literales negados como antecedente y el literal afirmado como consecuente separados por el signo de implicación:

$$A(x), D(x), E(x) \rightarrow F(x, y)$$

De esta forma, un programa lógico consiste en reglas de la forma 'si A entonces B'.

En la Web Semántica, la programación lógica cumple dos funciones principales. Por un lado se utiliza para razonar con RDF, RDF Schema y partes de OWL y, por otro, es usada para representar conocimiento en forma de *reglas*. Las reglas se pueden ver como un paradigma de representación de conocimiento complementario a la lógica descriptiva. La lógica descriptiva es conveniente para definir clases, jerarquías, propiedades y las relaciones entre ellas y entre su poder expresivo se encuentra el siguiente: cuantificación existencial, disyunción y negación clásica. Por su parte, la programación lógica, posee el siguiente poder expresivo adicional: predicados con aridades arbitrarias, variables encadenadas sobre predicados (sin restricciones en el uso de variables) y el uso de negación no-monótona (de Bruijn, 2006). El lenguaje *Prolog* es el principal representante de este paradigma de programación.

I.2.4. Lenguajes para la representación de ontologías

Aunque las ontologías se han utilizado durante muchos años en diversos campos de investigación, se han convertido en la tecnología estándar *de facto* para la representación del conocimiento tras el surgimiento de la Web Semántica. Antes de este hecho, ya se habían ideado numerosos lenguajes basados en distintos formalismos lógicos para la representación de ontologías. Entre los primeros en aparecer, se encuentran KIF (‘*Knowledge Interchange Format*’), OCML (‘*Operational Conceptual Modelling Language*’) y F-Logic (‘*Frame Logic*’). Sin embargo, han sido aquellos lenguajes surgidos tras la aparición de la Web Semántica los que han conseguido mayor aceptación. Nos referimos, por ejemplo, a lenguajes como SHOE (‘*Simple HTML Ontology Extensions*’, 1999), DAML-ONT (‘*DARPA Agent Markup Language*’, 2000), OIL (‘*Ontology Inference Layer*’, 2000), DAML+OIL (2001) y, finalmente, el estándar en la actualidad, OWL (‘*Web Ontology Language*’, 2004).

La diferencia entre los lenguajes ontológicos aparecidos junto con la Web Semántica y los anteriores es que estos últimos no fueron definidos para que fueran compatibles con la arquitectura de la World Wide Web, en general, ni con la Web Semántica, en particular. Los lenguajes actuales siguen la estructura presentada por Tim Berners-Lee (2000, p.10) y que se muestra en la siguiente figura (Fig. 1):

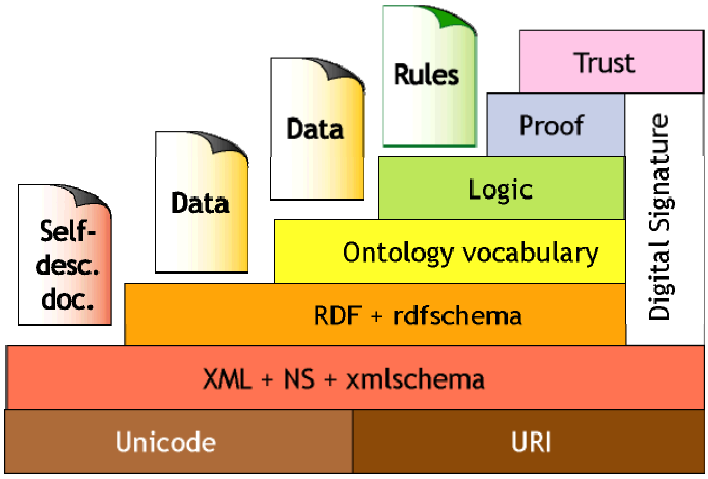


Fig. 1. Arquitectura en capas de la Web Semántica

Estos lenguajes se basan primeramente en la identificación de todos los elementos a través de una URI (*Uniform Resource Identifier*) y en la sintaxis proporcionada por XML (*eXtensible Markup Language*). Por encima de esto, apareció RDF (*Resource Description Framework*), un modelo de datos simple que permite identificar recursos y sus relaciones con otros recursos y RDFS (*RDF Schema*), que es un vocabulario para describir las propiedades y las clases de los recursos RDF con una semántica para establecer jerarquías de generalización entre dichas propiedades y clases.

Debido a su interés para el desarrollo de este trabajo de investigación, a continuación se indican, con mayor grado de detalle, algunas de las características de RDF, RDFS, OWL y WSML.

I.2.4.1. Resource Description Framework (RDF)

RDF es un modelo de datos para representar los recursos y las relaciones que se pueden establecer entre ellos. Además, el modelo de datos RDF se ha definido con una semántica básica que puede representarse mediante XML.

El modelo de datos de RDF está basado en '**tripletas**', cada una de las cuales está formada por (Klyne & Carroll, 2004):

- **Sujeto**: identifica al recurso (persona, lugar o cosa) que la sentencia describe. Un recurso RDF puede ser cualquier cosa en un modelo de datos (documento, usuario, producto, etc.). A cada recurso se le asigna un identificador único en forma de URI.

- **Predicado**: representa una propiedad (nombre, ciudad, título, color, forma, característica) del sujeto (persona, lugar o cosa). Al igual que los sujetos, cada propiedad viene identificada por medio de una URI única.
- **Objeto**: especifica el valor (Pedro, Madrid, "Braveheart", azul, circular, duro) para la propiedad (nombre, ciudad, título, color, forma, característica) del sujeto (persona, lugar o cosa).

La información almacenada por medio de tripletas RDF se puede representar a partir de grafos (Fig. 2).

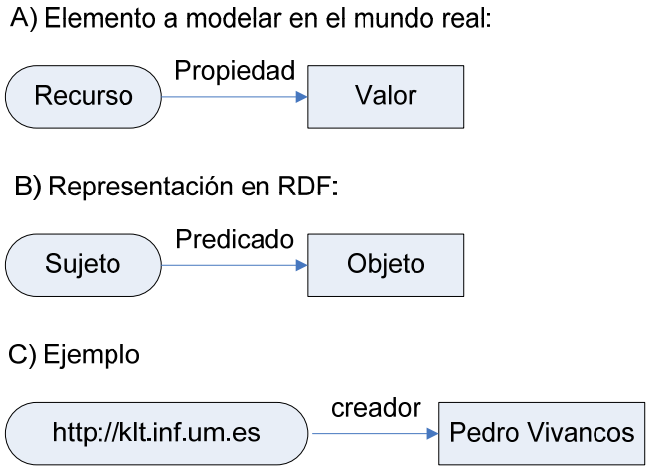


Fig. 2. Grafo RDF

Existen numerosas notaciones para representar grafos RDF. Así, por ejemplo, si representamos en XML el grafo de ejemplo obtendríamos lo siguiente:

```
<rdf:RDF xmlns:rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns#
xmlns:dc = "http://purl.oclc.org/DC">
<rdf:description rdf:href = "http://klt.dif.um.es">
<dc:creator>Pedro Vivancos</dc:creator>
</rdf:description>
</rdf:RDF>
```

Como se puede comprobar en el ejemplo, el problema de esta notación radica en la cantidad de texto necesario para expresar las sentencias más simples. Dado que uno de los requisitos para la elaboración de una notación para la representación de grafos RDF es que sea fácilmente entendible por humanos, se han diseñado nuevos mecanismos de representación mucho más apropiados como, por ejemplo, la notación N-Triple y la notación N3. A continuación, se muestra el mismo ejemplo representado en estas dos notaciones:

N-Triple

```
<http://klt.inf.um.es> <http://purl.oclc.org/DC#creator> "Pedro
Vivancos"
```

N3

```
# Base: http://klt.inf.um.es/
@prefix: <#> .
<http://klt.inf.um.es>
<http://purl.oclc.org/DC#creator>
"Pedro Vivancos"
```

En resumen, RDF aporta una forma de expresar enunciados simples acerca de recursos usando propiedades y valores.

I.2.4.2. RDF Schema (RDFS)

RDF es un lenguaje de propósito general para representar información en la Web. La especificación RDFS describe cómo usar RDF para describir vocabularios RDF. En particular, el propósito de RDFS es proveer de un vocabulario XML a través del cual expresar clases y sus relaciones (permite definir jerarquías de clases), al tiempo que se definen propiedades y se asocian las mismas con clases (Brickley & McBride, 2004). De este modo, RDFS facilita mejoras en los procesos de búsqueda y permite hacer inferencias.

El sistema de clases y propiedades de RDFS es similar al sistema de tipos de algunos lenguajes de programación orientados a objetos, con la diferencia de que, en vez de definir clases en términos de las propiedades que las instancias de estas clases deben de tener, RDFS describe propiedades en términos de las clases de recursos a los que estas se pueden aplicar. Para este propósito, aparecen los roles de '*dominio*' y '*rango*'. Así, por ejemplo, se podría definir la propiedad `eg:autor` con un dominio `eg:Documento` y un rango `eg:Persona`. De este modo, es posible definir más propiedades que tengan como dominio `eg:Documento` o como rango `eg:Persona` sin necesidad de modificar la descripción original de estas clases.

Por tanto, se puede reconocer RDFS como un lenguaje ontológico primitivo que ofrece ciertas primitivas para el modelado de clases, relaciones de subclases, propiedades, relaciones de subpropiedades y restricciones de dominio y rango, con un significado fijo (Antoniou & van Harmelen, 2004). Desgraciadamente, RDFS es demasiado primitivo como lenguaje de modelado para la Web y carece de muchas primitivas de modelado interesantes en relación a otros lenguajes más recientes. Entre sus limitaciones, cabe destacar las siguientes (Antoniou & van Harmelen, 2004):

- Ámbito local de las propiedades: no permite que las restricciones se apliquen sólo a algunas clases (p.ej. '*Las vacas sólo comen hierba*').
- • Clases disjuntas (p.ej. clase *Hombre* vs. clase *Mujer*).
- Combinaciones booleanas de clases: definir clases mediante unión, intersección, complementario, etc. (p.ej. *Persona* = *Hombre* U *Mujer*).
- Restricciones de cardinalidad (p.ej. una persona sólo tiene 2 progenitores).
- Características de propiedades: transitividad, unicidad, inversa, etc.

Sin embargo, a pesar de las limitaciones presentadas, ya es posible con el uso de RDF y RDFS realizar diversas tareas de razonamiento automáticas para inferir nuevas relaciones a partir de una base de conocimiento dada. En la Tabla 1 se muestran las capacidades de razonamiento de RDF/RDFS:

Herencia de tipos
`rdfs:subClassOf`
	`(rdf:type Perla Gato)`
	`(rdfs:subClassOf Gato Mamífero)`
	`→ (rdf:type Perla Mamífero)`

Reflexividad
`rdfs:subPropertyOf` y `rdfs:subClassOf`
	`(rdf:Property p) →`
	`(rdfs:subPropertyOf p p)`
	`(rdf:Class C) →`
	`(rdfs:sublClassOf C C)`

Inferencia de tipos
`Rdfs:range` y `rdfs:domain`
	`(rdfs:domain enseña Profesor)`
	`(rdfs:range enseña Estudiante)`
	`(enseña Rafa Jose)`
	`→`
	`(rdf:type Rafa Profesor)`
	`(rdf:type Jose Estudiante)`

Transitividad
`rdfs:subClassOf`
	`(rdfs:subClassOf Gato Mamifero)`
	`(rdfs:subClassOf Mamifero Animal)`
	`→`
	`(rdfs:subClassOf Gato Animal)`

Transitividad
`rdfs:subPropertyOf`
	`(rdfs:subPropertyOf padre progenitor)`
	`(rdfs:subPropertyOf progenitor familiar)`
	`→`
	`(rdfs:subPropertyOf padre familiar)`

Tabla 1. Capacidades de razonamiento con RDF/RDFS

I.2.4.3. Web Ontology Language (OWL)

Debido a las limitaciones de RDF/RDFS, principalmente la carencia del suficiente poder expresivo, se comenzaron a definir tanto en Estados Unidos como en Europa nuevos lenguajes ontológicos con mayor capacidad expresiva. Así surgieron SHOE, DAML-ONT, OIL, y DAML+OIL, que sería el precursor de OWL, la actual recomendación del W3C (*World Wide Web Consortium*) (OWL, 2004). A continuación,

se enumeran algunas de las posibilidades adicionales que proporciona el vocabulario de OWL sobre el de sus predecesores (McGuinness & van Harmelen, 2004):

- Definición de clases mediante restricciones sobre propiedades, valores o cardinalidad.
- Definición de clases mediante operaciones booleanas sobre otras clases: intersección, unión y complemento.
- Relaciones entre clases (p.ej. inclusión, disyunción, equivalencia).
- Propiedades de las relaciones (p.ej. inversa, simétrica, transitiva).
- Cardinalidad (p.ej. "únicamente una").
- Igualdad y desigualdad de clases.
- Igualdad y desigualdad de instancias.
- Clases enumeradas.

Estas características introducidas en el lenguaje, si bien mejoran el poder expresivo del lenguaje, también limitan la capacidad de los razonadores para inferir nuevo conocimiento a partir de unas sentencias dadas. En particular, no se pueden garantizar ni (1) la *completitud computacional*, esto es, que el razonador encuentre todas las conclusiones válidas, ni (2) la *"decidibilidad" computacional*, esto es, que se obtenga la respuesta para cualquier entrada en un periodo de tiempo finito. Esto llevó a la creación de tres variantes de OWL que presentan diferentes estados en la relación expresividad/decidibilidad. De menor capacidad expresividad/mayor eficiencia de razonamiento a mayor capacidad expresiva/menor eficiencia de razonamiento nos encontramos las siguientes (McGuinness & van Harmelen, 2004):

- OWL Lite: permite una jerarquía de clasificación y restricciones simples (restricciones de cardinalidad, pero sólo valores de cardinalidad de 0 o 1). Entre las ventajas, se encuentran la facilidad de entender (por parte de los usuarios) y la facilidad de implementar (para constructores de herramientas). La mayor desventaja es, por supuesto, la restringida expresividad.
- OWL DL (del inglés '*Description Logic*', lógica descriptiva): es el lenguaje indicado para aquellos usuarios que requieren el máximo de expresividad, mientras conservan completamente la propiedad de ser computable (se garantiza que todas las conclusiones son computables) y resoluble (todos los cálculos

terminarán en tiempo finito). Incluye todos los constructores del lenguaje OWL, pero solamente se pueden utilizar bajo ciertas restricciones.

- OWL Full: permite la máxima expresividad y ofrece la libertad sintáctica de RDF pero carece de garantías en cuanto a la propiedad de ser computable.

Cada uno de estos sub-lenguajes es una extensión de su predecesor más simple, en los que ambos pueden ser expresados legalmente (esto es, las fórmulas bien formadas en OWL Lite son válidas en OWL DL y las bien formadas en OWL DL válidas en OWL Full) y en los que pueden ser válidamente concluidos (esto es, toda fórmula inferida en OWL Lite puede ser válidamente concluida en OWL DL partiendo del mismo conocimiento, y, con la misma asunción, las inferidas en OWL DL válidamente concluidas en OWL Full) (Fig. 3).

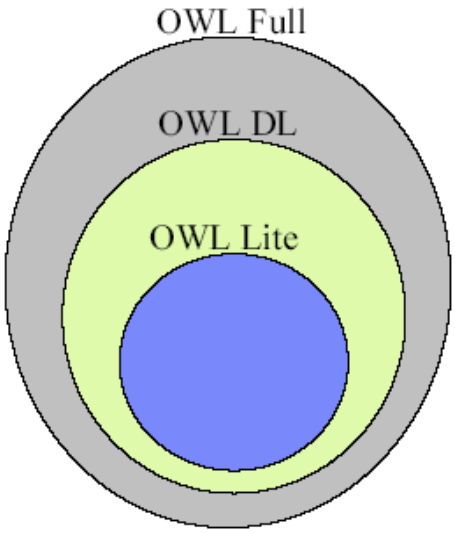


Fig. 3. Versiones del lenguaje OWL

El propósito de OWL es similar al de RDFS, esto es, servir de vocabulario XML para definir clases, sus propiedades y las relaciones entre clases. En comparación con RDFS, OWL permite expresar relaciones mucho más ricas y, por tanto, disponer de capacidades de inferencia mejoradas. La semántica formal y el soporte al razonamiento en OWL se garantizan a través del mapeo de OWL en formalismos lógicos utilizando la lógica de predicados (lógica de primer orden) y la lógica descriptiva (Antoniou & van Harmelen, 2005). Las capacidades de razonamiento que soporta OWL y que extienden lo indicado para RDF/RDFS se presentan en la Tabla 2.

Transitividad
`owl:TransitiveProperty`
	`(rdf:type progenitor owl:TransitiveProperty)`
	`(progenitor Blas Francisco)`
	`(progenitor Francisco Maria)`
	→
	`(progenitor Blas Maria)`

Propiedades inversas
`owl:inverseOf`
	`(owl:inverseOf padreDe tienePadre)`
	`(padreDe Francisco Maria)`
	→
	`(tienePadre Maria Francisco)`

Herencia de disyunción
`owl:disjointWith`
	`(owl:disjointWith Planta Animal)`
	`(rdfs:subClassOf Mamífero Animal)`
	→
	`(owl:disjointWith Planta Mamífero)`

Disyunción y Complementario
`owl:complementOf`
	`(owl:complementOf Animal NoAnimal)`
	`(rdfs:subClassOf Mamífero Animal)`
	→
	`(owl:disjointWith Mamífero NoAnimal)`

Pertenencia a clases	Si x pertenece a A y A es una subclase de B, entonces x pertenece a B
Equivalencia de clases	Si A es equivalente a B y B es equivalente a C, entonces A es equivalente a C
Consistencia	Si x pertenece a A y a B pero A y B son disjuntas, entonces se ha producido un error
Clasificación	Si para pertenecer a una clase A es necesario cumplir ciertas propiedades y x cumple dichas propiedades, entonces x pertenece a A

Tabla 2. Capacidades de razonamiento con OWL

I.2.4.4. Web Services Modeling Language (WSML)

WSML es una familia de lenguajes de representación para la Web Semántica (WSML, 2005). La semántica de esta familia de lenguajes está basada en lógica descriptiva, programación lógica y lógica de primer orden, y posee influencias de F-logic y sistemas de representación basados en marcos. Con WSML, se consigue un lenguaje formal que, junto con el modelo conceptual presentado por WSMO (Web Services Modeling Ontology), permite añadir semántica al mundo de los Servicios Web.

Existen numerosas variantes de WSML (Fig. 4). A continuación, se ofrece una breve descripción de cada una de ellas:

- **WSML-Core**: se basa en la intersección de la lógica descriptiva $\mathcal{SHIQ}$ y lógica de Horn, basado en programas lógico-descriptivos. Es la variante de WSML que ofrece menor expresividad. Entre las características principales de este lenguaje, destacan los conceptos, atributos, relaciones binarias y las instancias. También se permite la jerarquía de conceptos y relaciones, y se da soporte a los tipos de datos.

- **WSML-DL**: captura la lógica descriptiva $\mathcal{SHIQ}$(D), una parte muy importante de (la variante DL de) OWL. Además, incluye una extensión en los tipos de datos basada en OWL-DL que permite el uso de tipos de datos más complejos.
- **WSML-Flight**: es una extensión de WSML-Core que proporciona un lenguaje de reglas muy expresivo. Añade características como meta-modelado, restricciones y negación no-monótona. WSML-Flight se basa en una variante de programación lógica de F-Logic y es equivalente semánticamente a *Datalog* con desigualdad y negación (localmente) estratificada.
- **WSML-Rule**: extiende WSML-Flight con más características provenientes de la programación lógica como el uso de símbolos de función y reglas inseguras.
- **WSML-Full**: unifica WSML-DL y WSML-Rule bajo el soporte de la lógica de primer orden y con extensiones para soportar la negación no-monótona de WSML-Rule. Actualmente, todavía se sigue trabajando para definir completamente la semántica de WSML-Full.

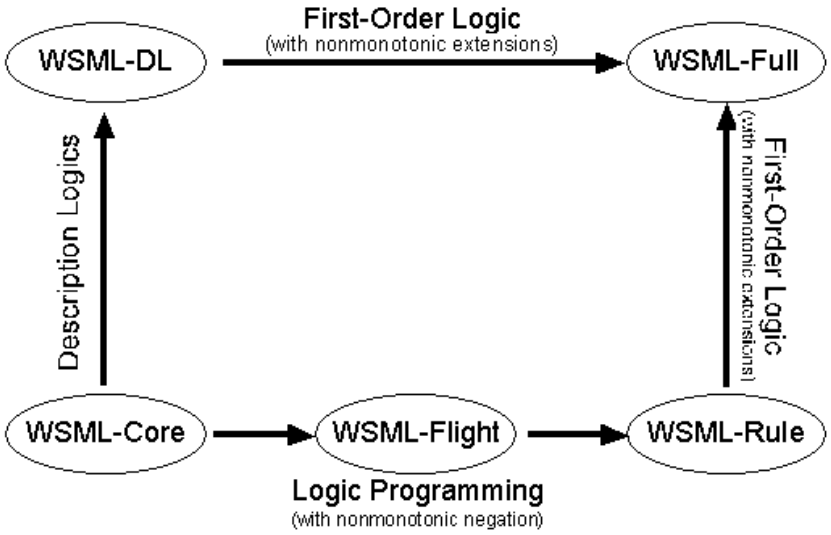


Fig. 4. Variantes de WSML (WSML, 2005)

Las diferentes variantes difieren en la expresividad lógica y en el formalismo lógico utilizado (Fig. 5), permitiendo al usuario disponer de diversas posibilidades con diferente relación entre expresividad y complejidad.

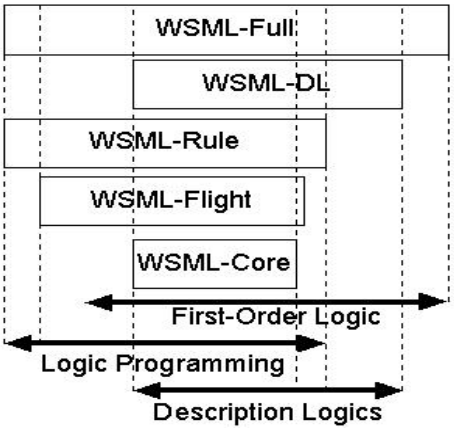


Fig. 5. Formalismos lógicos en las variantes de WSML (WSML, 2005)

I.3. Tecnología de Agentes

I.3.1. Agentes Inteligentes

Como ya vimos para el caso de las ontologías, no existe una definición consensuada del término '*agente*' ni acuerdo en cuanto a las propiedades que este tipo de entidades debe de presentar. El diseño de agentes inteligentes es una rama del mundo de la Inteligencia Artificial. En este dominio, una de las definiciones de agente más citadas es la establecida por Russell y Norvig (2004):

> *"Un agente es cualquier cosa capaz de percibir su medioambiente con la ayuda de sensores y actuar en ese medio utilizando actuadores."*

Esta definición se centra en el componente físico del término y en su interacción con el mundo que le rodea. Acercándonos más a la parte funcional del concepto, una definición comúnmente aceptada es la propuesta por Wooldridge y Jennings (1995), posteriormente adaptada por Wooldridge (2000):

> *"Un agente es un sistema computerizado que está **situado** en algún **entorno**, y que es capaz de actuar de forma **autónoma** en este entorno para satisfacer sus **objetivos** de diseño."*

En esta definición, ya se destacan algunas de las propiedades comunes que debe de poseer todo sistema compuesto por agentes, como son la autonomía y su localización en algún medio con el que tiene que interactuar. En la siguiente definición, Jennings (2001) señala otras características como la flexibilidad y la actuación en representación de

otros, además de enfatizar nuevamente la idea de que todo agente realiza las acciones pertinentes con el único propósito de alcanzar unos objetivos predefinidos:

> *"Los agentes son programas software que actúan **flexiblemente** en **representación** de sus propietarios para alcanzar unos **objetivos** particulares."*

Wooldridge (2000) destaca que todo agente, para llegar a ser denominado "inteligente", debe satisfacer el siguiente conjunto mínimo de propiedades:

> *"**Reactividad**: los agentes inteligentes son capaces de percibir su entorno y responder en un periodo de tiempo adecuado a cambios que ocurren en este para satisfacer sus objetivos de diseño.*
>
> ***Proactividad**: los agentes inteligentes son capaces de mostrar un comportamiento dirigido por objetivos tomando la iniciativa para satisfacer sus objetivos de diseño.*
>
> ***Habilidad social**: los agentes inteligentes son capaces de interaccionar con otros agentes (y posiblemente humanos) para satisfacer sus objetivos de diseño. "*

Jennnings y Wooldridge (1995) dan un paso más en esta definición y clasifican los agentes según una noción débil o fuerte de agencia. En la noción débil de agencia, los agentes tienen voluntad propia (autonomía), pueden interaccionar entre ellos (habilidad social), responder a estímulos (reactividad), y tomar la iniciativa (pro-actividad). En la noción fuerte, se preservan las propiedades de agencia débil, pero, además, los agentes pueden moverse alrededor de una red (movilidad), son veraces (veracidad), hacen lo que se les dijo que hicieran (benevolencia) y consiguen realizar los objetivos propuestos de manera óptima (racionalidad). Pero son innumerables las propiedades que se pueden esperar de un agente. Elamy (2005) enumera trece propiedades que, además de albergar las ya mencionadas (autonomía, reactividad, etc.), incluyen, entre otras, la continuidad temporal (un agente funciona sin cesar), la capacidad de aprendizaje, el razonamiento (como mecanismo de toma de decisiones) y la cooperación (comunicación e interacción entre agentes para conseguir un objetivo común).

La definición de agente que se asume en esta tesis doctoral es la siguiente: un agente inteligente es una entidad software situada en un entorno concreto, capaz de mostrar un comportamiento autónomo, reactivo y proactivo sobre este entorno y preparado para interaccionar con otros agentes con el único fin de satisfacer unos determinados objetivos establecidos por un ente (software o humano) que es representado por el agente.

I.3.2. Sistemas Multi-Agente

Los agentes pueden ser útiles como entidades independientes en entornos aislados a las que se les delegan ciertas tareas repetitivas y que se pueden automatizar en representación de unos determinados usuarios. Sin embargo, en la mayoría de los casos, los agentes se encuentran en entornos que contienen otros agentes constituyendo de este modo un Sistema Multi-Agente (SMA), esto es, un sistema constituido por un grupo de agentes que pueden interaccionar (Vlassis, 2003). Una definición de SMA más elaborada es la propuesta por el Laboratorio de Agentes Software Inteligentes (2001):

> *"Un Sistema Multi-Agente (SMA) es una red poco acoplada de agentes software que interaccionan para resolver problemas que van mas allá de las capacidades o conocimiento individual de cada uno de los componentes que resuelven problemas."*

Cuando un grupo de agentes individuales forma un SMA, surge la necesidad de disponer de un mecanismo para coordinar dicho grupo de agentes y de un lenguaje para permitir la comunicación entre ellos. Entre los mecanismos de coordinación, se pueden distinguir los casos en los que los agentes tienen objetivos comunes y, por tanto, *cooperan*, de los casos en los que los agentes son "auto-interesados" y tienen objetivos conflictivos con los demás, para lo cual precisaremos de mecanismos de *negociación* (Huhns et al., 1999). De forma análoga a esta clasificación, Wooldridge (2002) distingue entre *sistemas distribuidos de resolución de problemas* (constituidos por agentes diseñados explícitamente para conseguir de forma cooperativa un objetivo dado) y *sistemas abiertos* (donde confluyen agentes elaborados por distintos desarrolladores y que poseen posiblemente objetivos diferentes).

En el caso de agentes cooperativos, los *mecanismos de cooperación* más comunes son las estructuras organizacionales, la planificación multi-agente (centralizada y distribuida), redes de contratos y cooperación funcionalmente exacta. Por otro lado, para el caso de agentes competitivos, se hace necesario un *mecanismo de negociación*. Entre los tipos de mecanismos de negociación más utilizados en la literatura destaca la formación de coaliciones, los mecanismos de mercados, la teoría del regateo, la votación, las subastas y la asignación de tareas entre dos agentes.

Para que se pueda producir una comunicación efectiva entre los agentes, se han elaborado diversos lenguajes. Estos lenguajes fueron diseñados bajo la influencia de la *Teoría de los actos del habla* (Austin, 1962; Searle, 1969). Esta teoría está basada, fundamentalmente, en la existencia de los denominados *actos del habla* o *performativas*

(p.ej. solicitar, informar, prometer). Por tanto, las conversaciones se basan en estos actos del habla y proporcionan el potencial necesario para la coordinación, la cooperación y la negociación. La coordinación se puede definir como la manera en que los agentes se comportan individual y socialmente para que se satisfagan los objetivos personales y los globales. Por su parte, la cooperación puede verse como la actuación coordinada entre agentes de tal manera que unos colaboran, interesada o desinteresadamente, en la resolución de tareas de otros. Por último, la negociación es un proceso mediante el cual un grupo de agentes llegan a un acuerdo mutuamente aceptable sobre algún asunto. Se indican más detalles acerca de la negociación y sus posibilidades en la sección I.3.3.

Los lenguajes de comunicación entre agentes más destacables son KQML (*Knowledge Query and Manipulation Language*), KIF (*Knowledge Interchange Format*) y FIPA-ACL (*FIPA Agent Communication Language*). KQML se ocupa de definir un formato común para los mensajes sin meterse en el contenido de los mismos (Finin et al., 1995). Cada mensaje en KQML está formado por una performativa que define el tipo de mensaje y un conjunto de parámetros (p.ej. contenido, remitente, receptor). KIF es un lenguaje que permite representar conocimiento sobre un determinado 'dominio del discurso' (Genesereth & Fikes, 1992). KIF es usado, principalmente, para formar la parte del contenido de mensajes KQML. FIPA-ACL es similar a KQML en tanto que define el formato de los mensajes sin preocuparse ni implicar el uso de un lenguaje específico para el contenido de los mensajes (FIPA, 2002b).

La programación orientada a agentes es un nuevo paradigma de programación basado en SMA. El uso de SMA implica una serie de beneficios sobre la aplicación de otros paradigmas de programación. Elamy (2005) destaca las siguientes ventajas del uso de los SMA:

- Fiabilidad: los SMA son más robustos y tolerantes a fallos.
- Modularidad y escalabilidad: los agentes pueden ser añadidos o borrados de un entorno concreto sin la necesidad de interrumpir o detener el sistema.
- Adaptabilidad: los agentes tienen la habilidad de reconfigurarse de forma que se adapten a fallos y cambios.
- Concurrencia: los agentes son capaces de razonar y realizar tareas en paralelo.
- Dinamismo: los agentes pueden colaborar dinámicamente para compartir sus recursos y resolver problemas.

En el futuro, se atisban diferentes desafíos que la tecnología de agentes deberá afrontar. Sycara (1998) identifica seis retos principales a la hora de diseñar e implementar SMA: (1) formular, describir y descomponer problemas y asignar tareas a agentes individuales; (2) hacer que agentes heterogéneos sean capaces de comunicarse e interaccionar; (3) hacer que los agentes individuales actúen de forma coherente para realizar acciones y tomar decisiones; (4) hacer que los agentes individuales reacciones y razonen sobre las acciones, planes y el conocimiento de otros agentes; (5) reconocer y reconciliar intenciones y objetivos conflictivos entre agentes coordinados; y (6) alcanzar una visión mas ingenieril de los sistemas de inteligencia artificial distribuida empleando metodologías para el desarrollo de SMA.

Por otro lado, Luck y McBurney (2004) destacan, en un nivel de abstracción superior, los siguientes retos tecnológicos para la investigación y el desarrollo de SMA en la próxima década: (1) incrementar la calidad del software de agentes hasta alcanzar el estándar industrial; (2) acordar estándares efectivos para permitir el desarrollo de sistemas abiertos; (3) proporcionar infraestructura semántica a las comunidades abiertas de agentes; (4) desarrollar capacidades de razonamiento para agentes en entornos abiertos; (5) desarrollar la habilidad del agente para entender los requisitos del usuario; (6) desarrollar la habilidad del agente para adaptarse a cambios en el entorno; (7) asegurar la confianza del usuario y la confiabilidad de los agentes. Ciertamente, en los últimos 5 años, la mayor parte de la investigación en el campo de los agentes inteligentes se ha dirigido a la resolución de alguna de las dificultades aquí mencionadas.

I.3.3. Negociación en Sistemas Multi-Agente

El SMA diseñado como resultado de esta tesis doctoral contiene un número de agentes que, si bien comparten los objetivos globales de mantener satisfechos a los usuarios y un correcto funcionamiento del sistema, difieren en sus objetivos a corto plazo. Sin duda, es fácil de deducir que, en un entorno donde unos agentes representan a usuarios proveedores de servicios y otros a usuarios consumidores de estos servicios, los agentes disponen de objetivos que están en conflicto (unos pretenden conseguir el máximo beneficio para los proveedores y otros obtener los servicios más adecuados de acuerdo con las necesidades de los consumidores). Por tanto, en un entorno de estas

características, se hace necesario un proceso de negociación. En este apartado, se define el concepto de negociación y se indican cuáles son los componentes necesarios para llevar a cabo, de forma satisfactoria, un proceso de negociación en un SMA.

La negociación en interacciones humanas como medio para alcanzar decisiones conjuntas ha sido objeto de investigación durante muchos años. Los resultados de este tipo de investigación en este contexto han servido como base para la investigación de los procesos de negociación automatizados, esto es, los llevados a cabo por componentes software. En general, la negociación se puede ver como el proceso por el cual dos (negociación bilateral) o más partes (negociación multilateral) pueden obtener una decisión conjunta. En el dominio del comercio electrónico, por ejemplo, se define negociación como el proceso por el cual dos o más entidades regatean sobre recursos multilateralmente con el propósito de alcanzar una ganancia conjunta (Beam & Segev, 1997). La negociación llevada a cabo por humanos es relativamente lenta y sub-óptima (esto es, no se obtiene el mejor resultado posible para todas las partes), de forma que la negociación automatizada cobra mayor interés.

La negociación automatizada, foco de interés en este trabajo, tiene lugar cuando la función de negociación la realizan sistemas computerizados como, por ejemplo, los agentes software. El proceso de negociación se puede entender como un tipo de interacción automática entre las distintas partes que mantienen propósitos y objetivos diferentes (Narayanan & Jennings, 2005). Aunque la negociación entre humanos es un proceso extremadamente complejo, los agentes que se encuentren llevando a cabo este proceso en un entorno automatizado no deben de incorporar dicha complejidad. Ciertamente, la unión de diversos agentes 'sencillos' puede llegar a constituir un entorno que actúe de forma muy sofisticada (Beam & Segev, 1997). Fatima et al. (2004) propone una definición muy sencilla de negociación en entornos multi-agente:

> "*Negociación es el medio por el cual los agentes se comunican y se comprometen a alcanzar acuerdos mutuamente beneficiosos.*"

En esta definición, se deja claro el hecho de que en los procesos de negociación se debe de buscar un acuerdo que satisfaga a todas las partes en la negociación. Es posible llegar a una situación en la que una negociación no llegue a un final satisfactorio por el hecho de que alguna de las partes no esté de acuerdo con ninguna de las propuestas realizadas. Sin embargo, en la mayoría de los casos, esta es la situación menos deseada y, por tanto, siempre se busca llegar a buen término el proceso de negociación por

medio de concesiones de las partes. Rahwan et al. (2004) plantea otra definición donde son resaltados otros elementos importantes del proceso de negociación en SMA:

> *"Negociación es una forma de interacción en la cual un grupo de agentes, con intereses conflictivos y un deseo de cooperar, intentan llegar a un acuerdo mutuamente aceptable en la división de recursos escasos."*

En esta definición, se destaca la necesidad de que los agentes tengan un deseo en cooperar que, tal y como se mencionó anteriormente, indica que el final menos deseado es aquel en el que las partes no alcanzan un acuerdo. Por otro lado, se utiliza el término 'recurso', en el sentido más amplio de la palabra, para identificar servicios, tiempo, dinero, artículos, etc. En general, la necesidad de negociar surge cuando se tiene que compartir el uso de uno de estos recursos.

Lomuscio et al. (2001) han identificado dos componentes básicos en todo sistema de negociación automática: el *protocolo de negociación* y las *estrategias de negociación*. Un protocolo de negociación define formalmente las reglas por las cuales se rige el proceso de negociación, esto es, las circunstancias bajo las cuales la interacción entre los agentes tiene lugar (qué tratos se pueden hacer y qué secuencia de ofertas son permitidas). Por otro lado, una estrategia de negociación de un agente es la especificación de la secuencia de acciones, ofertas y respuestas, que el agente planea hacer durante la negociación (muchas estrategias tienen cabida para un mismo protocolo y, posiblemente, cada una produciría un resultado diferente). Un *mecanismo de negociación* está compuesto por un protocolo de negociación junto con las estrategias de negociación para los agentes involucrados. Fatima et al. (2004) extiende este modelo con dos componentes adicionales: el *estado de la información de los agentes* y el *equilibrio de negociación*. El primero de estos componentes se encarga de describir la información que los agentes tienen acerca del juego de la negociación. Se distingue entre los casos en que los agentes disponen de información completa acerca de las 'reglas del juego' y las preferencias de las distintas partes y los casos en los que no. Un proceso de negociación se encuentra en equilibrio cuando éste es estable y la estrategia de los agentes que forman parte de la negociación ofrece la mejor respuesta dadas las estrategias de sus oponentes (así se alcanza el *equilibrio de Nash*). Jennings et al. (2001) proponen otro modelo distinto, extendido posteriormente por Narayanan y Jennings (2005), compuesto por cinco componentes y que comparte alguno de los mencionados anteriormente: (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, penalizaciones, etc.); (3) preferencias de negociación de los participantes, esto es, 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; y (5) parámetros de negociación de los participantes, los cuales establecen distintas restricciones como el tiempo límite para alcanzar un acuerdo.

Existen tres técnicas o aproximaciones principales a la negociación automatizada (Jennings et al., 2001; Rahwan et al., 2004):

- **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. Las aproximaciones clásicas basadas en Teoría de Juegos tienen algunas limitaciones computacionales muy importantes (asumen que los agentes tiene recursos computacionales infinitos y que el espacio de los resultados es completamente conocido, lo cual es falso en la mayoría de los casos).
- **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), y (2) es muy difícil predecir precisamente cómo el sistema se comportará.
- **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.

En general, es importante tener en cuenta que no existe una solución válida para todos los problemas y dominios (Rahwan et al., 2004), sino que debe ser el diseñador quien, dependiendo de las condiciones en las que el proceso de negociación se vaya a producir, elija la técnica a aplicar en cada momento. Dada esta limitación y los requisitos a satisfacer para el desarrollo con éxito del entorno multi-agente objetivo de esta tesis doctoral, resulta manifiesta la necesidad de encontrar una solución que permita a los agentes, dinámicamente y en tiempo de ejecución, determinar el mecanismo de negociación más apropiado para cada interacción que deba de producirse. La solución elegida, que será detallada en secciones posteriores, se basa en el trabajo de Tamma et al. (2005) y consiste en codificar distintos protocolos de negociación, y las estrategias asociadas a los mismos, mediante el uso de ontologías, de forma que son los agentes quienes, en el momento que precisen llevar a cabo una interacción, deciden qué mecanismo (esto es, qué protocolo y qué estrategia) desean utilizar.

I.3.4. Metodologías para el diseño de Sistemas Multi-Agentes

Como ya se mencionó anteriormente, la *Programación Orientada a Agentes* (POA) es un nuevo paradigma de programación diseñado con el propósito de mejorar y extender el paradigma de programación orientada a objetos. En POA, son agentes, y no objetos, los que forman el sistema constituyendo un SMA. Wooldridge (2002) apunta tres diferencias principales entre los conceptos de agente y de objeto:

1. Un agente decide, de acuerdo con su estado interno, si desea o no realizar las acciones que le han sido solicitadas por los otros agentes, mostrando así un comportamiento más autónomo que el de los objetos.
2. Los agentes pueden mostrar un comportamiento flexible (reactivo, proactivo o social), mientras que los objetos no.
3. Un SMA es inherentemente multi-hilo, dado que cada agente tiene al menos un hilo de control.

Estas diferencias hacen que los SMA sean apropiados especialmente para el desarrollo de sistemas que operan en entornos complejos, dinámicos e impredecibles tales como control aéreo, control autónomo de naves espaciales, servicios médicos, y sistemas de control industrial. Por esto, y dado que la influencia de unos agentes en otros viene dada no sólo por la comunicación explícita, sino también por la actuación sobre el entorno, es fácil deducir que el desarrollo de SMA es una tarea muy

complicada. Como consecuencia de esto, surgió recientemente lo que se denomina *Ingeniería del Software Orientada a Agentes* (AOSE por sus siglas en inglés, 'Agent-Oriented Software Engineering) con el propósito de crear metodologías y herramientas que permitiesen un desarrollo y mantenimiento económico de software basado en agentes (Tveit, 2000).

En esta sección, se trata el tema de las metodologías dentro de la investigación en AOSE, mientras que la siguiente sección está dedicada a las plataformas que implementan estándares de desarrollo de SMA. Una metodología proporciona los medios adecuados para construir SMA de forma disciplinada y repetible. Las metodologías ideadas hasta el momento para facilitar y mejorar el proceso de producción de los SMA se basan, fundamentalmente, en las metodologías de software tradicionales a las que trasladan los requerimientos específicos de este nuevo paradigma de programación. Entre las más conocidas de dicho tipo de metodologías, se encuentran las siguientes: MAS-CommonKADS, ZEUS, GAIA, MaSE, MOBMAS e INGENIAS. Algunas de estas metodologías se discuten a continuación, conforme al estudio realizado por Gómez-Sánz y Pavón (2004). Se pone especial énfasis en lo relativo a la metodología INGENIAS, dado que ésta ha sido la metodología utilizada para el diseño del entorno presentado en esta tesis doctoral.

I.3.4.1. INGENIAS

INGENIAS[1] es una metodología de desarrollo de SMA que proporciona un conjunto de métodos y herramientas para desarrollar dicho tipo de sistemas (Gómez-Sanz & Fuentes, 2002; Pavón & Gómez-Sanz, 2003). El método de desarrollo de SMA propuesto en INGENIAS concibe el SMA como la representación computacional de un conjunto de modelos. Cada uno de estos modelos muestra una visión parcial del SMA: los agentes que lo componen, las interacciones que existen entre ellos, cómo se organizan para proporcionar la funcionalidad del sistema, qué información es relevante en el dominio y cómo es el entorno en el que se ubica el sistema a desarrollar. Para presentar estas 'vistas', INGENIAS define un conjunto de meta-modelos (esto es, una descripción de alto nivel de qué elementos tiene un modelo: entidades, relaciones y restricciones) que comprenden toda la información que se debe tener en cuenta al especificar un SMA. Existen un total de cinco meta-modelos definidos en INGENIAS:

- Meta-modelo de agente: describe agentes particulares, sus tareas, objetivos, los roles que desempeñan y los estados mentales en que se encontrarán a lo largo de su vida.
- Meta-modelo de tareas y objetivos: define las relaciones entre objetivos y tareas, estructuras de objetivos y estructuras de tareas. También permite indicar las entradas y salidas de cada tarea y cuales son los efectos de las mismas, tanto en el entorno como en el estado mental de los agentes.
- Meta-modelo de organización: describe cómo se agrupan los distintos componentes del sistema (agentes, roles, recursos y aplicaciones), la funcionalidad del sistema y las restricciones que hay que imponer sobre la interacción entre los agentes.
- Meta-modelo de interacción: detalla cómo se coordinan y comunican los agentes. Cada declaración de interacción incluye los actores, el objetivo perseguido y el protocolo que sigue la interacción.
- Meta-modelo de entorno: define qué existe alrededor del nuevo sistema y cómo lo percibe cada agente. También identifica los recursos del sistema y quién es el responsable de su gestión.

El proceso de instanciación de los meta-modelos (esto es, la producción de los diagramas o modelos que representan cada una de las vistas de un SMA particular) no es trivial, ya que existen muchas entidades y relaciones a identificar, además de dependencias entre distintos modelos. Por ello, INGENIAS define un conjunto de actividades en el proceso de desarrollo de software cuya ejecución termina en la especificación final del SMA. Estas actividades están organizadas de acuerdo con las relaciones entre los meta-modelos de INGENIAS con los elementos que incorpora la metodología *Rational Unified Process* (RUP). En la tabla que se presenta a continuación, se puede observar cada una de las actividades que tienen lugar en las fases de inicio, elaboración y construcción (ver Tabla 3).

[1] http://grasia.fdi.ucm.es/ingenias/

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

Tabla 3. Actividades a realizar en las etapas de inicio, elaboración y construcción (Pavón y Gómez-Sanz, 2003)

Para terminar, es importante destacar que la metodología INGENIAS viene, además, acompañada por una herramienta para el modelado visual, el INGENIAS Development Kit (IDK), actualmente en la versión 2.6 beta. Mediante esta herramienta, es posible la construcción de todos los modelos que se precisan de acuerdo con la metodología, e incluye un generador automático de código capaz de devolver código para JADE, la plataforma multi-agente elegida para el desarrollo presentado en esta tesis doctoral, y de la que se ofrecen más detalles en las próximas secciones.

I.3.4.2. Otras

En este apartado, se indica un breve resumen de las principales propiedades de algunas de las metodologías para SMA más conocidas y utilizadas. Este sumario se basa en el estudio realizado por Gómez-Sánz y Pavón (2004). En particular, se describen las características de cinco metodologías: MAS-CommonKADS, MaSE, ZEUS, GAIA y MOBMAS.

MAS-CommonKADS (Iglesias, 1998) extiende la metodología para el desarrollo de sistemas expertos CommonKADS aplicando ideas de metodologías orientadas a objetos para su aplicación a la producción de SMA. En particular, extiende los modelos de

CommonKADS para tener en cuenta la posibilidad de que dos o más componentes del sistema interactúen. Son siete los modelos disponibles: agente, tareas, experiencia, coordinación, comunicación, organización y diseño. MAS-CommonKADS plantea el desarrollo de SMA integrado con un ciclo de vida de software en espiral dirigido por riesgos.

MaSE (Multi-agent systems Software Engineering) (DeLoach, 2001) parte del paradigma orientado a objetos asumiendo que un agente es una especialización de un objeto capaz de coordinarse con otros agentes a través de conversaciones y de actuar proactivamente (esto es, por iniciativa propia) para alcanzar metas individuales y del sistema. AgentTool es la herramienta que acompaña a MaSE que habilita la ejecución de la mayoría de los pasos de los que consta el proceso de desarrollo especificado en MaSE. Este proceso está dividido en dos fases, la fase de análisis y la de diseño. En la fase de análisis, los pasos a realizar consisten en la captura de los objetivos, captura de los casos de uso y el refinamiento de los roles. El diseño consta de cuatro pasos: crear clases de agentes, construir conversaciones, ensamblar clases de agentes y realizar el diseño del sistema. AgentTool, además, permite generar código (independiente del lenguaje) automáticamente a partir de la especificación del sistema (los diagramas finales).

ZEUS (Collis y Ndumu, 1999) propone un desarrollo en cuatro etapas: el análisis del dominio, el diseño de los agentes, la realización de los agentes y el soporte en tiempo de ejecución. La metodología ZEUS viene acompañada con una herramienta que abarca y soporta las etapas de realización de los agentes y de soporte en tiempo de ejecución. En comparación con MaSE, ZEUS es conceptualmente superior, ya que incorpora más elementos relativos a la tecnología de agentes (planificación, definición de ontologías, relaciones sociales entre agentes, asignación de responsabilidades y secuenciación de tareas). Sin embargo, metodológicamente MaSE es más poderosa, ya que se acerca más a las prácticas de ingeniería convencional.

GAIA (Wooldridge et al., 2000; Zambonelli et al., 2003) es una metodología para el diseño de sistemas basados en agentes cuyo objetivo es obtener un sistema que maximice la calidad global ayudando al analista a ir de forma sistemática desde los requisitos iniciales al diseño. El proceso metodológico consta de dos etapas: el análisis de alto nivel, donde se elaboran los modelos de roles y de interacciones, y el diseño de alto nivel, donde se generan los modelos de agentes, de servicios y de conocidos. A

partir de aquí, se propone la aplicación de técnicas clásicas de diseño orientado a objetos. En general, esta metodología pretende especificar cómo una sociedad de agentes colabora para alcanzar los objetivos del sistema y qué se requiere de cada uno para lograr esto último, pero siempre manteniendo un alto nivel de abstracción de forma que el diseño sea independiente de una solución de implementación de agentes concreta.

MOBMAS (Tran, 2005) es una metodología que incluye de forma explícita un soporte extensivo del uso de las ontologías en el diseño del SMA. El proceso de desarrollo propuesto por la metodología MOBMAS consiste en cinco actividades: (1) *actividad de análisis*, durante la cual se constituye una concepción del SMA identificando los roles y las tareas precisas a partir de la ontología del dominio y los requerimientos del sistema; (2) *diseño de organización del SMA*, donde se especifica la estructura organizacional del SMA que se tiene como objetivo y donde se identifican el conjunto de agentes y recursos que componen el sistema; (3) *diseño interno de los agentes*, en la que se especifica, para cada agente, sus objetivos, la conceptualización de sus creencias, los eventos, plantillas de sus planes y las reglas reactivas; (4) *diseño de interacción de los agentes*, donde se definen las interacciones que pueden tener lugar entre los agentes, eligiendo el mecanismo de interacción más apropiado y los patrones de intercambio de datos; y (5) diseño de la arquitectura, donde se trata con asuntos relacionados con la arquitectura como la identificación de los requisitos de la interfaz de los agentes, selección de la arquitectura, identificación de las propiedades requeridas de la infraestructura e instanciación y configuración de los agentes.

La elección de una metodología u otra no es una tarea sencilla y, en general, depende del dominio de aplicación y de las características del sistema final que se pretenda construir. En muchas ocasiones, es apropiado disponer de una herramienta que dé soporte a la metodología y permita la elaboración de los diagramas y modelos propuestos por la metodología de una manera más eficiente. En el caso particular de esta tesis, se estuvieron barajando diversas posibilidades entre las que destacan MOBMAS e INGENIAS. MOBMAS se adecua bastante a las características del entorno por su uso exhaustivo de las ontologías (elemento central del sistema desarrollado) pero carece de las herramientas de soporte apropiadas y de una evaluación completa de sus capacidades a partir de su aplicación en entornos reales y complejos. Por su parte, INGENIAS tiene a su favor la existencia de un kit de desarrollo, la estrecha relación con la plataforma JADE (genera código para este entorno a partir de los diagramas

elaborados) y la madurez y robustez de su proceso de desarrollo, características que finalmente han fundamentado la elección de la misma.

I.3.5. Plataformas Multi-Agente

La investigación en AOSE, además del desarrollo de metodologías, también promueve la creación de herramientas que permitan la interconexión y ejecución de los agentes de un SMA, a las cuales se denomina *plataformas de agente* ó *multi-agente*. Existe una gran cantidad de plataformas multi-agente disponibles que, en su mayoría, siguen el estándar y modelo de referencia definido por FIPA (The Foundation for Intelligent Physical Agents)[2], que engloba a grupos industriales y de investigación con el objetivo de estandarizar los modelos y tecnologías de agentes. En esta sección, se estudia en detalle la arquitectura propuesta por FIPA y se describen las características principales de las plataformas más conocidas que satisfacen este modelo de referencia: JADE, que ha sido la plataforma elegida en la investigación descrita en esta tesis y que es comentada en mayor profundidad, FIPA-OS y ZEUS.

I.3.5.1. Modelo de Referencia FIPA

Como se mencionó anteriormente, FIPA es un consorcio internacional que aglutina tanto a instituciones académicas como a compañías con intereses en el estudio y la promoción de la tecnología de agentes. FIPA es la organización encargada de producir especificaciones para la interacción de agentes y sistemas de agentes heterogéneos con el objetivo último de permitir la interoperabilidad en estos tipos de sistemas. En la actualidad, FIPA es uno de los comités de estándares de la "*IEEE Computer Society*". Desde su aparición en 1996, FIPA ha generado numerosos estándares, entre los que destacan la especificación de una arquitectura abstracta y el lenguaje de comunicación entre agentes (FIPA-ACL). En este apartado, se enumeran los elementos y componentes más importantes que se tienen en cuenta en la arquitectura propuesta por FIPA.

La Arquitectura Abstracta de FIPA (FIPA, 2002a) define cómo dos agentes son capaces de localizarse y comunicarse entre sí por medio de mecanismos que les permiten registrarse e intercambiar mensajes. Para esto, FIPA enumera un conjunto de elementos arquitecturales y las relaciones entre ellos, pero siempre en un nivel alto de abstracción, indicando únicamente el comportamiento externo (interfaz), dejando las

[2] http://www.fipa.org

decisiones de diseño a cada equipo de desarrollo (Ana Mas, 2005, p. 52). Además, se establece un modelo lógico relativo a la creación, destrucción, registro, localización y comunicación de agentes. El propósito final es la especificación de una arquitectura independiente del protocolo de transporte, del lenguaje de comunicación entre agentes, del lenguaje de contenido y del servicio de directorio concreto utilizados.

Una plataforma de agentes FIPA se define como el software que implementa un conjunto de especificaciones FIPA. Para que se considere que sigue las normas de FIPA, una plataforma debe implementar, al menos, la especificación sobre la gestión de agentes y las relativas al lenguaje de comunicación de agentes. La primera se ocupa del control y gestión de agentes dentro y a través de plataformas de agentes. Las segundas se encargan del formato de los mensajes, los protocolos de interacción y de intercambio de mensajes entre agentes, la descripción de los actos comunicativos que definen la semántica de los mensajes intercambiados y los diferentes lenguajes para expresar el contenido de un mensaje (lenguaje de contenido) (Amor et al., 2003). En la siguiente figura, se presenta el modelo de referencia FIPA (ver Fig. 6):

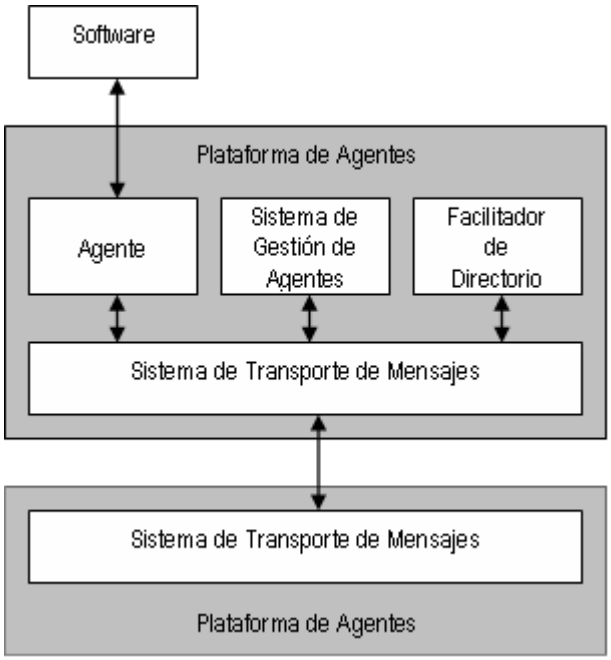


Fig. 6. Modelo de Referencia de FIPA (FIPA, 2004)

Los componentes básicos que deben formar parte de toda plataforma de agentes según el modelo de referencia de FIPA son los siguientes (FIPA, 2004; Fig. 6):

- **Agente**. Se trata de un proceso computacional que implementa la funcionalidad autónoma y comunicativa de una aplicación. Un agente debe tener, al menos, un propietario y debe poseer un identificador que permita distinguirlo sin ambigüedades del resto de agentes. Dependiendo de la implementación concreta, un agente puede ser un objeto Java, un componente COM, un programa Lisp auto-contenido o un script TCL (FIPA, 2002a).
- **Facilitador de Directorio** (*Directory Facilitator*, DF). Es un componente opcional de la plataforma. Proporciona un servicio de *páginas amarillas* que permite buscar un agente por sus capacidades, y no sólo por su nombre. Los agentes se registran en el DF indicando los servicios que ofrecen. Cuando otro agente tiene unas necesidades concretas, lanza una búsqueda del servicio deseado obteniendo los agentes que le ofrecen estos servicios.
- **Sistema de Gestión de Agentes** (*Agent Management System*, AMS). Es un componente obligatorio de la plataforma. El AMS ejerce un control de supervisión sobre el acceso y uso de la plataforma de agentes. En cada plataforma, sólo puede existir un único AMS. Entre sus responsabilidades, están la creación, destrucción y control del cambio de estado de los agentes, supervisión de los permisos para que nuevos agentes se registren en la plataforma, control de la movilidad de los agentes, gestión de los recursos compartidos y gestión del canal de comunicación (Ana Mas, 2005, p. 52). El AMS ofrece un servicio de *páginas blancas* mediante el cual se asocia a cada agente (su identificador único) la dirección de transporte real en la que se encuentra este agente, proporcionando así un método básico para la búsqueda de agentes.
- **Sistema de Transporte de Mensajes** (*Message Transport Service*, MTS). Es el método de comunicación por defecto entre agentes de una plataforma y entre agentes de distintas plataformas. Todo agente debe tener acceso a un MTS, ya que éste es el encargado de hacer llegar los mensajes ACL desde su agente origen a su agente destino. El modelo de comunicación entre agentes es asíncrono, lo que implica que el MTS no se queda bloqueado ante el envío o recepción de mensajes y que existen colas de envío y recepción de mensajes (Ana Mas, 2005, p. 53).
- **Plataforma de Agentes** (*Agent Platform*, AP). Proporciona la infraestructura física en la cual los agentes pueden ser desplegados. La AP está compuesta por la máquina o máquinas (recursos hardware), el sistema operativo, el software de

gestión de agentes, los componentes de gestión de agentes FIPA (DF, AMS y MTS) y los agentes (recursos software), esto es, todo lo necesario para poner en marcha la infraestructura.

- **Software**. Representa a todas las colecciones de instrucciones ejecutables que no son parte de los agentes pero que son accesibles a través de los mismos. Los agentes pueden acceder al software para, por ejemplo, añadir nuevos servicios, incorporar nuevos protocolos de comunicación, procurar nuevos algoritmos y protocolos de seguridad, permitir nuevos protocolos de negociación, etcétera.

Estos componentes se basan en los distintos elementos que se definen en la arquitectura abstracta de FIPA (FIPA, 2002a). Los elementos abstractos que son obligatorios en toda implementación de la arquitectura junto con una breve descripción de los mismos se presentan en la tabla siguiente (Tabla 4):

Elemento	Descripción
Estado-acción	Indicación de estado enviada por un servicio y que muestra si una acción se ha ejecutado con éxito o no.
Agente	Proceso computacional que implementa la funcionalidad autónoma y comunicativa de una aplicación.
Lenguaje-comunicación-agente	Lenguaje con sintaxis, semántica y pragmática definidas de forma precisa, el cual es la base de la comunicación entre agentes diseñados y desarrollados independientemente.
Entrada-directorio-agente	Entidad compuesta que contiene el nombre, localizador-agente y atributos-agente de un agente.
Servicio-directorio-agente	Aun servicio que provee un repositorio de información compartida en el que las entrada-directorio-agente pueden ser almacenadas y consultadas.
Localizador-agente	Consiste en un conjunto de descripción-transporte usados para comunicarse con un agente.
Nombre-agente	Un token opaco y no falsificable que identifica de forma unívoca a un agente.
Contenido	La parte de un mensaje (acto comunicativo) que representa el componente dependiente del dominio de la comunicación.
Lenguaje-contenido	Un lenguaje para expresar el contenido de una comunicación entre agentes.
Representación-codificación	Modo de representar una sintaxis abstracta en una sintaxis particular (p.ej. XML, FIPA Strings, objetos Java serializados).
Servicio-codificación	Servicio que codifica un mensaje a y desde una carga-útil.
Sobre	Parte del mensaje-transporte que contiene información acerca de cómo enviar el mensaje al receptor. Puede incluir información adicional.
Mensaje	Unidad de comunicación entre dos agentes. Se expresa en un lenguaje-comunicación-agente y se codifica en una representación-codificación.
Servicio-transporte-mensaje	Servicio que permite enviar y recibir mensaje-transporte entre agentes.
Carga-útil	Un mensaje codificado de un modo apropiado para su inclusión en un mensaje-transporte.
Servicio	Servicio proporcionado por agentes y otros servicios.
Dirección-servicio	Cadena específica para un tipo-servicio que contiene información sobre el direccionamiento del transporte.
Entrada-directorio-servicio	Entidad compuesta que contiene el nombre-servicio, localizador-servicio, y tipo-servicio de un servicio.

Servicio-directorio-servicio	Un servicio de directorio para registrar y descubrir servicios.
Nombre-servicio	Identificador único de un servicio en particular.
Descripción-localización-servicio	Una tupla clave-valor que contiene un tipo-firma, una firma-servicio y dirección-servicio.
Localizador-servicio	Consiste en un conjunto de descripción-localización-servicio usados para acceder a un servicio.
Raíz-servicio	Conjunto de entrada-directorio-servicio.
Firma-servicio	Identificador que describe la firma de comprobación para un servicio.
Tipo-servicio	Tupla clave-valor describiendo el tipo de un servicio.
Transporte	Servicio de entrega de datos sustentado por un servicio-transporte-mensaje.
Descripción-transporte	Estructura auto-descriptiva que contiene tipo-transporte, dirección-específica-transporte y cero o más propiedad-específica-transporte.
Mensaje-transporte	Objeto transmitido de agente a agente. Contiene la descripción-transporte para el remitente y receptor o receptores, junto con una carga-útil que contiene el mensaje.
Dirección-específica-transporte	Dirección de transporte específica para un tipo-transporte dado.
Tipo-transporte	Describe el tipo de transporte asociado con una dirección-específica-transporte.

Tabla 4. Descripción de los elementos obligatorios de la Arquitectura Abstracta de FIPA (FIPA, 2002a)

A pesar de este intento de estandarización, en las implementaciones del modelo de referencia de FIPA, como Jade, FIPA-OS o ZEUS, pueden hallarse diferencias debidas a las distintas posibles interpretaciones que se pueden tomar de la descripción de las distintas entidades (Amor et al., 2003). Esto lleva desafortunadamente a la existencia de incompatibilidades entre estos sistemas que, en la mayoría de los casos, impiden la correcta comunicación entre los agentes implementados en diversas plataformas.

I.3.5.2. Jade

La plataforma JADE[3] (Java Agent DEvelopment Framework) es la implementación más extendida del estándar de FIPA. JADE facilita el desarrollo de aplicaciones basadas en agentes (SMA) de acuerdo con las especificaciones de FIPA para habilitar la interoperabilidad entre SMA. La plataforma JADE, basada en una arquitectura de comunicación peer-to-peer, ha sido desarrollada completamente en Java y es de código abierto. JADE ha sido diseñada de acuerdo con los siguientes principios básicos (Bellifemine et al., 2003):

- Interoperabilidad: JADE cumple las especificaciones de FIPA y, por consiguiente, los agentes JADE pueden actuar conjuntamente con otros agentes, siempre que éstos estén definidos/implementados conforme al mismo estándar.

[3] http://jade.tilab.com/

- Uniformidad y portabilidad: JADE provee un conjunto de APIs ('*Application Program Interface*') homogéneas que son independientes de la red subyacente y la versión de Java. Por tanto, en teoría, los desarrolladores de aplicaciones podrían decidir el entorno Java a utilizar (p.ej. J2EE, J2SE, J2ME) al desplegar el sistema.
- Facilidad de uso: la complejidad del middleware está oculta tras un conjunto de APIs simples e intuitivas.
- Filosofía de 'pago por uso': los programadores no necesitan utilizar todas las características disponibles a través de JADE. Las características que no son utilizadas, no deben ser conocidas por los programadores ni suponen ningún coste computacional.

JADE incluye tres elementos fundamentales (Bellifemine et al., 2003; Caire, 2003): (1) un conjunto de paquetes (esto es, clases Java) que proporcionan una serie de componentes que dan soporte a los desarrolladores a la hora de crear los agentes; (2) un entorno de ejecución que aporta los servicios básicos necesarios para la ejecución de los agentes; y (3) un conjunto de herramientas gráficas que permiten administrar y monitorizar la actividad de los agentes activos en el sistema. Cada instancia del entorno de ejecución se denomina *contenedor,* y ofrece una serie de servicios de acuerdo con el modelo de referencia FIPA, como el registro, el sistema de intercambio de mensaje o el servicio de autentificación. En un mismo entorno o *plataforma*, pueden subsistir, al mismo tiempo, numerosos *contenedores*. En estos casos, toda plataforma deberá disponer de un *contenedor principal* en el que el resto de *contenedores* de esta plataforma se han de registrar (Fig. 7).

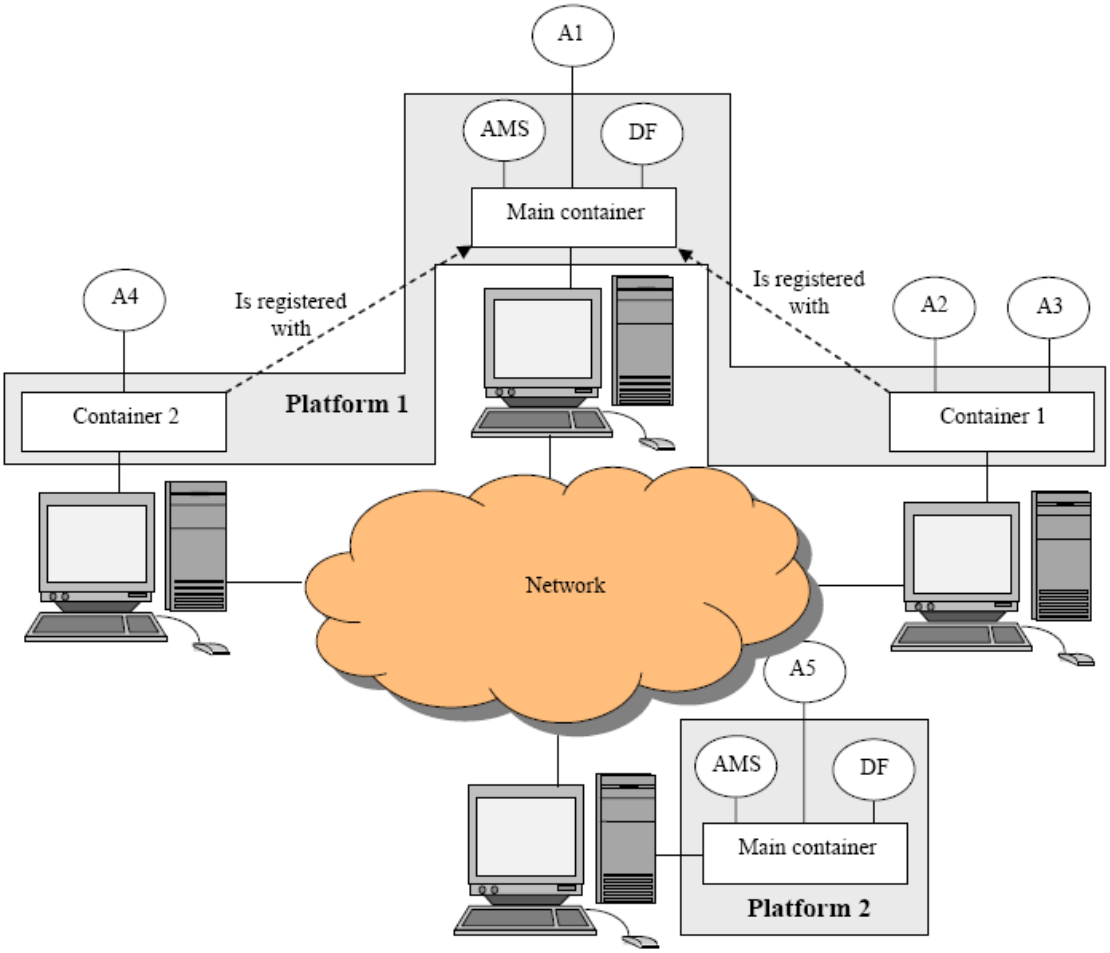


Fig. 7. Plataformas y contenedores en JADE (Caire, 2003)

Como se puede comprobar en la figura anterior, el *contenedor principal* alberga dos agentes específicos que se arrancan de forma automática cuando el contenedor es lanzado:

- AMS (*Agent Management System*). Proporciona un servicio de páginas blancas, asegurándose de que todo agente en la plataforma tiene un nombre único. Además, este agente representa la autoridad en la plataforma, supervisando el acceso y el uso de la misma.
- DF (*Directory Facilitator*). Es un servicio de páginas amarillas, por medio del cual un agente puede encontrar otros agentes a partir de los servicios que requiere para alcanzar un determinado objetivo.

Además, cuando arranca JADE, también se prepara un *canal de comunicación entre agentes* (ACC, '*Agent Communication Channel*') que controla el intercambio de mensajes, tanto dentro como fuera de la plataforma.

Desde el punto de vista funcional (Bellifemine et al., 2003), JADE proporciona los servicios básicos necesarios para el desarrollo de aplicaciones distribuidas peer-to-peer, tanto en entornos fijos como móviles. En este sentido, permite a los agentes descubrir dinámicamente a otros agentes y comunicarse con ellos conforme al paradigma peer-to-peer. Cada agente viene identificado por un nombre único, y suministra un conjunto de servicios. Así, los agentes pueden registrar y modificar sus servicios, además de buscar agentes que dispongan de unos determinados servicios. La comunicación entre los agentes se produce a través de mensajes asíncronos, permitiendo, de este modo, un modelo de comunicación distribuido y débilmente acoplado. Para que se produzca una comunicación efectiva entre los agentes, éstos deben de compartir el mismo *lenguaje*, el mismo *vocabulario* y los mismos *protocolos*. Dado que JADE sigue el estándar propuesto por FIPA, tanto los actos comunicativos especificados en esta normativa (que constituyen los distintos protocolos posibles) como el lenguaje de contenido (SL-0), son comunes para todos los agentes. Sin embargo, es necesaria, la definición de un vocabulario y la semántica del mismo, esto es, una **ontología**, para que la comunicación entre los agentes se pueda producir con éxito. En realidad, en JADE se contemplan tres métodos distintos para la definición del contenido de los mensajes:

1. El modo más básico posible consiste en el uso de cadenas de texto para representar este contenido. Esta técnica es conveniente cuando el contenido de los mensajes son datos atómicos. En cambio, no es adecuado utilizar cadenas de texto cuando se tratan conceptos abstractos, objetos o datos estructurados.
2. El segundo modo de especificar el contenido de los mensajes es a través de objetos Java *serializables*. El problema de esta técnica radica en que se limita la aplicación al hecho de que todos los agentes deben estar implementados en Java. Otro inconveniente es que estos mensajes no pueden ser leídos por humanos.
3. El tercer método contempla el uso de ontologías y consiste en la definición de los objetos a ser transmitidos como extensión de clases predefinidas por JADE, de forma que la plataforma sea capaz de codificar y decodificar los mensajes en un formato estándar FIPA. En JADE, una ontología está compuesta por dos partes, un vocabulario que describe la terminología de conceptos usados por agentes en sus espacios de comunicación y la nomenclatura de las relaciones entre esos conceptos, que describen su semántica y estructura. Se puede implementar una ontología extendiendo la clase *Ontology* predefinida en JADE y añadiendo un

conjunto de elementos describiendo la estructura de los conceptos, acciones y predicados permitidos en el contenido de los mensajes. También es posible crear una ontología extendiendo las clases de ontologías básicas *BasicOntology* y *ACLOntology*. En general, esta técnica es la más conveniente, ya que permite comunicar JADE con otras plataformas multi-agente.

A modo de resumen, en Bellifemine et al. (2006, p. 6) se identifican las siguientes características básicas que el entorno de desarrollo JADE proporciona a los programadores:

- Plataforma de agentes *distribuida* que puede separarse entre varios servidores. Sólo una aplicación Java y, por tanto, sólo una máquina virtual de Java, es ejecutable en cada servidor. Los agentes se implementan como hilos Java que habitan en contenedores de agentes, que proporcionan los elementos necesarios para permitir la ejecución de los agentes.
- *Interfaz gráfica de usuario* para gestionar los agentes y los contenedores desde un servidor remoto.
- Herramientas para *encontrar y eliminar errores* en el desarrollo del SMA.
- *Movilidad* de los agentes entre plataformas, incluyendo la transferencia tanto del estado como del código (en caso de ser necesario) del agente.
- Soporte a la ejecución de actividades múltiples, paralelas y concurrentes de agentes a través de un modelo de *comportamientos*.
- Plataforma de agentes que cumple con el *estándar FIPA*, que incluye el *AMS*, el *DF* y el *ACC*. Estos tres componentes se activan de forma automática en cuanto se arranca la plataforma.
- Numerosos DF compatibles con FIPA pueden ser arrancados durante tiempo de ejecución para permitir aplicaciones multi-dominio, donde un dominio consiste en un conjunto lógico de agentes cuyos servicios se publicitan a través de un *facilitador* (DF) común.
- Transporte eficiente de mensajes ACL dentro de la misma plataforma de agentes. Cuando un mensaje va de una plataforma a otra, éste se convierte a, y desde, la sintaxis, codificación y protocolo de transporte compatible con FIPA.
- Librerías con la implementación de algunos de los protocolos de interacción definidos en el modelo de referencia de FIPA.

- Automatización de procesos de inserción de registros y cancelación de los mismos en el AMS.
- Servicio de nombres compatible con FIPA: al iniciarse, los agentes obtienen su GUID (*Global Unique Identifier*, Identificador Global Único) de la plataforma.
- Se ofrece soporte a lenguajes de contenidos y ontologías dependientes de la aplicación.
- Interfaz "*InProcess*", que permite a aplicaciones externas lanzar agentes autónomos.

Debido a que JADE es un software de código abierto bajo licencia LGPL (*Lesser General Public License Version 2* ó Licencia Pública General Menor), ha sido objeto de actualizaciones y refinamiento por parte de numerosos investigadores tanto del sector industrial como educativo y, en la actualidad, se están llevando a cabo numerosos proyectos basados en esta plataforma. Esto ha hecho que la herramienta haya madurado considerablemente durante los últimos años, siendo la última versión oficial la número 3.4.1. Este dato, unido al hecho de que JADE es la herramienta más utilizada en los proyectos de investigación relacionados con la tecnología de agentes, ha constituido uno de los motivos principales por los que se ha elegido esta plataforma multi-agente para el desarrollo del marco de trabajo elaborado como parte de esta tesis.

I.3.5.3. Otras

Las otras dos herramientas que constituyen los máximos exponentes en lo que respecta a plataformas para la elaboración de SMA, y que satisfacen los requisitos y pautas establecidas por la organización para la elaboración de estándares FIPA, son ZEUS y FIPA-OS. A continuación, se presenta una breve descripción de estas dos plataformas.

FIPA-OS[4], acrónimo de FIPA Open Source (FIPA-OS, 2001), es un marco de trabajo de agentes desarrollado con el fin de construir plataformas multi-agente heterogéneas, agentes y servicios que se ajusten a los estándares de FIPA. FIPA-OS, además de incluir los componentes preceptivos del modelo de referencia FIPA (AMS, DF, ACC e IPMT), ofrece diferentes tipos de soporte, destacando los siguientes: (1) Diferentes tipos de shells de agentes que permiten implementar agentes que pueden comunicarse entre sí usando las facilidades de FIPA-OS (clases Java con puntos de extensión). (2) Soporte

[4] http://sourceforge.net/projects/fipa-os/

multi-capa para la comunicación de agentes (conversación, mensaje ACL, contenido –sintaxis– y ontología –contenido semántico–). (3) Gestión de conversación y mensajes (puntos de extensión dentro del ACL). (4) Configuración dinámica de la plataforma. (5) Patrones de diseño de software e interfaces abstractas. (6) Herramientas de diagnóstico y visualización. FIPA-OS, además de la ventaja de ser un sistema abierto en términos de licencia de uso y extensibilidad, al igual que ZEUS y JADE, presenta una serie de características adicionales que merecen ser enumeradas: (1) interoperabilidad probada con otras plataformas de agentes FIPA no desarrolladas con FIPA-OS; (2) permite heterogeneidad de agentes, esto es, agentes implementados con diferentes lenguajes de programación; (3) soporta la especificación FIPA para la gestión de agentes; y (4) proporciona abstracciones e interfaces (APIs) que permiten extender e integrar una plataforma de agentes con otras plataformas software ya existentes. El mayor inconveniente de FIPA-OS es que la última versión conocida (2.2.0) es de Marzo de 2003 y, desde entonces, no se han producido nuevos cambios en la aplicación y el desarrollo ha sido discontinuo.

ZEUS[5] (Nwana et al., 1999) aglutina un conjunto de herramientas (*toolkit*) desarrolladas con el objetivo de facilitar el proceso ingenieril relacionado con la implementación de aplicaciones de agentes colaborativos. Como se mencionó anteriormente, ZEUS consiste en una metodología de desarrollo de aplicaciones multi-agente que proporciona un entorno visual que permite capturar la especificación de los usuarios acerca de los agentes necesarios, lo cual es posteriormente utilizado para generar el código fuente en Java de estos agentes. Los elementos que constituyen ZEUS se agrupan en tres grandes grupos funcionales: la librería de componentes de agentes, la herramienta de desarrollo de agentes y la suite de agentes básicos. La librería de componentes de agentes es una colección de clases que constituyen los bloques constitutivos de los agentes individuales. En su conjunto, estas clases implementan los aspectos y funcionalidades requeridas para elaborar un SMA y que son independientes de la aplicación a elaborar (p.ej. comunicación, planificación y programación, interacción social, interfaz de usuario y estructuras de datos).

Por su parte, la herramienta de desarrollo de agentes constituye el elemento clave para posibilitar un desarrollo rápido y eficiente de los SMA. Incluye editores visuales que

[5] http://sourceforge.net/projects/zeusagent

permiten generar los diagramas correspondientes especificados por la metodología ZEUS, así como generadores de código capaces de producir la implementación de los agentes a partir de los diagramas visuales. Además, esta herramienta incorpora diversas APIs para facilitar el uso de sistemas heredados (*legacy systems*). Por último, la suite de agentes básicos incorpora el conjunto mínimo de agentes preciso para cumplir el estándar FIPA (agente "servidor de nombres" y agente "facilitador"), además del agente "visualizador" encargado de encontrar y eliminar posibles errores entre los agentes ZEUS. Actualmente, y desde el 10 de Enero de 2006, se encuentra disponible la versión 2.0 de este toolkit. En esta última versión, se ha incluido soporte a descripciones semánticas de servicios mediante el uso de DAML-S (predecesor de OWL-S). Además, se pueden generar ontologías definidas en lenguajes como OIL, aunque, hasta el momento, no se pueden utilizar estas estructuras para definir los modelos de información de los agentes. Finalmente, también se habilita el paso de mensajes mediante SOAP usando la librería AXIS de Servicios Web (en la siguiente sección se ofrecerán más detalles relacionados con los Servicios Web).

I.3.6. Problemas y Carencias de los Sistemas Multi-Agente

Como se ha visto en detalle en los apartados anteriores, la tecnología de agentes en general, y los SMAs en particular, ofrecen unas características muy 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 (Elamy, 2005). Por su parte, los agentes inteligentes añaden a este tipo de sistemas propiedades como la autonomía, proactividad y comportamiento dirigido por objetivos, que permiten el desarrollo de numerosas actividades sin la necesidad de la intervención humana. Adicionalmente, estos agentes pueden incorporar habilidades como el aprendizaje y el razonamiento que, junto a la aplicación de técnicas de Inteligencia Artificial, permiten mostrar un comportamiento inteligente y realizar las tareas de un modo óptimo.

Sin embargo, y a pesar de las numerosas ventajas que supone la aplicación de esta tecnología, no son muchas las aplicaciones reales desarrolladas que exploten todo el potencial de la misma. En realidad, este hecho viene justificado por la existencia de numerosos problemas que limitan la utilidad y aplicabilidad de este tipo de sistemas en entornos reales. El primer y mayor inconveniente de los SMAs radica en la comunicación entre agentes situados en diversas plataformas localizadas en servidores

remotos. El problema surge cuando un mensaje debe de atravesar las fronteras de una compañía para llegar a su destino. En este punto, y dado que las plataformas multi-agente hacen uso de protocolos propietarios no estándar, como RMI (*Remote Method Invocation*) de Java o el protocolo propuesto por la OMG para CORBA, IIOP (*Internet Inter-Orb Protocol*), en la mayoría de los casos, los mensajes no son capaces de llegar a su destino, ya que son rechazados por los cortafuegos de entrada a los sistemas de la otra compañía. De hecho, los problemas asociados al uso de protocolos no estándar en tecnologías como CORBA, RMI o DCOM propició el origen de los Servicios Web que, como se detallará en secciones posteriores, hacen uso de protocolos estándar como HTTP que evita este tipo de situaciones. En el caso particular de las plataformas multi-agente, se diseñó una solución consistente en la creación de los denominados *puentes de comunicación* o *gateways*. De esta forma, las plataformas que se encuentren localizadas en servidores de distintas compañías pero que tengan disponible entre ellas un *Gateway*, serán capaces de comunicarse sin temor a que sus mensajes se pierdan por la acción de un cortafuegos. Desgraciadamente, los gateways entre plataformas multi-agente deben de establecerse al tiempo que se arrancan los sistemas y, por tanto, no es posible la comunicación entre agentes situados en plataformas de compañías distintas que no hayan establecido previamente un gateway. Esto implica, consecuentemente, la imposibilidad de crear vínculos dinámicos entre compañías que previamente no habían contemplado esta posibilidad. Este hecho se reafirma en el trabajo de Louis y Martinez (2005) donde se afirma que, en casi todos los casos, los agentes sólo pueden interactuar con aquellos agentes que en tiempo de diseño les han sido especificados.

Otra dificultad añadida a la que se deben enfrentar los SMAs está relacionada con la semántica en 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 la especificación FIPA y satisfacen el estándar, es capaz de proporcionar el soporte adecuado para manejar esta dimensión semántica del lenguaje FIPA-ACL (Louis y Martinez, 2005). De esta forma, 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 entran en un proceso de negociación.

En general, la mayor parte de los investigadores en Inteligencia Artificial y tecnología de agentes reconoce 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, en primer lugar, afrontar los numerosos retos que se plantean (Ana Mas, 2005, pp.25-27):

- Obtener una definición "ingenieril" del concepto de agente: 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 (p.ej., procedimientos o clases) 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 relacionadas: 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, etc.).
- 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.
- Producir aplicaciones BURI (Baratas, Útiles, Robustas e 'Inteligentes').

I.4. Servicios Web Semánticos

Los Servicios Web Semánticos son la evolución de la popular tecnología de los Servicios Web a los que se les aplica los conceptos introducidos por la Web Semántica. En esta sección se describen, en primer lugar, las partes constituyentes de la tecnología de los Servicios Web Semánticos, esto es la Web Semántica y los Servicios Web. A continuación, se presenta una descripción detallada de las principales características de esta tecnología y se enumeran las distintas aproximaciones propuestas al W3C para alcanzar un estándar. Finalmente, se discuten algunas de las carencias y problemas de esta tecnología que, en cierto modo, justifica la idoneidad de integrar los Servicios Web Semánticos con la tecnología de agentes.

I.4.1. La Web Semántica

1.4.1.1. Antecedentes

La *World Wide Web* (WWW) fue concebida por Tim Berners-Lee en 1989 a partir de un proyecto del CERN (Consejo Europeo para la Investigación Nuclear) y cambió radicalmente el modo en que la gente recopila información y accede a ésta. Hoy en día, la WWW se ha convertido en un gigantesco repositorio de información en continuo crecimiento y al que se puede acceder desde cualquier punto con el único requisito de disponer de un navegador Web.

A día de hoy más de 14 billones de páginas[6] se encuentran indexadas por los distintos buscadores existentes, de forma que la búsqueda de un dato concreto se ha convertido en uno de los mayores cuellos de botella de la WWW. El ejemplo típico que se utiliza para presentar este problema y demostrar las carencias de la Web actual es el de la búsqueda de información concerniente a la *isla de Java*. Si en un buscador cualquiera, como por ejemplo en Google[7], escribimos la consulta "*Java*" nos aparecen cientos de miles de páginas que nada tienen que ver con lo que el usuario está buscando realmente, la isla de Java, sino que la mayoría se referirán al lenguaje de programación *JavaTM*. Este ejemplo, sin embargo, no demuestra las verdaderas carencias de la WWW ni de las tecnologías detrás de ésta, en particular el lenguaje HTML (lenguaje de marcas hipertextuales), ya que el usuario sólo tendría que refinar la búsqueda con la consulta "*la isla de Java*" y, ahora sí, encontraría todos aquellos documentos y páginas Web en los que puede estar interesado. El problema surge cuando se trata de obtener información sobre una relación concreta entre dos conceptos. Como ejemplo, si realizamos la consulta "*artículos sobre Tim Berners-Lee*" muchos de los resultados que obtendremos serán enlaces a artículos escritos por Tim Berners-Lee. De forma resumida, la principal limitación de la Web actual es el hecho de que las tecnologías que la conforman no son capaces de capturar, o representar formalmente, la semántica del contenido presentado. La Web Semántica surgió con el propósito de restringir estos efectos nocivos causados por el crecimiento del número de páginas publicadas en la WWW.

[6] http://www.worldwidewebsize.com/

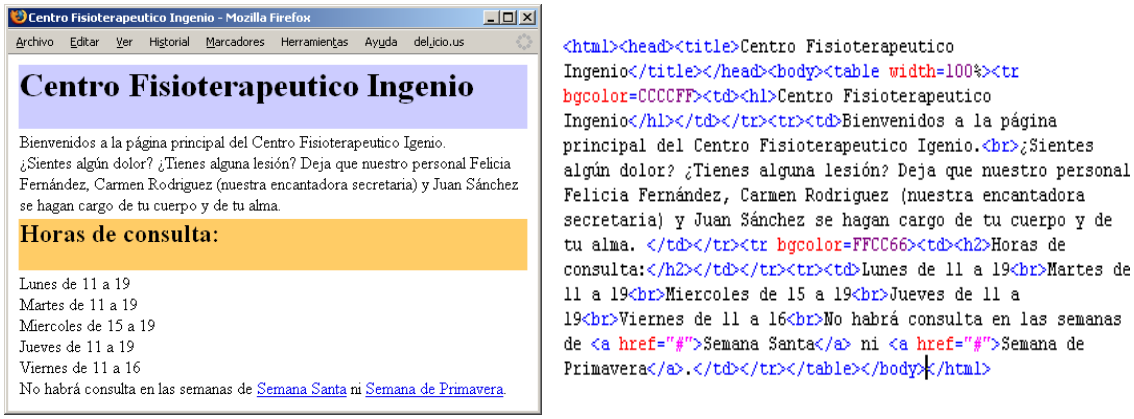
[7] http://www.google.com

1.4.1.2. Fundamentos de la Web Semántica

El término '*Web Semántica*' fue presentado al dominio público tras la publicación del artículo "The Semantic Web" aparecido en Scientific American en Mayo de 2001 y del que fueron coautores Tim Berners-Lee, James Hendler y Ora Lassila (Berners-Lee et al., 2001). En este artículo se establecen diversos escenarios imaginarios en los que agentes software son capaces de realizar numerosas tareas accediendo al contenido de diferentes páginas de la WWW. Los autores señalan que, para que este escenario sea factible, debería cambiar la manera de representar contenido en la Web (hasta ahora diseñado para que los seres humanos puedan leerlo) para incluir una "semántica bien definida" que permitiese a componentes software acceder al mismo.

> *"La Web Semántica añadirá estructura al contenido lleno de significado de las páginas Web, creando un entorno donde agentes software, transitando de página en página, puedan llevar a cabo fácilmente tareas sofisticadas para los usuarios."*

La Web Semántica se basa, por tanto, en añadir *metadatos* semánticos a los datos publicados en la WWW de forma que las entidades software puedan procesar el contenido de forma similar a como lo hacen los humanos. El término *metadato* se refiere a ofrecer datos sobre datos, de forma que el significado de los datos es capturado. En la siguiente figura (Fig. 8) se muestra cómo es una página Web de ejemplo en la actualidad (basado en Antoniou & van Harmelen, 2003, pp. 8-9):



a) La Web vista por una persona b) La Web vista por el ordenador

Fig. 8. Página HTML en la Web actual

Una entidad software puede hacer búsquedas basadas en palabras clave que identifiquen palabras como 'fisioterapéutico', 'horas de consulta', etc., pero tendrá problemas para identificar al personal y distinguir entre los fisioterapeutas y la

secretaria. La Web Semántica propone modificar la forma en que se presentan los contenidos en la Web de modo que no sólo se indique información para formatear el contenido, sino que también se incluya información que describa este contenido. Siguiendo nuestro ejemplo, el código quedaría como se presenta en la siguiente figura (Fig. 9). Esta representación mediante metadatos hace mucho más sencillo para las máquinas el procesamiento de la información.

```
<empresa>
   <nombre>Centro Fisioterapeutico Ingenio</nombre>
   <tratamientoOfrecido>Fisioterapia</tratamientoOfrecido>
   <personal>
      <terapeuta>Felicia Fernandez</terapeuta>
      <terapeuta>Juan Sanchez</terapeuta>
      <secretaria>Carmen Rodriguez</secretaria>
   </personal>
   <horario>
   ...
   </horario>
</empresa>
```

Fig. 9. Página Web con metadatos

En la siguiente figura (Fig. 10) se puede comprobar la evolución que ha tenido lugar en cuanto a la estructuración de los contenidos en la Web tradicional y en la Web Semántica. Si la conexión entre recursos en la Web tradicional se produce a través de enlaces únicamente entendibles por usuarios humanos (aunque no en todos los casos), la estructura de la Web Semántica se basa en relaciones semánticas que son capaces de interpretar tanto humanos como entidades software.

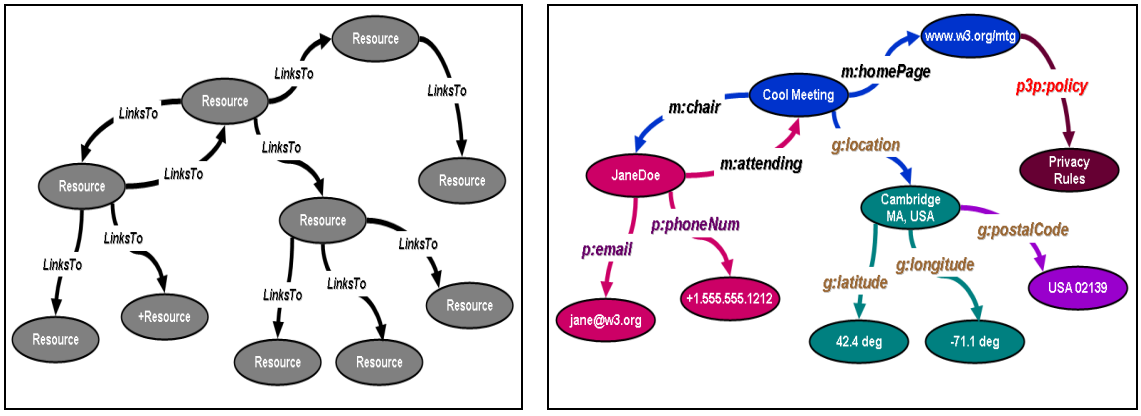


Fig. 10. Evolución de la estructura de la Web tradicional a la estructura de la Web Semántica (W3C Oficina Española, 2006)

Esta nueva aproximación de la Web Semántica a la estructuración de la información se basa en la utilización de ontologías como mecanismo para la representación del

conocimiento. En este entorno, las ontologías se utilizan para aportar un vocabulario que describa las relaciones entre diferentes términos, permitiendo a los sistemas computerizados (y a los humanos) interpretar su contenido flexiblemente y sin ambigüedades (Horrocks et al., 2003). Se puede decir que una ontología, en el contexto de la Web, describe formalmente un dominio del discurso, facilitando un entendimiento compartido (Antoniou & van Harmelen, 2003, p. 10). En general, una ontología contiene una lista finita de términos y relaciones entre esos términos. Los términos denotan conceptos (esto es, clases de objetos) importantes del dominio. En el ámbito de la universidad, por ejemplo, estudiantes, profesores, facultades, aulas, clases, etc. serían algunos de los conceptos a utilizar. Las relaciones incluyen generalmente jerarquía de clases. Además de este tipo de relaciones, las ontologías incluyen propiedades (el profesor X enseña la asignatura Y), restricciones de valor (sólo profesores pueden enseñar asignaturas), enunciados disyuntivos (profesores y personal de administración y servicios son disjuntos), relaciones lógicas entre objetos (cada departamento debe componerse de, al menos, 10 profesores), etc. Como ya fue resaltado anteriormente, las ontologías tienen un fundamento formal por lo que los lenguajes ontológicos como OWL están basados en formalismos lógicos derivados, en su mayoría, de la lógica de primer orden (una descripción más detallada del término ontología se encuentra en la sección I.2).

1.4.1.3. Arquitectura de la Web Semántica

El desarrollo de la Web Semántica se está produciendo en forma de capas, de modo que el desarrollo de nuevas capas se realice sobre las ya existentes. En la actualidad, el W3C maneja una estructura en lo que respecta a la Web Semántica que no dista mucho de lo que estableciera Tim Berners-Lee en el año 2000 (Fig. 11).

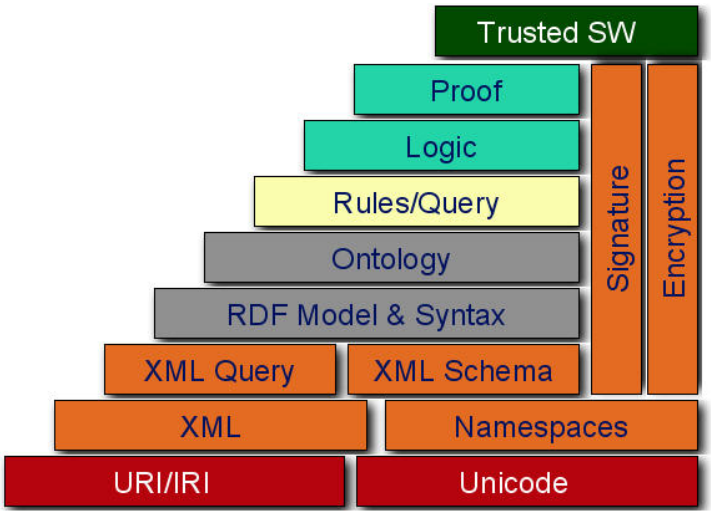


Fig. 11. Estructura en forma de pastel de la Web Semántica para el W3C

De forma resumida, a continuación se presentan las características de las primeras capas de esta estructura (Antoniou & van Harmelen, 2003):

- XML aporta la sintaxis superficial para los documentos estructurados, pero sin dotarles de ninguna restricción sobre el significado.
- XML Schema es un lenguaje para definir la estructura de los documentos XML.
- RDF es un modelo de datos para los recursos y las relaciones que se puedan establecer entre ellos. Aporta una semántica básica para este modelo de datos que puede representarse mediante XML.
- RDF Schema es un vocabulario para describir las propiedades y las clases de los recursos RDF, con una semántica para establecer jerarquías de generalización entre dichas propiedades y clases.
- OWL, finalmente, añade más vocabulario para describir propiedades y clases. Entre otras mejoras en expresividad, se encuentra la posibilidad de definir relaciones entre clases, cardinalidad, igualdad, tipologías de propiedades más complejas, caracterización de propiedades o clases enumeradas.

1.4.1.4. Ventajas e Inconvenientes

Entre las ventajas que pretende aportar la Web Semántica podemos destacar las siguientes:

- Ahorro de tiempo en el procesado de los datos (tiempo de búsqueda, gestión de la información, etc.): la mayor parte de las tareas relativas al procesado de datos

podrán realizarse por parte de componentes software automatizados y sin requerir, en muchos casos, intervención humana.

- Resultados más adecuados de búsqueda: se mejorará la precisión de las búsquedas en Web, debido a que el contenido de las páginas Web estará anotado semánticamente y, por tanto, los motores de búsqueda serán capaces de obtener aquellos resultados que más se adecuen a la consulta del usuario. En particular, los motores de búsqueda podrán buscar páginas que se refieran a un determinado concepto de una ontología, al contrario de lo que ocurre ahora, devolviendo todas aquellas páginas que contienen una palabra clave que, en muchos casos, es ambigua.
- Mejora de la comunicación entre servicios Web: la comunicación entre distintos componentes y servicios, sobre todo en aquellos casos en los que los componentes no han sido diseñados para trabajar conjuntamente, ha sido siempre fuente de problemas en cuanto a la interoperabilidad debido, principalmente, a la ambigüedad del lenguaje. El uso de ontologías compartidas (y, posiblemente, mapeadas con otras ontologías) propuesto por la Web Semántica, solventa el problema en la comunicación entre servicios Web gracias a que esta comunicación se produce utilizando conceptos pertenecientes a una ontología.

La Web Semántica mantiene los principios que han contribuido al éxito de la Web actual, como son los principios de descentralización, compartición, compatibilidad, máxima facilidad de acceso y contribución, o la apertura al crecimiento y uso no previstos de antemano (Castells, 2003). Sin embargo, existen ciertos problemas que deben solucionarse para alcanzar todo el potencial que se le supone a la Web Semántica. En primer lugar, la aparición de la Web Semántica supondrá mayor trabajo para los creadores de páginas Web ya que éstas deberán estar anotadas semánticamente. Por tanto, es crítico el desarrollo de herramientas que permitan a usuarios no experimentados la creación de páginas para la nueva Web Semántica con la misma facilidad con la que éstos lo pueden hacer para la Web tradicional. Otro de los inconvenientes de la Web Semántica son los efectos que puede producir en relación a la privacidad y la censura. En la actualidad, las técnicas de análisis de textos utilizadas por los gobiernos para controlar los contenidos de la Web son vulnerables a simples modificaciones de palabras (usando, por ejemplo, metáforas) o al uso de imágenes en vez de texto consiguiendo, de esta forma, limitar las posibilidades de censura. En la

Web Semántica sería mucho más sencillo controlar la creación y visualización de contenidos Web. Además, la aplicación de propuestas como FOAF (*Friend of a Friend*)[8], que permite la definición semántica de perfiles de usuario y redes sociales, supondría una limitación severa al anonimato en la Web. Si los investigadores en Web Semántica son capaces de eliminar esta problemática, no cabe duda de que esta tecnología supondrá una clara evolución con respecto a la Web tradicional, transformándose en la nueva Web 3.0[9].

I.4.2. Servicios Web

Los Servicios Web proporcionan un mecanismo estándar para que diferentes aplicaciones software que están siendo ejecutadas en diferentes plataformas y/o marcos de trabajo puedan interoperar (Booth et al., 2004). En (Booth et al., 2004) se propone la siguiente definición de Servicio Web, que coincide con la proporcionada por el glosario de la actividad "Web Services Activity" del W3C:

> *"Un Servicio Web es un sistema software designado para soportar interacciones interoperables de máquina a máquina sobre una red de comunicaciones. Un Servicio Web tiene una interfaz descrita en un formato procesable por el ordenador (en particular WSDL). Otros sistemas pueden interaccionar con el Servicios Web de la manera prescrita en su descripción usando mensajes SOAP, generalmente transmitidos mediante el uso del protocolo HTTP con una serialización en XML en conjunción con otros estándares relacionados con la Web"*

La tecnología de los Servicios Web no es, sin embargo, una idea completamente novedosa. El origen de esta tecnología, en cuanto a principios se refiere, bien se podría remontar a los años 70 del siglo XX con el surgimiento de Internet y la computación distribuida.

En esta sección, se estudian los antecedentes de lo que hoy conocemos como Servicios Web y los componentes que forman la arquitectura actual de dichos servicios. Además, se explican los fundamentos de la Arquitectura Orientada a Servicios (SOA) y

[8] http://www.foaf-project.org/

[9] "El término Web 2.0 fue acuñado por O'Reilly Media en 2004 para referirse a una segunda generación de Web basada en comunidades de usuarios y una gama especial de servicios, como las redes sociales, los blogs, los wikis o las folksonomías, que fomentan la colaboración y el intercambio ágil de información entre los usuarios." (fuente: http://es.wikipedia.org/wiki/Web_2.0)

la programación SOA, un nuevo paradigma de programación basado en el uso de componentes débilmente acoplados como los Servicios Web. Finalmente, se indicarán algunas de las ventajas y los inconvenientes del uso de esta tecnología.

1.4.2.1. Antecedentes

La computación distribuida comenzó alrededor de 1970 con el origen de dos tecnologías (Acton, 2004):

- Minicomputadores (posteriormente denominadas estaciones de trabajo y después ordenadores personales o PCs).
- Redes de ordenadores (ethernet e Internet).

Un sistema distribuido consiste en entidades software heterogéneas y discretas que deben trabajar de forma conjunta para llevar a cabo algunas tareas (Booth et al., 2004). Además, las entidades software en un sistema distribuido no operan en el mismo entorno de procesamiento, de forma que deben de comunicarse a través de pilas de protocolos hardware/software sobre una red de comunicaciones. Los sistemas de objetos distribuidos son sistemas en los que la semántica de la inicialización de un objeto y la de la invocación de los métodos se expone a sistemas remotos por medio de mecanismos estándares o propietarios que permiten intermediar en las peticiones que salen de los límites del sistema.

Numerosas son las aproximaciones que se han realizado que permiten la construcción de sistemas distribuidos. En este apartado, se destacan las más reconocidas y aquellas que más han influido para la posterior elaboración de la tecnología de los Servicios Web: RPC, DCOM, RMI y CORBA.

Llamada a Procedimiento Remoto (RPC). Es un protocolo que permite a un programa ejecutar una subrutina o procedimiento en otra máquina remota sin necesidad, por parte del programador, de codificar los detalles de la interacción remota (ver Fig. 12). Esta tecnología, que data de mediados de los años 70, supuso un gran avance en el desarrollo de sistemas dentro del paradigma cliente-servidor. Existen numerosas implementaciones de este protocolo como el ONC RPC de Sun, el DCE/RPC de la '*Open Software Foundation*' (OSF) y el Modelo de Objetos de Componentes Distribuidos (DCOM) de Microsoft. Estas implementaciones no son, en su mayoría,

compatibles entre sí. Las soluciones más populares basadas en objetos (esto es, objetos distribuidos) previas a los Servicios Web se describen brevemente a continuación.

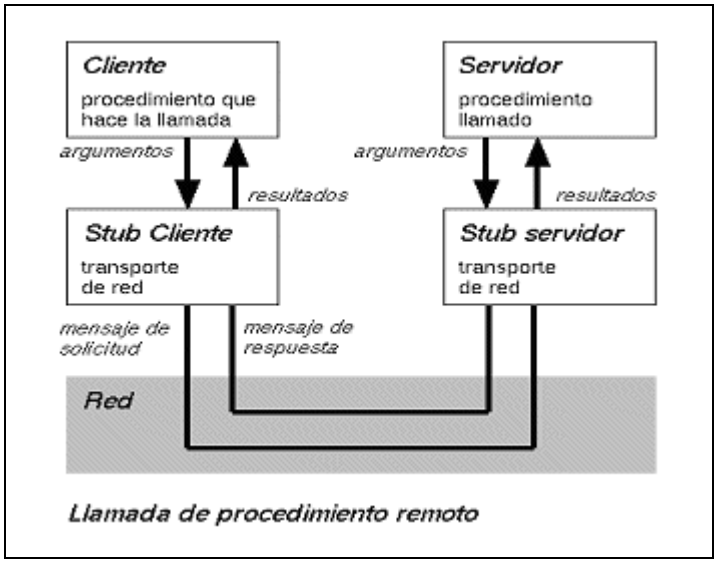


Fig. 12. Modelo de funcionamiento de RPC

Modelo de Objetos de Componentes Distribuidos (DCOM). Es una tecnología propietaria de Microsoft para desarrollar componentes distribuidos sobre varios ordenadores y que se comunican entre sí. La base de DCOM es el Microsoft RPC (MSRPC), una extensión de DCE/RPC de la OSF. Si bien se ha trabajado en la implementación de DCOM para su compatibilidad con sistemas UNIX, esta tecnología fue ideada para su uso sobre el sistema operativo Microsoft. DCOM soporta varios lenguajes de programación, como Visual Basic, C y C++. En la actualidad, la tecnología DCOM ha sido abandonada en favor del *framework .NET*.

Invocación de Métodos Remotos (RMI). Es un mecanismo ofrecido en Java para invocar un método remotamente. Al estar integrado en Java, RMI es independiente del sistema operativo pero únicamente soporta el lenguaje de programación Java. La arquitectura de RMI se puede ver como un modelo de cuatro capas (ver Fig. 13). La primera capa es la de aplicación, correspondiendo con la implementación de las aplicaciones cliente y servidor. La capa Proxy es la segunda, también llamada stub-skeleton porque es la que contiene estos componentes que permiten a los desarrolladores elaborar aplicaciones sin tener en cuenta los detalles referentes a la interacción. La tercera capa es la de referencia remota, responsable del manejo de la parte semántica de las invocaciones remotas. Finalmente, la última capa es la de

transporte, encargada de realizar las conexiones necesarias y del manejo del transporte de los datos entre las máquinas.

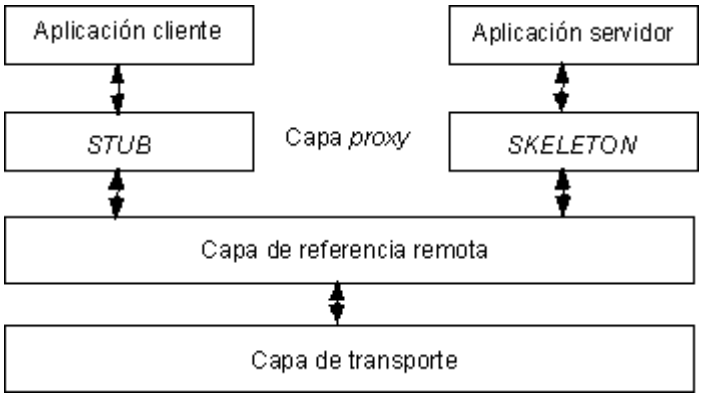


Fig. 13. Arquitectura en capas de RMI

Arquitectura Común de Intermediarios en Peticiones a Objetos (CORBA). El problema de las arquitecturas mencionadas anteriormente, RMI de Sun y DCOM de Microsoft, junto con otras que se desarrollaron de la misma índole (SOM de IBM, OpenDoc de Apple) es que se basan en estándares propietarios, de forma que la interoperabilidad entre ellas es bastante reducida, cuando no nula. Este es el motivo por el cual se desarrolló CORBA, a saber, un estándar que establece una plataforma de desarrollo de sistemas distribuidos, facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos. Fue definido y está controlado por el '*Object Management Group*' (OMG), un consorcio que incluye, entre sus compañías y organizaciones constituyentes más renombradas, a Hewlett-Packard, IBM, Sun Microsystems y Apple Computer. CORBA es independiente del sistema operativo y soporta varios lenguajes, entre ellos, Java, C, C++, Ada, Cobol y Visual Basic.

A pesar de propuestas consensuadas como CORBA, los desarrolladores de software no han sido capaces de alcanzar la interoperabilidad deseada. Esto ha sido debido a la falta de aceptación propiciada por la ausencia de Microsoft en el consorcio que elaboró CORBA (Alesso & Smith, 2005, p.14). Se demostró, por tanto, que para desarrollar estándares aceptados globalmente para la tecnología de componentes distribuidos, era necesario encontrar uno conjunto mínimo de elementos comunes. El lenguaje XML y el protocolo HTTP han sido capaces de ofrecer este conjunto mínimo tecnológico para el éxito de la computación distribuida. Estos elementos constituyen la base de los Servicios Web.

1.4.2.2. Componentes de la Arquitectura

Como se ha mencionado anteriormente, las soluciones previas a la tecnología de los Servicios Web tenían una serie de problemas, principalmente asociados con la limitación en cuanto a la interoperabilidad. Así, mientras unas se restringían a unas plataformas concretas, otras no eran flexibles en cuanto al lenguaje de programación. En general, el uso de soluciones propietarias suponía un problema en la mayoría de los casos porque, en primer lugar, imposibilitaba la interacción entre los elementos elaborados de acuerdo con las distintas soluciones y, lo que es peor, su uso global estaba fuertemente restringido por los cortafuegos (*firewalls*) que bloqueaban los puertos por los que se comunicaban estos componentes. Por tanto, la solución radicaba en encontrar un conjunto de estándares que fueran aceptados mundialmente para construir una tecnología que fuese más acorde a la visión de Internet. XML y HTTP constituyen la base sobre la que se ha desarrollado la tecnología de los Servicios Web.

Fig. 14. Arquitectura de los Servicios Web

La tecnología de los Servicios Web se fundamenta en tres estándares principales (ver Fig. 14): el "*Web Services Description Language*" (WSDL, Lenguaje de Descripción de Servicios Web), el "*Simple Object Access Protocol*" (SOAP, Protocolo Simple de Acceso a Objetos) y el "*Universal Description, Discovery, and Integration*" (UDDI, Descripción, Descubrimiento e Integración Universales). De forma resumida, WSDL es un lenguaje basado en XML que permite describir las características de los Servicios Web; SOAP es un protocolo estándar que define cómo dos objetos localizados

remotamente pueden comunicarse por medio de intercambio de datos XML y que funciona sobre HTTP (posee también enlaces con otros protocolos de Internet); y UDDI es un registro que proporciona mecanismos estándares para publicar documentos WSDL que contienen descripciones de Servicios Web y realizar búsquedas sobre los mismos.

Son cinco los pasos principales que constituyen el *modus operandi* para la utilización de los Servicios Web (ver Fig. 14). En primer lugar, es necesario que la organización o entidad que desee ofrecer un servicio, añada una descripción del mismo (fichero WSDL del servicio) en un registro UDDI, esto es, registre el servicio para su posterior utilización. Esto debe ocurrir para todos los servicios, de todas las organizaciones, que se deseen que sean accesibles globalmente. El siguiente paso, una vez que la descripción de todos los servicios se encuentra almacenada en los registros UDDI accesibles por Internet, consiste en que los clientes consumidores de servicios busquen aquellos que les sean necesarios para alcanzar un objetivo. El proceso de búsqueda lo realiza el propio registro UDDI, que recibe como entrada la funcionalidad esperada del servicio y debe devolver todos aquellos servicios que cumplan con esa funcionalidad. El registro de servicios devuelve al cliente, como respuesta a su petición, una lista con los ficheros de descripción de los servicios (esto es, en WSDL) cuyas capacidades cubran las necesidades del cliente.

Una vez el cliente consumidor de servicio dispone de los ficheros WSDL de todos aquellos servicios a los que el cliente puede acceder para alcanzar un objetivo, aquel debe determinar qué servicio, de entre los de la lista, es el más apropiado de acuerdo con las preferencias del cliente. Cuando el cliente ha seleccionado el servicio más apropiado a ejecutar, realiza la petición al proveedor del servicio a través de un mensaje SOAP. El servicio es entonces ejecutando utilizando los parámetros de entrada indicados en el mensaje SOAP, y el resultado de la ejecución se devuelve al cliente en forma de otro mensaje SOAP, terminando así el proceso de invocación de un Servicio Web.

A continuación, se muestra una descripción más detallada de los elementos que constituyen el núcleo de la tecnología de los Servicios Web: WSDL, UDDI y SOAP.

Web Services Description Language (WSDL)

El lenguaje de descripción de servicios en su versión 2.0 proporciona un modelo y un formato XML para describir Servicios Web (Chinnici et al., 2007). En particular, WSDL 2.0 permite separar la descripción de la funcionalidad abstracta ofrecida por el

servicio de los detalles más concretos relativos al "cómo" y "dónde" obtener dicha funcionalidad. Por tanto, una descripción WSDL está estructurada en dos etapas fundamentales, una concreta y una abstracta. En el nivel abstracto, WSDL 2.0 describe un Servicio Web en términos de los *mensajes* que éste envía y recibe. Los *mensajes* están descritos independientemente del formato utilizado, empleándose para ello un sistema de tipos como XML Schema. Una *operación* ('*operation*') asocia un *patrón de intercambio de mensajes* a uno o más mensajes. Un *patrón de intercambio de mensajes* ('*message exchange pattern*') identifica la secuencia y cardinalidad de los mensajes enviados y/o recibidos, así como a quién se envían y/o de quién se reciben. Una *interfaz* ('*interface*') agrupa las operaciones sin concretar un formato específico. En el nivel concreto, una *vinculación* ('*binding*') especifica los detalles relativos al formato para una o más interfaces. Por su parte, un *punto final* ('*endpoint*') asocia una dirección de red con una vinculación. Finalmente, un *servicio* ('*service*') agrupa los *puntos finales* que implementan una interfaz común.

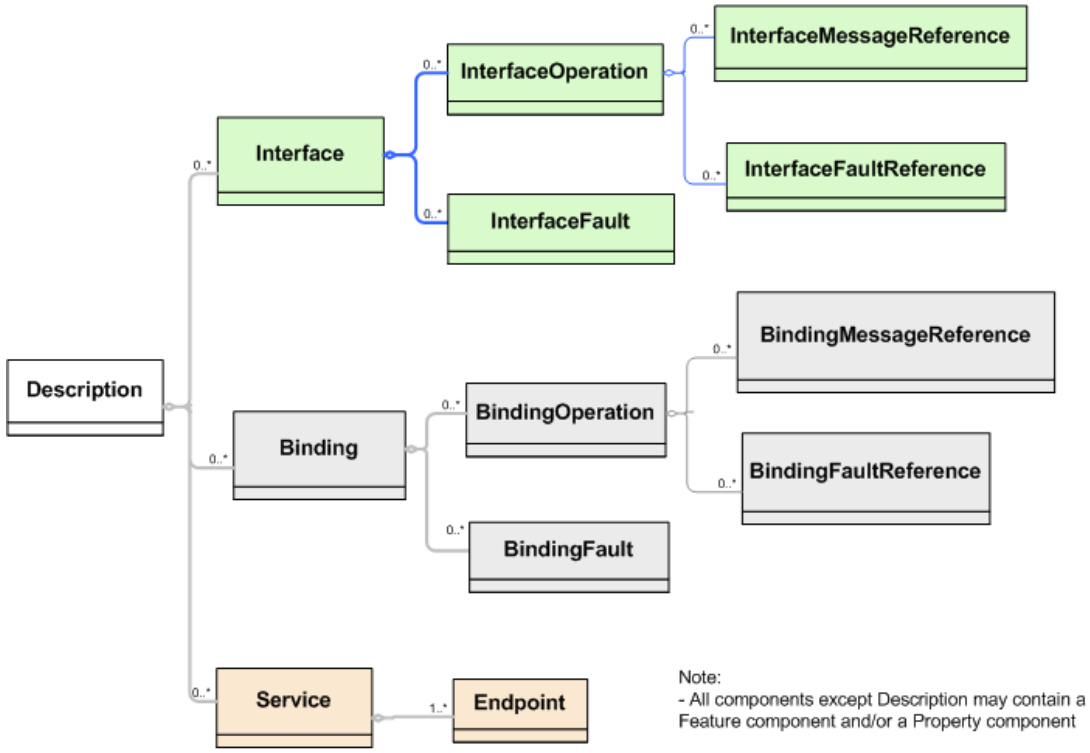


Fig. 15. Modelo de componentes de WSDL 2.0 (Booth & Liu, 2007)

El modelo conceptual de WSDL 2.0 puede verse como un conjunto de componentes a los que se les asocian propiedades y que, colectivamente, describen un Servicio Web. Este modelo es lo que se denomina "Modelo de Componentes" de WSDL 2.0

(ver Fig. 15). *Descripción* (‘*description*’) es el componente contenedor dentro del cual toda la información relativa a un servicio debe incluirse. El componente *descripción* contiene dos categorías de componentes, los componentes de tipos del sistema (declaración de elementos y definiciones de tipos) y los componentes WSDL 2.0 (interfaces, vinculaciones y servicios). Como se puede comprobar parcialmente en la figura anterior (Fig. 15), ‘*interface*’, ‘*binding*’, ‘*service*’, ‘*element declaration*’ y ‘*type definition*’ son los componentes que están directamente contenidos en ‘*description*’, estos son los denominados *top-level components*. Estos componentes pueden a su vez contener otros componentes, denominados componentes anidados.

A continuación se muestra un ejemplo simple de un fichero WSDL describiendo un servicio facilitado por un hotel:

```
<?xml version="1.0" encoding="utf-8" ?>
<description
xmlns="http://www.w3.org/ns/wsdl"
targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsdlx= "http://www.w3.org/ns/wsdl-extensions">
<documentation>
This document describes the GreatH Web service.  Additional
application-level requirements for use of this service—
beyond what WSDL 2.0 is able to describe—are available
at http://greath.example.com/2004/reservation-documentation.html
</documentation>
<types>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://greath.example.com/2004/schemas/resSvc"
xmlns="http://greath.example.com/2004/schemas/resSvc">
<xs:element name="checkAvailability" type="tCheckAvailability"/>
<xs:complexType name="tCheckAvailability">
<xs:sequence>
<xs:element  name="checkInDate" type="xs:date"/>
<xs:element  name="checkOutDate" type="xs:date"/>
<xs:element  name="roomType" type="xs:string"/>
</xs:sequence>
</xs:complexType>
<xs:element name="checkAvailabilityResponse" type="xs:double"/>
<xs:element name="invalidDataError" type="xs:string"/>
</xs:schema>
</types>
<interface  name = "reservationInterface" >
<fault name = "invalidDataFault"
element = "ghns:invalidDataError"/>
<operation name="opCheckAvailability"
pattern="http://www.w3.org/ns/wsdl/in-out"
style="http://www.w3.org/ns/wsdl/style/iri"
wsdlx:safe = "true">
<input messageLabel="In"
element="ghns:checkAvailability" />
```

```
<output messageLabel="Out"
element="ghns:checkAvailabilityResponse" />
<outfault ref="tns:invalidDataFault" messageLabel="Out"/>
</operation>
</interface>
<binding name="reservationSOAPBinding"
interface="tns:reservationInterface"
type="http://www.w3.org/ns/wsdl/soap"
wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
<fault ref="tns:invalidDataFault"
wsoap:code="soap:Sender"/>
<operation ref="tns:opCheckAvailability"
wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
</binding>
<service name="reservationService"
interface="tns:reservationInterface">
<endpoint name="reservationEndpoint"
binding="tns:reservationSOAPBinding"
address ="http://greath.example.com/2004/reservation"/>
</service>
</description>
```

Fig. 16. Fichero WSDL del Servicio Web de un hotel (Booth & Liu, 2007)

Universal Description, Discovery and Integration (UDDI)

UDDI es un protocolo que constituye uno de los bloques fundamentales de la arquitectura de los Servicios Web (Clement et al., 2004). UDDI proporciona una plataforma estándar e interoperable que permite a las compañías y a las aplicaciones encontrar y usar de forma rápida, fácil y dinámica Servicios Web a través de Internet. UDDI es una iniciativa industrial abierta de OASIS ('*Organization for the Advancement of Structured Information Standards*'), un consorcio internacional que abarca a algunas de las organizaciones más importantes a nivel mundial y que se orienta al desarrollo y la adopción de estándares para negocio electrónico y Servicios Web.

Los Servicios Web sólo tienen sentido si existe un mecanismo mediante el cual los usuarios potenciales son capaces de encontrar información suficiente sobre los servicios que les permitan ejecutarlos. UDDI se centra en la definición de un conjunto de servicios para dar soporte a la descripción y descubrimiento de (1) negocios, organizaciones y otros proveedores de Servicios Web, (2) los Servicios Web que éstos hagan disponibles y (3) las interfaces técnicas que pueden ser utilizadas para acceder a dichos servicios.

Los registros UDDI se basan en estándares como HTTP y XML, y consisten en tres partes fundamentales:

- Páginas blancas, que registran información acerca de la dirección, el contacto y otros identificadores conocidos.

- Páginas amarillas, las cuales indican una categorización industrial basada en taxonomías.
- Páginas verdes, que ofrecen información técnica sobre los servicios que aportan las propias empresas.

Esta infraestructura permite, por tanto, el acceso tanto a servicios disponibles públicamente como a aquellos que se exponen únicamente en el interior de las organizaciones.

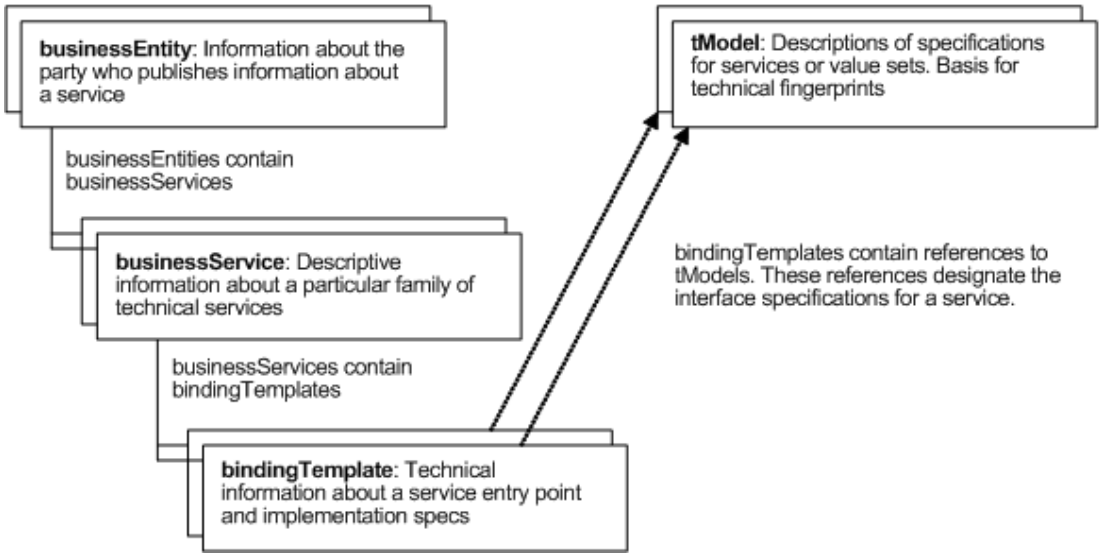


Fig. 17. Estructuras de datos centrales de UDDI (Clement et al., 2004)

La información que forma parte de un registro UDDI consiste en instancias de cuatro tipos de estructuras de datos (ver Fig. 17): la entidad de negocio ('*businessEntity*'), el servicio de negocio ('*businessService*'), la plantilla de vinculación ('*bindingTemplate*') y el '*tModel*'. Cada entidad de negocio contiene información descriptiva acerca de un negocio u organización y, a través de sus relaciones con entidades de servicio de negocio, información acerca de los servicios que ofrece. Una estructura de servicio de negocio representa un servicio lógico, y contiene información descriptiva en términos del negocio. Cada servicio de negocio está relacionado con la entidad de negocio que proporciona el servicio. Por otro lado, la descripción técnica de los Servicios Web se encuentra contenida en entidades de plantilla de vinculación, cada una de las cuales describe una instancia de Servicio Web ofrecida desde una dirección de red particular. La plantilla de vinculación también aporta información acerca del tipo de Servicio Web ofrecido usando referencias a *tModels*, esto es, parámetros específicos de la aplicación y

configuraciones. Finalmente, el propósito de las entidades *tModel* es el de proporcionar un sistema de referencia basado en abstracciones para indicar la conformidad de los servicios a determinadas especificaciones.

De forma resumida, UDDI sirve para dos propósitos fundamentales. En primer lugar, facilita la descripción de los Servicios Web de forma que sean lo suficientemente expresivos para ser útiles en el proceso de búsqueda. El segundo objetivo general de UDDI es aportar las herramientas necesarias para hacer las descripciones lo suficientemente completas para que tanto las personas como los programas puedan descubrir cómo interaccionar con servicios de los que no tenían constancia hasta ese momento.

Simple Object Access Protocol (SOAP)

SOAP es un protocolo ligero que permite intercambiar información estructurada en un entorno descentralizado y distribuido (Gudgin et al., 2007; Mitra & Lafon, 2007). SOAP utiliza tecnologías XML para definir un marco de intercambio de mensajes extensible que permite trabajar sobre múltiples pilas de protocolos. Entre los objetivos de diseño del protocolo, destacan la simplicidad y la extensibilidad. Así, SOAP ha sido desarrollado para ser independiente de cualquier modelo de programación u otra semántica específica de implementación.

Un *mensaje SOAP* es, fundamentalmente, una transmisión en un sentido entre *nodos SOAP*, desde un *emisor SOAP* a un *receptor SOAP*. De esta forma, y conforme al objetivo de simplicidad, SOAP aporta un paradigma de intercambio de mensajes sin estado y en una sola dirección, aunque, satisfaciendo de igual modo el objetivo de extensibilidad, las aplicaciones pueden crear patrones de interacción más complejos (p.ej., petición/respuesta, petición/múltiples respuestas, etc.) combinando los intercambios en un sentido con las características provistas por un protocolo subyacente y/o información especifica de la aplicación.

Existen numerosas especificaciones que determinan cómo han de realizarse algunas de las tareas más importantes relacionadas con SOAP: reglas para el procesado de los mensajes, modelo de extensibilidad de SOAP, reglas para la vinculación de SOAP con un protocolo subyacente que permita a los nodos intercambiar mensajes y la estructura de los mensajes SOAP. En este apartado, se pretende mostrar, de forma resumida, los elementos que deben formar parte de un mensaje de acuerdo con la especificación de la estructura de los mensajes SOAP. En la siguiente figura, se enumeran las partes de que consta un mensaje SOAP (ver Fig. 18):

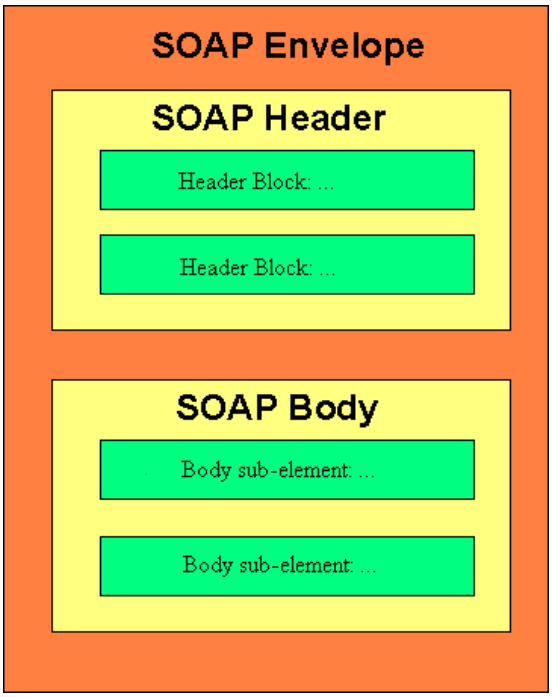


Fig. 18. Estructura de un mensaje SOAP (adaptado de Miltra & Lafon, 2006)

Todo mensaje SOAP está compuesto por un elemento contenedor '*envelope*' que, además del nombre, espacio de nombres y los atributos, contiene de forma opcional un elemento '*header*' o cabecera del mensaje y, obligatoriamente, un elemento '*body*' o cuerpo del mensaje. El elemento '*header*' proporciona un mecanismo para extender un mensaje SOAP de manera modular y descentralizada. Es aquí donde se incluye la información de control distinta a la carga útil de la aplicación que está contenida en el cuerpo del mensaje. Entre la información de control que se puede transmitir en la cabecera del mensaje, se incluyen directivas de transición e información contextual relacionada con el procesado del mensaje. Esto permite a los nodos intermedios proporcionar servicios de valor añadido. Los elementos descendientes directos de '*header*' son denominados '*header blocks*' o bloques de cabecera que representan una agrupación lógica de datos y que pueden estar destinadas a nodos intermedios específicos en el camino entre el emisor y el receptor del mensaje. El elemento '*body*' es obligatorio dentro de '*envelope*', y proporciona un mecanismo para transmitir información relativa a la aplicación al receptor SOAP final. Este elemento contiene, además de un nombre, un espacio de nombres y una lista de atributos, sub-elementos que contienen la carga útil del mensaje.

A continuación, se muestra un ejemplo simple de un mensaje SOAP para la reserva de un viaje:

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Header>
<m:reservation
xmlns:m="http://travelcompany.example.org/reservation"
env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
env:mustUnderstand="true">
<m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
<m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
</m:reservation>
<n:passenger xmlns:n="http://mycompany.example.com/employees"
env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
env:mustUnderstand="true">
<n:name>Åke Jógvan Øyvind</n:name>
</n:passenger>
</env:Header>
<env:Body>
<p:itinerary
xmlns:p="http://travelcompany.example.org/reservation/travel">
<p:departure>
<p:departing>New York</p:departing>
<p:arriving>Los Angeles</p:arriving>
<p:departureDate>2001-12-14</p:departureDate>
<p:departureTime>late afternoon</p:departureTime>
<p:seatPreference>aisle</p:seatPreference>
</p:departure>
<p:return>
<p:departing>Los Angeles</p:departing>
<p:arriving>New York</p:arriving>
<p:departureDate>2001-12-20</p:departureDate>
<p:departureTime>mid-morning</p:departureTime>
<p:seatPreference/>
</p:return>
</p:itinerary>
<q:lodging
xmlns:q="http://travelcompany.example.org/reservation/hotels">
<q:preference>none</q:preference>
</q:lodging>
</env:Body>
</env:Envelope>
```

Fig. 19. Mensaje SOAP para la reserva de un viaje (Miltra & Lafon, 2006)

1.4.2.3. Arquitectura Orientada a Servicios (SOA)

Aunque no existe un acuerdo general en cuanto a su definición, la Arquitectura Orientada a Servicios supone la utilización de servicios débilmente acoplados y altamente interoperables para dar soporte a los requisitos de los procesos de negocio y los usuarios. A continuación, se muestra la definición de SOA propuesta por el consorcio OASIS (http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf):

> *"Un paradigma para organizar y utilizar capacidades distribuidas que pueden encontrarse bajo el control de diferentes dominios de propiedad. Proporciona un mecanismo uniforme para ofrecer, descubrir, interaccionar con y usar las capacidades para producir los efectos deseados de forma consistente con precondiciones y expectaciones medibles."*

En un entorno SOA, los recursos de la red se hacen disponibles a través de servicios independientes que pueden ser accedidos a través de métodos estándar y sin necesidad de conocer cómo están implementados internamente.

SOA es una forma de arquitectura de sistemas distribuidos que está caracterizada generalmente por las siguientes propiedades (Booth et al., 2004):

- Vista lógica. El servicio es una vista abstracta y lógica de programas reales, bases de datos, procesos de negocio, etc., definida en términos de qué hace (nada se define respecto al cómo se hace) y, normalmente, llevando a cabo una operación de la lógica de negocio.
- Orientado a mensajes. El servicio está formalmente definido como un intercambio de mensajes entre entidades suministradoras de servicios y entidades solicitantes de los mismos, sin mencionar las propiedades de las entidades en sí (estructura interna, lenguaje de programación, estructura del proceso, etc.). Esta característica cobra mayor importancia cuando se pretende interaccionar con sistemas heredados ('*legacy systems*') ya que, al abstraerse de cualquier conocimiento relacionado con la estructura interna de una entidad software, cualquier componente software que permita ser accedido por medio de un sistema basado en mensajes, podrá ser utilizado por medio de servicios.
- Orientado a la descripción. Un servicio se describe a través de meta-datos procesables por el ordenador. La descripción da soporte a la naturaleza pública de SOA: sólo aquellos detalles que se exponen públicamente y que son importantes para la utilización del servicio, deben ser incluidos en la descripción.
- Granularidad. Los servicios tienden a utilizar pocas operaciones con mensajes relativamente largos y complejos.
- Orientación a la red. Los servicios tienden a ser orientados hacia el uso sobre una red de comunicaciones, aunque éste no es un requisito indispensable.
- Independiente de la plataforma. Los mensajes se envían en un formato estandarizado e independiente de la plataforma. XML, por ejemplo, satisface estas restricciones.

Existen numerosos métodos para implementar sistemas SOA, incluido el uso de la tecnología de los Servicios Web. En general, la aplicación de Servicios Web para el desarrollo SOA es conveniente cuando la aplicación a desarrollar cumple los siguientes

requisitos (Booth et al., 2004): (i) debe de operar sobre Internet, donde la fiabilidad y velocidad no puede ser garantizada; (ii) donde no hay posibilidad de gestionar el despliegue de la aplicación de forma que todos los proveedores y solicitantes son mejorados de forma instantánea; (iii) donde los componentes del sistema distribuido se ejecutan en diferentes plataformas y productos propietarios ('*vendor products*'); (iv) donde una aplicación existente necesite ser expuesta para su utilización a través de una red de comunicaciones y pueda ser 'envuelta' ('*wrapped*') por un Servicio Web.

1.4.2.4. Ventajas e Inconvenientes

La aplicación de arquitecturas orientadas a servicios y, más concretamente, su implementación a través de Servicios Web, conlleva numerosas ventajas de las que pueden beneficiarse los sistemas desarrollados. Una de las mayores ventajas de la tecnología de los Servicios Web es la utilización de protocolos de transporte estándar como HTTP. Esta medida permite el paso de mensajes entre servicios sin necesidad de preocuparse por los sistemas de seguridad y, en particular, los cortafuegos de las distintas organizaciones. En general, el uso de protocolos estándar facilita la interoperabilidad entre plataformas de distintos fabricantes.

Por otro lado, como ya se mencionó anteriormente, los Servicios Web aportan interoperabilidad entre aplicaciones software con independencia de su implementación, del lenguaje de programación utilizado y de la plataforma en que estén instaladas. Esto ha provocado que esta tecnología se utilice, cada vez con más frecuencia, para permitir el uso de sistemas heredados por parte de las nuevas aplicaciones que se desarrollan en el ámbito de una organización. Por su parte, la independencia con respecto a la implementación aporta flexibilidad al sistema, de forma que se puede modificar la implementación del servicio sin necesidad de actualizar su descripción. Además, los Servicios Web permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.

Por otra parte, no son numerosos pero sí importantes los problemas que presentan los Servicios Web y el paradigma SOA. En general, el mayor limitante de la tecnología de los Servicios Web es su rendimiento. Comparado con otros modelos de computación distribuida como RMI, DCOM o CORBA, la prestación de esta solución es baja. Este problema viene ocasionado por el uso de XML como lenguaje para la codificación de

mensajes. En particular, el problema radica en la necesidad de parsear y componer los ficheros XML, lo cual requiere mayor capacidad computacional y hace que las aplicaciones se ejecuten más lentamente. Como resultado del esfuerzo por resolver este limitante, la W3C creó en 2003 el grupo de trabajo para la caracterización binaria de XML ("*XML Binary Characterization Working Group*", http://www.w3.org/XML/Binary/) que en 2005 se transformó en el grupo de trabajo para el intercambio eficiente de XML ("*Efficient XML Interchange Working Group*", http://www.w3.org/XML/EXI/). El objetivo final es el desarrollo de una especificación de un formato de codificación que permita el intercambio eficiente de documentos XML.

Otro de los inconvenientes de los Servicios Web es que el grado de madurez de algunas de sus especificaciones no puede compararse con el grado de desarrollo de estándares abiertos en computación distribuida como CORBA. Hasta ahora, los investigadores en Servicios Web se han preocupado más de cómo realizar tareas de alto nivel como el descubrimiento y la invocación de servicios, dejando sin cubrir aspectos más básicos como la seguridad, la gestión de transacciones, etc. Hoy en día existen numerosos grupos de trabajo para el desarrollo de este tipo de especificaciones (*WS-*: WS-Security, WS-Trust, WS-Policy, WS-Privacy, WS-ReliableMessaging, WS-Routing, WS-Addressing, etc.*). Esto introduce un plus de riesgo cuando se trata del desarrollo de sistemas comerciales que deben ser aplicados en entornos reales.

Disponiendo de todos los componentes que constituyen la tecnología de Servicios Web, cualquiera podría hacer uso de los servicios ofrecidos por todos los Servicios Web disponibles en Internet. Sin embargo, y a medida que la Web crece en tamaño y diversidad, cada vez es más necesaria la automatización de aspectos relacionados con los Servicios Web como el descubrimiento, la selección, la composición y la invocación. De hecho, una de las principales ventajas de esta tecnología es la posibilidad de componer de forma dinámica servicios utilizando componentes software independientes y reutilizables. El problema, entonces, es que la tecnología actual (en torno a UDDI, WSDL y SOAP) no proporciona los medios para conseguir este dinamismo (Fensel & Bussler, 2002).

I.4.3. Servicios Web Semánticos

Para resolver los limitantes mencionados en el mundo de los Servicios Web, y asimilando conceptos surgidos con la aparición de la Web Semántica, se originó lo que se ha resuelto en denominar Servicios Web Semánticos. Éstos se pueden definir como la aplicación conjunta de conceptos de Servicios Web y de Web Semántica para crear Servicios Web inteligentes (Lara et al., 2003).Esta tecnología es un nuevo paso adelante en la evolución de la Web, que consiste en describir a los Servicios Web con contenido semántico de forma que el descubrimiento de servicios, su composición e invocación se pueda realizar de forma automática por parte de entidades software (p.ej., mediante agentes inteligentes) capaces de procesar la información semántica disponible. En la siguiente figura, se pueden distinguir las principales etapas en el desarrollo de la Web hasta el origen de los Servicios Web Semánticos (Fig. 20):

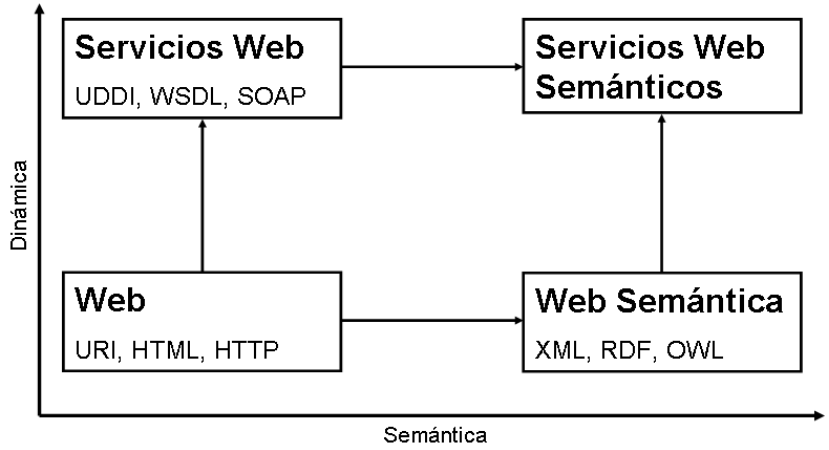


Fig. 20. Evolución de la Web (adaptado de Fensel & Bussler, 2002)

En esta sección se estudiarán las distintas aproximaciones propuestas para alcanzar un estándar en la tecnología de los Servicios Web Semánticos y se enumerarán las plataformas de ejecución de servicios descritos semánticamente que se han desarrollado hasta el momento.

I.4.3.1. Aproximaciones a los Servicios Web Semánticos

En la actualidad, el W3C está examinando cinco aproximaciones a los Servicios Web Semánticos con el propósito de alcanzar un estándar para esta tecnología: OWL-S, WSMO, SWSF, WSDL-S y SAWSDL. A continuación, se presenta un breve estudio acerca de cada una de estas propuestas.

Web Ontology Language for Services (OWL-S)

El origen de la Web Semántica permitió el acceso a funcionalidad en la Web nunca antes concebida. En este entorno, tanto usuarios como agentes software deben ser capaces de realizar ciertos procesos con un alto grado de automatización. Entre éstos, se pueden destacar los de descubrir, invocar, componer y monitorizar recursos Web que ofrecen determinados servicios y poseen propiedades particulares. OWL-S (anteriormente DAML-S) es una ontología de servicios que hace todo esto posible (OWL-S, 2004). En OWL-S, un servicio se entiende como "*un sitio Web que no solamente provee información estática sino que permite efectuar alguna acción o cambio en el mundo, tal como la venta de un producto o el control de un dispositivo físico*". Así, para que un agente software pueda acceder a un Servicio Web, es necesaria una descripción de las capacidades del servicio y de cómo acceder al mismo que sea interpretable por componentes software. Con este propósito se hace uso del concepto de ontología, en particular del lenguaje ontológico OWL y de los mecanismos asociados al mismo.

Una característica a destacar en OWL-S es la distinción que hace entre servicios "atómicos" y servicios "compuestos". Los *servicios atómicos* son aquellos donde un solo programa accesible por la Web, sensor o dispositivo es invocado por un mensaje de solicitud, realiza su tarea y, posiblemente, produce una única respuesta a la solicitud. Por su parte, los *servicios compuestos (o complejos)* son aquellos que están constituidos por múltiples servicios primitivos y pueden requerir una interacción extendida o conversación entre el solicitante, y que el conjunto de servicios que están siendo utilizados. El objetivo último de OWL-S es hacer posible la ejecución de las siguientes tareas sobre los servicios disponibles en la Web:

1. *Descubrimiento automático de servicio Web*. Se trata de automatizar el proceso de localización de los servicios Web que puedan proporcionar una clase particular de capacidades a la par que cumple unas determinadas restricciones especificadas por el cliente. Para esto, OWL-S permite anunciar de forma declarativa las propiedades y capacidades de los servicios.
2. *Invocación automática de servicio Web*. A través de esta tarea, se intenta realizar la ejecución automática de un servicio Web por parte de un programa computerizado o agente, dada únicamente una descripción declarativa de ese servicio. El marcado de servicios Web de OWL-S proporciona una API

declarativa e interpretable por la máquina que incluye la semántica de los argumentos a especificar cuando se ejecutan las llamadas, así como la semántica de los mensajes que se producen como resultado de la ejecución del servicio, tanto cuando éste falla, como cuando tiene éxito.

3. *Composición e interoperación automática de servicio Web*. Dada la descripción de alto nivel de un objetivo, se permite realizar una tarea compleja a partir de la selección automática, composición e interoperación de diversos servicios Web. Para dar soporte a estas tareas, OWL-S proporciona especificaciones declarativas de los prerrequisitos y consecuencias de la aplicación de servicios individuales y un lenguaje para describir composiciones de servicio e interacciones de flujos de datos.

En esencia, OWL-S es una ontología que contiene los elementos fundamentales que caracterizan un servicio y que permite describir las capacidades que sustenta un servicio. Esta ontología contiene tres tipos de conocimiento fundamentales: (1) qué es lo que un servicio hace (*ServiceProfile*, utilizado para publicitar el servicio), (2) cómo se usa el servicio (*ServiceModel*), y (3) cómo interaccionar con el servicio (*ServiceGrounding*, detalles sobre los protocolos de transporte). En la figura que se muestra a continuación, se presenta la ontología de alto nivel de OWL-S para describir un servicio (ver Fig. 21):

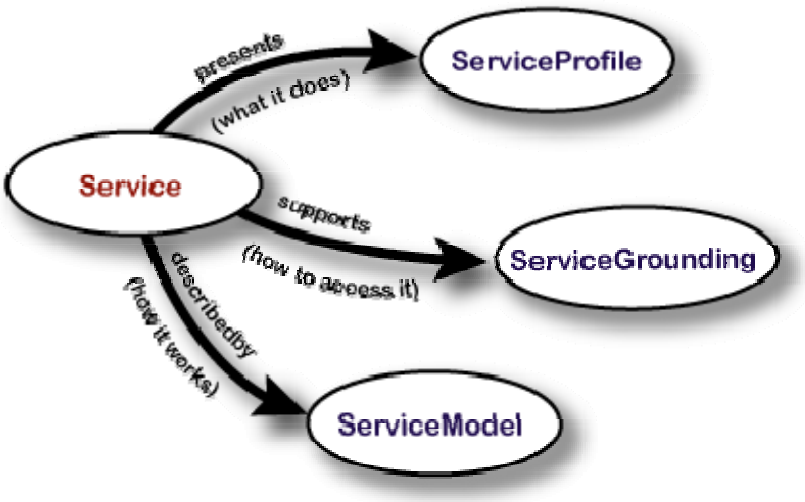


Fig. 21. Ontología de alto nivel de OWL-S (OWL-S, 2004)

El perfil de un servicio (*ServiceProfile*) indica lo que un servicio es capaz de realizar de un modo apropiado, para que un agente de búsqueda de servicios pueda determinar si un servicio satisface o no sus necesidades. Esta forma de representación incluye una descripción de la funcionalidad del servicio, las limitaciones en la aplicación del servicio, la calidad de servicio, y los requisitos que el solicitante debe de satisfacer para poder hacer uso del servicio de forma satisfactoria (ver Fig. 22).

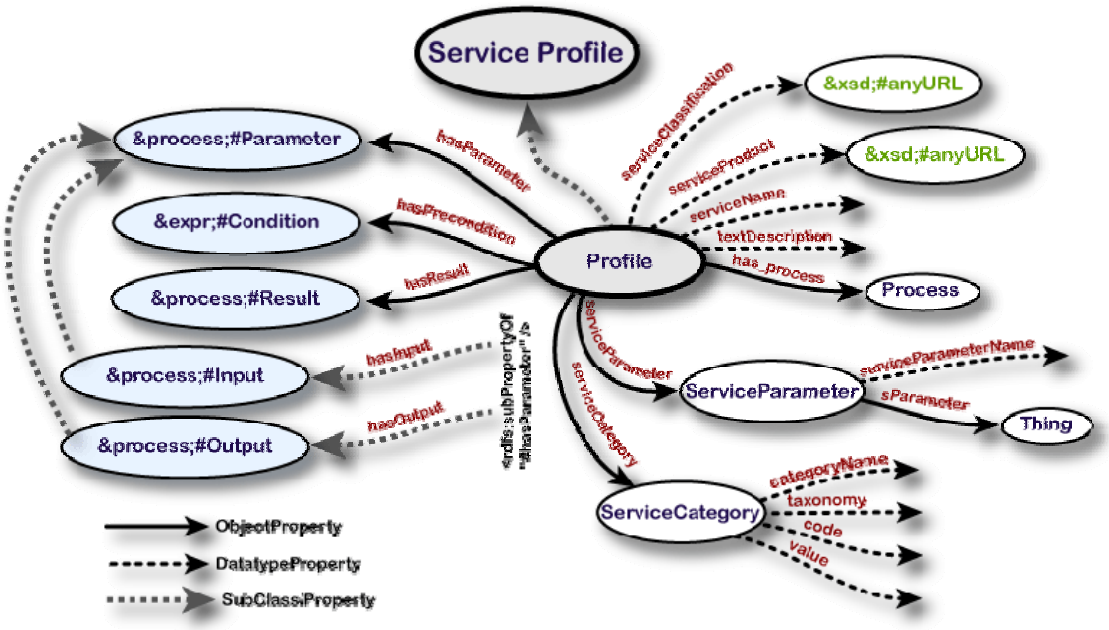


Fig. 22. Algunos de los conceptos y relaciones asociados al perfil del servicio (OWL-S, 2004)

Por su parte, el modelo de un servicio (*ServiceModel*) indica al cliente cómo utilizar el servicio, detallando el contenido semántico de las solicitudes, las condiciones bajo las cuales se pueden obtener determinados resultados y, donde sea necesario, la descripción del proceso que es necesario seguir para producir dichos resultados. Es decir, el *ServiceModel* describe cómo se ha de preguntar a un servicio y qué ocurre cuando un servicio se lleva a cabo. Para el caso de servicios compuestos, esta descripción puede ser utilizada por un agente buscador de servicios de cuatro formas diferentes: (1) para realizar un análisis en profundidad acerca de si el servicio realmente satisface sus necesidades; (2) para componer las descripciones de múltiples servicios con el fin de realizar una tarea específica; (3) para coordinar las actividades de los diferentes participantes durante el proceso de ejecución del servicio; y (4) para monitorizar la ejecución del servicio. En la siguiente figura, se muestran algunos de los conceptos y relaciones asociados al modelado de un servicio como un proceso (ver Fig. 23).

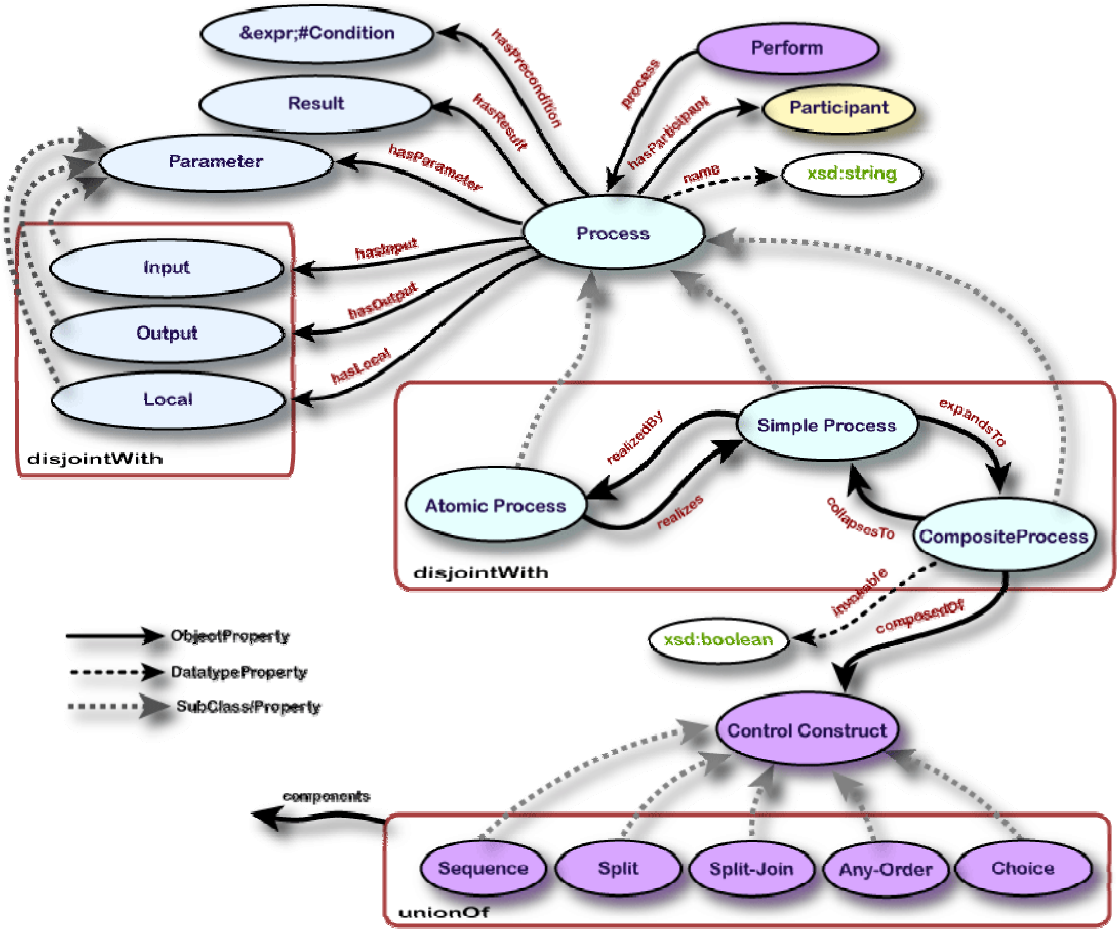


Fig. 23. Ontología del proceso de alto nivel (OWL-S, 2004)

Por último, el *ServiceGrounding* especifica los detalles de cómo un agente puede acceder a un servicio. En general, este elemento sirve para especificar un protocolo de comunicación, formato de mensajes y otros detalles específicos del servicio como los números de puertos que deben utilizarse para contactar con el servicio. Además, el *ServiceGrounding* debe indicar, para cada tipo semántico de entrada o salida especificado en el *ServiceModel*, un modo no ambiguo para intercambiar elementos de datos de ese tipo con el servicio (esto es, las técnicas de serialización empleadas). En la siguiente figura, se muestran las correspondencias entre los elementos de OWL-S y los de WSDL, que conforman un posible *ServiceGrounding* (ver Fig. 24).

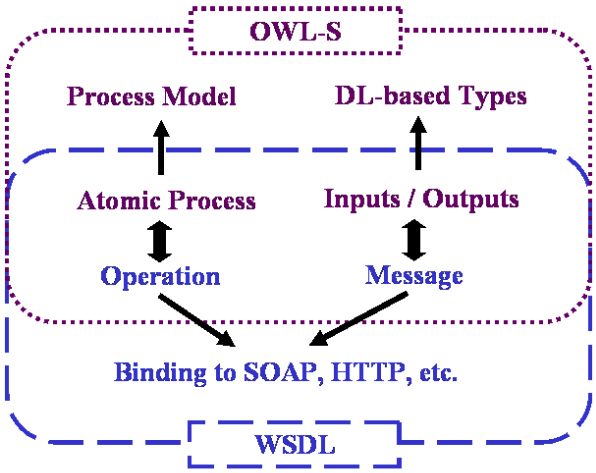


Fig. 24. Correspondencia entre OWL-S y WSDL (OWL-S, 2004)

Web Services Modeling Ontology (WSMO)

La propuesta relacionada con la descripción semántica de los Servicios Web que cronológicamente siguió a OWL-S fue WSMO (WSMO, 2005). Con una visión similar a la de los autores de OWL-S, WSMO proporciona un marco de trabajo conceptual y un lenguaje formal para describir de forma semántica todos y cada uno de los aspectos relevantes relacionados con los servicios Web para, así, facilitar la automatización de tareas tales como el descubrimiento, la combinación y la invocación de servicios electrónicos sobre la Web. En WSMO, un servicio Web se define como una entidad computacional capaz, una vez invocada, de satisfacer el objetivo de un usuario.

WSMO está basado en el modelo conceptual propuesto en el "*Web Service Modeling Framework*" (WSMF) (Fensel & Bussler, 2002), que identifica cuatro elementos fundamentales para describir servicios Web semánticos:

1. *Ontologías*: proporcionan la terminología que será usada por los restantes elementos.
2. *Objetivos*: representan los deseos de los usuarios o intenciones que deben ser satisfechas por algún servicio Web.
3. *Descripciones de servicios Web*: define los aspectos funcionales y de comportamiento de un servicio Web.
4. *Mediadores*: tienen el propósito de gestionar de forma automática los problemas de interoperabilidad que surjan entre los restantes elementos.

Teniendo como base los conceptos identificados en WSMF, WSMO proporciona una especificación ontológica para los elementos que conforman el núcleo de los servicios

Web semánticos. A diferencia de OWL-S, el lenguaje de ontologías utilizado con este propósito en WSMO es WSML (WSML, 2005). En la siguiente figura se muestran los conceptos de alto nivel que constituyen la ontología de WSMO (ver Fig. 25).

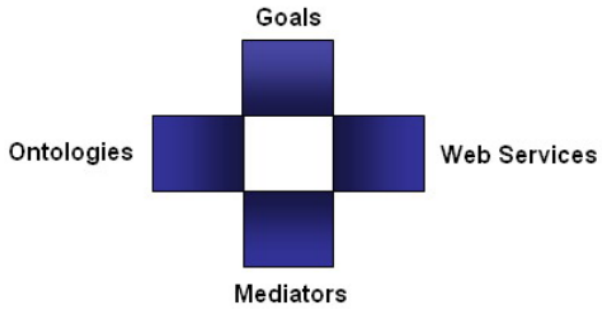


Fig. 25. Elementos de alto nivel de WSMO (WSMO, 2005)

A continuación, se muestran las relaciones que mantienen los conceptos de nivel superior de la ontología propuesta por WSMO (ver Fig. 26). Todos los conceptos son elementos WSMO y poseen una serie de propiedades no funcionales que los caracterizan de uno u otro modo.

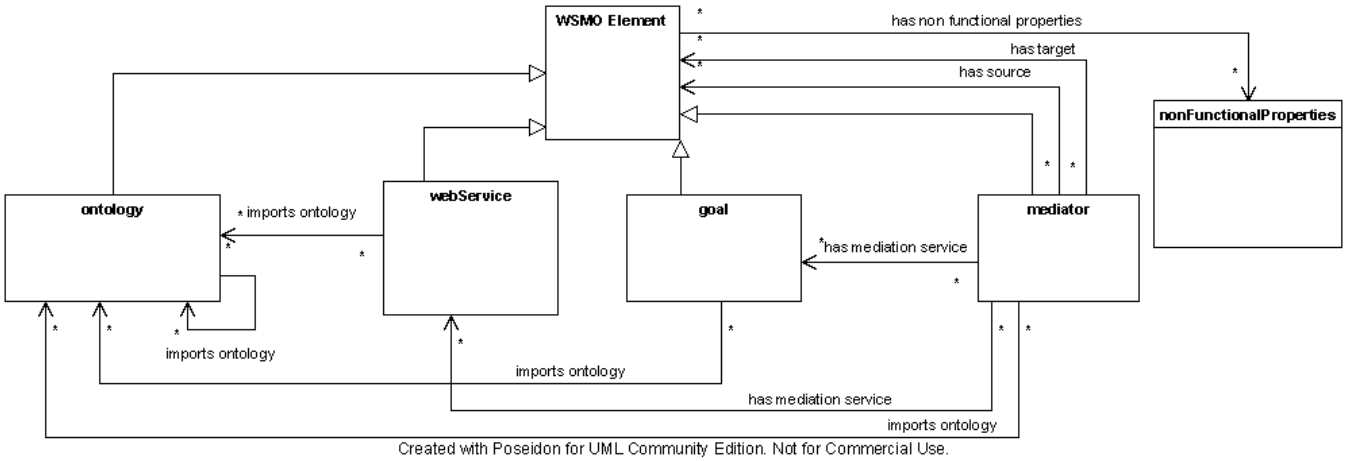


Fig. 26. Diagrama UML de los conceptos de nivel superior de WSMO (WSMO, 2005)

En WSMO, las ***ontologías*** son la clave para conectar la semántica conceptual del mundo real definida y acordada por comunidades de usuarios. Con este propósito, las ontologías definen una terminología común concertada indicando conceptos, relaciones entre conceptos y axiomas, que se pueden definir como expresiones en algún lenguaje lógico (ver Fig. 27). Los axiomas se utilizan para capturar las propiedades semánticas de las relaciones y los conceptos. Por tanto, las ontologías proporcionan en WSMO la

terminología básica de la que harán uso los restantes elementos para describir los aspectos relevantes del dominio del discurso.

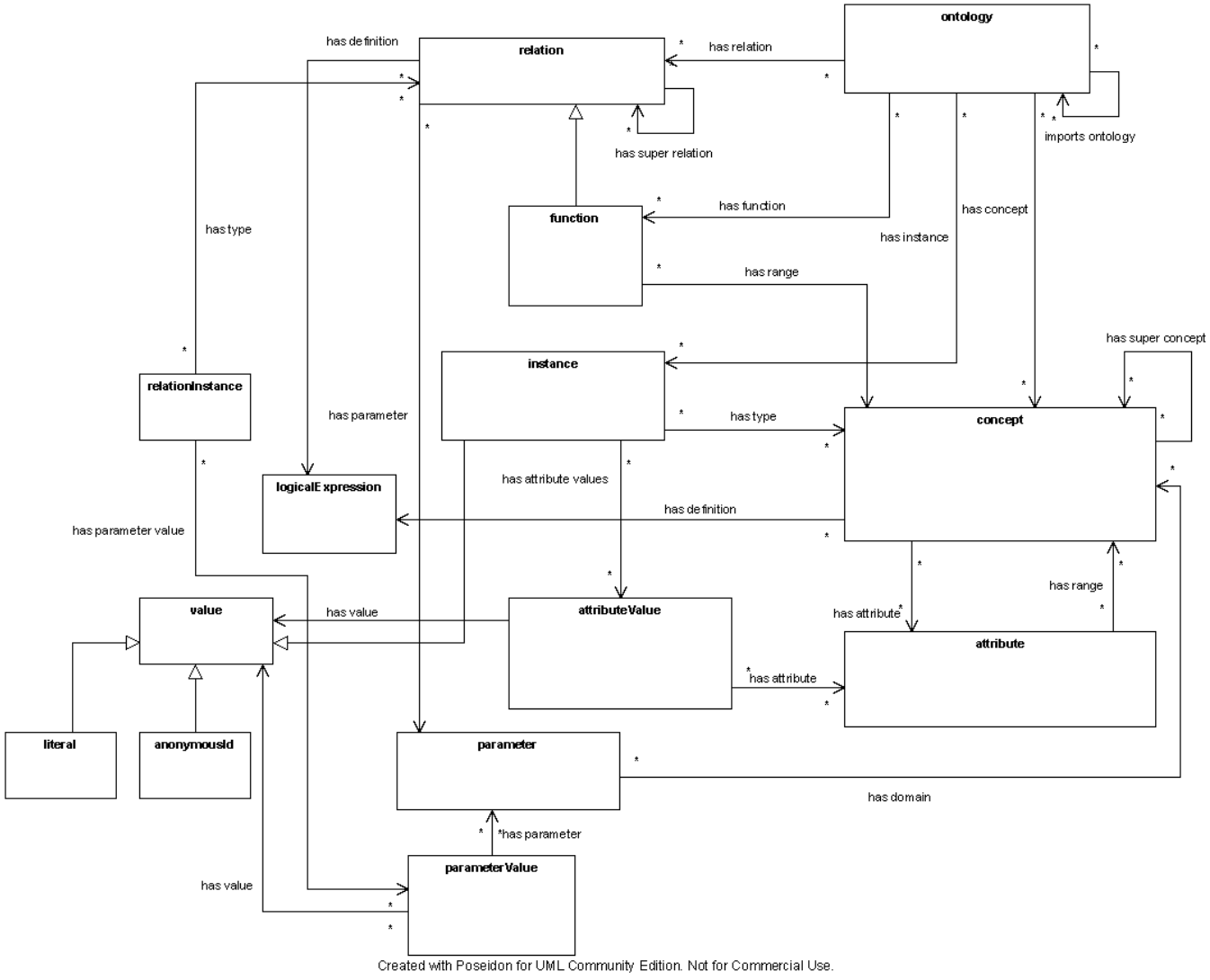


Fig. 27. Elementos relacionados con el concepto Ontología en WSMO (WSMO, 2005)

Un ***servicio Web*** en WSMO describe a una entidad computacional que permite el acceso a servicios que proporcionan algún valor en un dominio. La descripción de un servicio Web en WSMO consiste en aspectos funcionales, no funcionales y de comportamiento del servicio. Esta descripción comprende las capacidades, interfaces y el funcionamiento interno del servicio (ver Fig. 28). Todos estos aspectos de un servicio Web son descritos mediante el uso de la terminología definida por parte de las ontologías. Se distingue, además, el concepto de servicio Web, entendido como entidad computacional capaz de satisfacer un objetivo de usuario, del concepto servicio,

entendido como el valor real proporcionado por la invocación del servicio Web (un mismo servicio Web puede proveer varios servicios).

Por otra parte, un ***objetivo*** (ver Fig. 28) se refiere a la representación de una meta para cuya consecución es necesaria la ejecución de un servicio Web. Los objetivos pueden ser descripciones de servicios Web que, potencialmente, satisfarían los deseos del usuario. De forma similar a lo que ocurre en las descripciones de servicios Web, las ontologías pueden utilizarse para definir la terminología del dominio que describe los aspectos relevantes de los objetivos.

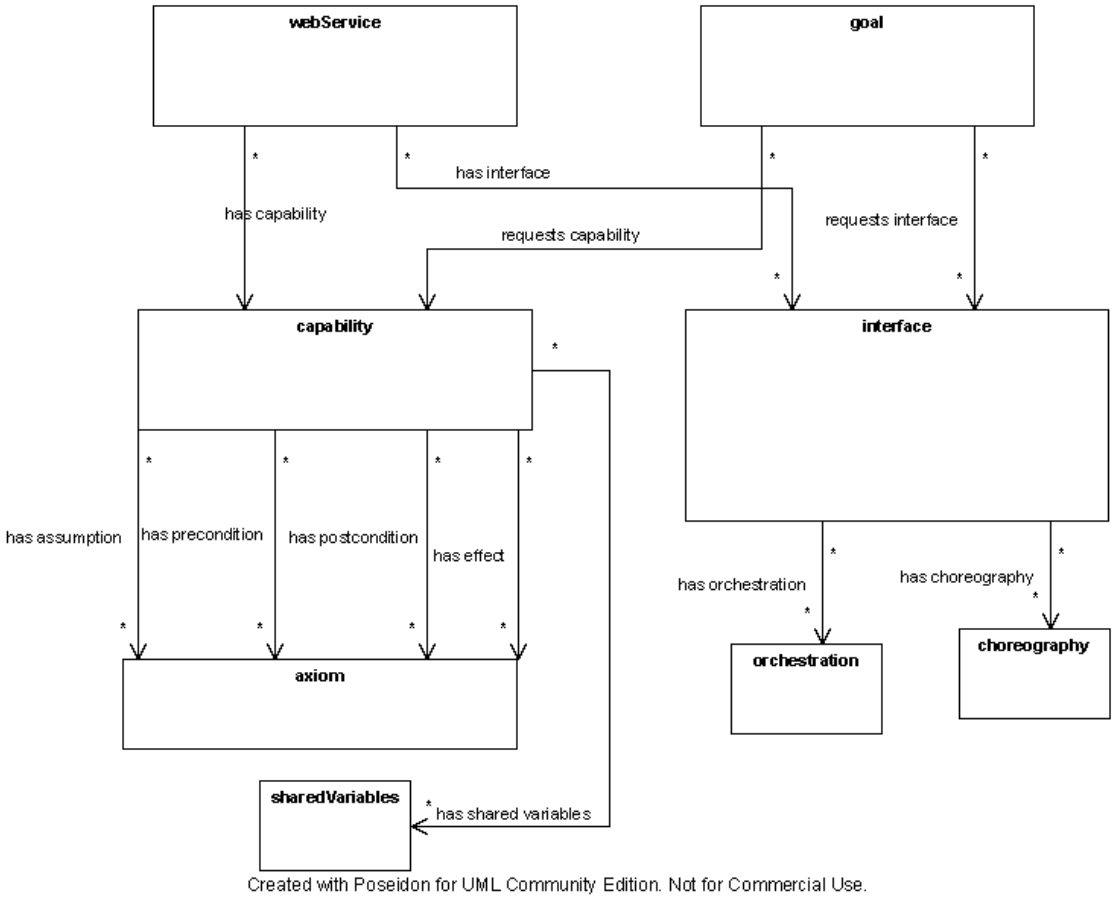


Fig. 28. Elementos relacionados con los conceptos Servicio Web y Objetivo de WSMO (WSMO, 2005)

Finalmente, por ***mediadores*** se entienden aquellos elementos capaces de superar los problemas de interoperabilidad que puedan surgir entre los distintos elementos de WSMO. Estos son el concepto clave para resolver incompatibilidades a nivel tanto de datos, como de procesos y protocolos. La mediación de datos permite resolver incongruencias entre diferentes terminologías utilizadas. Por su parte, la mediación a

nivel de protocolo trata con los problemas relacionados con las incompatibilidades en la comunicación entre servicios Web. Por último, la mediación a nivel de procesos permite la combinación de servicios Web (y objetivos). Para tratar todas estas cuestiones, se distinguen cuatro tipos de mediadores (ver Fig. 29):

- *ggMediators*: mediadores que conectan dos objetivos. Esta conexión representa el refinamiento del objetivo fuente en el objetivo destino, o establece una equivalencia si ambos objetivos son intercambiables.
- *ooMediators*: mediadores que importan ontologías y resuelven posibles incompatibilidades en la representación de las ontologías.
- *wgMediators*: mediadores que conectan servicios Web con objetivos. La conexión entre un servicio Web y un objetivo significa que el servicio Web (total o parcialmente) satisface el objetivo asociado.
- *wwMediators*: mediadores que conectan dos servicios Web.

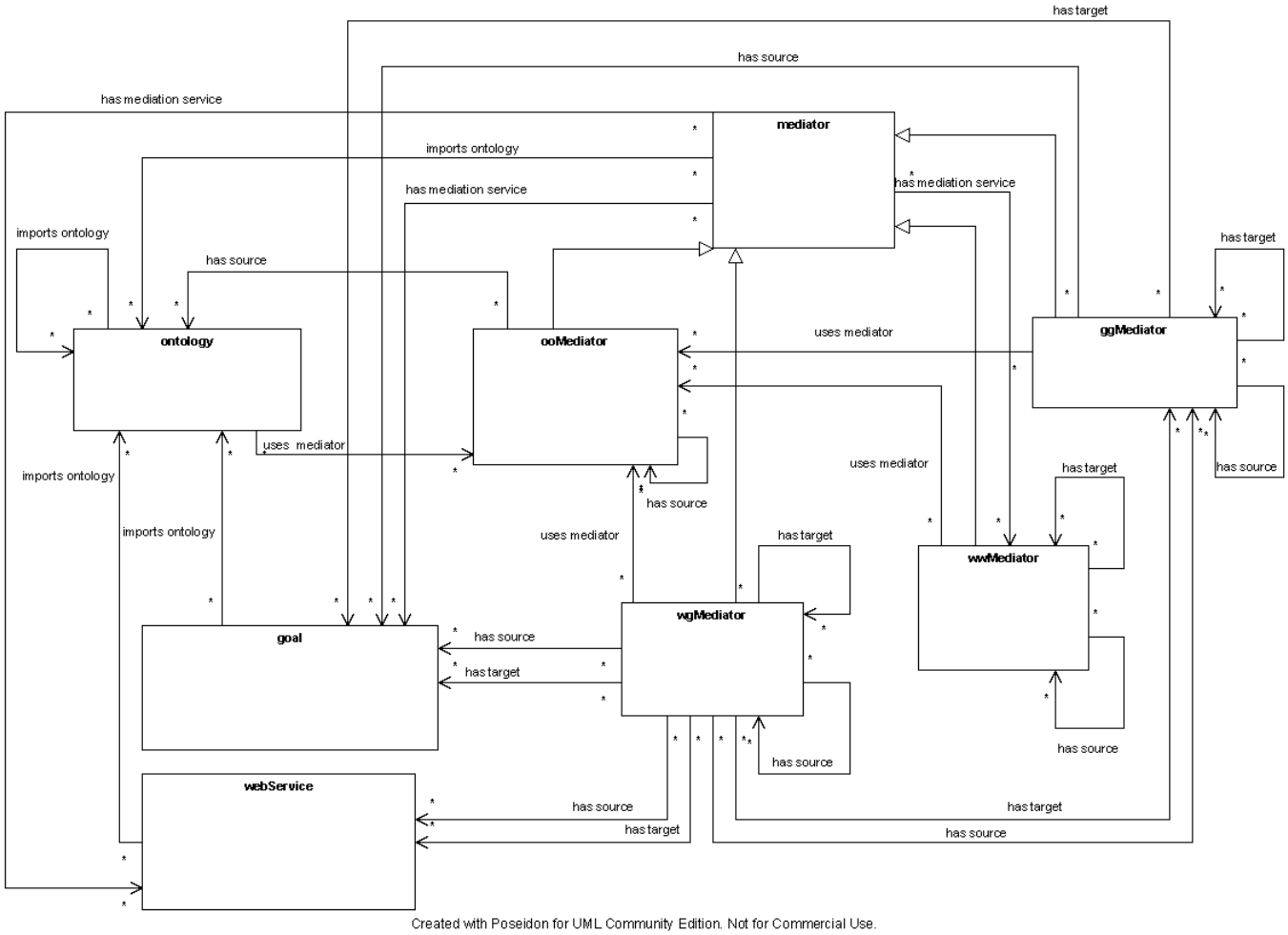


Fig. 29. Mediadores que se tienen en cuenta en WSMO (WSMO, 2005)

Tanto OWL-S como WSMO tienen un propósito común, la especificación de información semántica relacionada con los Servicios Web para permitir la automatización de tareas como el descubrimiento, la composición y la ejecución. Sin embargo, existen diferencias sustanciales entre ambas aproximaciones, como se destaca en (Polleres & Lara, 2005). Entre las diferencias más significativas, caben destacar las siguientes: (i) OWL-S no separa lo que un usuario quiere de lo que un servicio puede proporcionar (diferencia entre Objetivo y descripción de servicio Web en WSMO); (ii) WSMO permite indicar propiedades no funcionales en todos y cada uno de los elementos de la ontología, haciendo, además, uso de vocabularios comúnmente aceptados como "*Dublin Core*" (DC) y "*Friend of a Friend*" (FOAF), mientras que OWL-S sólo permite indicar estas propiedades en la ontología del perfil y no se basa en especificaciones de metadatos estándar; (iii) OWL-S no distingue entre coreografía y orquestación y la definición de procesos no se basa en un modelo formal, mientras que en WSMO se hace una clara distinción entre ambos conceptos y se permite la definición de diversos modos de interaccionar con cada servicio.

Semantic Web Services Framework (SWSF)

El "*Semantic Web Services Framework*" (SWSF) es una nueva iniciativa para la definición de una especificación semántica más rica sobre la actual tecnología de los Servicios Web que permita una mayor automatización y flexibilidad en la provisión y uso de servicios (SWSF, 2005). El marco de trabajo está diseñado a su vez con el propósito de soportar la construcción de herramientas y metodologías más potentes en el entorno de los Servicios Web, así como promocionar el uso de procesos de razonamiento sobre servicios fundamentados en semántica. En este sentido, y al igual que la aproximaciones anteriores, SWSF pretende incorporar una semántica más rica que dé soporte a una mayor automatización de tareas como la selección e invocación de servicios, traducción del contenido de los mensajes entre servicios heterogéneos que operan entre sí, composición de servicios, etc., además de permitir aproximaciones integrales para la monitorización de servicios y la recuperación de errores.

Esta propuesta está compuesta de dos componentes principales: el "*Semantic Web Services Language*" (SWSL) y el "*Semantic Web Services Ontology*" (SWSO). El SWSL es un lenguaje lógico de propósito general que incluye ciertas características para hacerlo más apropiado para las necesidades de la Web y los Servicios Web Semánticos (SWSF-SWSL, 2005). Entre estas peculiaridades, se incluye el uso de URIs, la

integración de los tipos que forman parte de XML y el uso de mecanismos de importación y espacios de nombres compatibles con XML. Este lenguaje es el utilizado para especificar las caracterizaciones formales sobre los conceptos relacionados con los Servicios Web y sus descripciones. Incluye dos sub-lenguajes: *SWSL-FOL*, basado en lógica de primer orden con extensiones de HiLog y sintaxis de marcos de F-Logic, y se utiliza para expresar la caracterización formal de los conceptos relacionados con los Servicios Web (lenguaje ontológico), y *SWSL-Rules*, basado en el paradigma de programación lógica, y utilizado para dar soporte al uso de la ontología de servicios en procesos de razonamiento y en entornos de ejecución basados en ese paradigma.

La SWSO define un modelo conceptual por el cual los Servicios Web pueden ser descritos, y una representación formal o axiomatización de dicho modelo (SWSF-SWSO, 2005). La axiomatización completa se implementa en lógica de primer orden utilizando SWSL-FOL, con una semántica de Teoría de Modelos que especifica un significado preciso de los conceptos de la ontología. Esta forma de la ontología que utiliza lógica de primer orden se denomina "*First-Order Logic Ontology for Web Services*" (FLOWS). Adicionalmente, los axiomas de FLOWS se han traducido al lenguaje de reglas SWSL-Rules, obteniéndose, de este modo, una ontología basada en semántica de programación lógica denominada "*Rules Ontology for Web Services*" (ROWS). Por tanto, FLOWS, al igual que WSMO y OWL-S (sobre el cual se fundamenta y extiende), es una ontología formal sobre conceptos de servicios, que proporciona un marco de trabajo conceptual para describir y razonar sobre servicios.

En FLOWS, un servicio es un objeto conceptual que corresponde a un servicio Web (u otro servicio accesible de forma electrónica). Una contribución clave de la ontología FLOWS sobre OWL-S es el desarrollo de un modelo de procesos de comportamiento rico, basado en el lenguaje de especificación de procesos PSL ("*Process Specification Language*"), que es un estándar ISO que proporciona una ontología extensible, con diferentes capas, para especificar las propiedades de los procesos. Este modelo de procesos permite la interoperabilidad entre los distintos modelos de procesos sobre Servicios Web que están apareciendo en la actualidad, y hace posible la automatización de tareas en línea con la perspectiva de los servicios Web semánticos. FLOWS está dividido en tres partes fundamentales:

1. *Ontología de descriptores de servicio*: esta ontología proporciona la información básica acerca de un servicio Web de forma similar a lo que podrían ser unas

páginas amarillas independientes del dominio o lo que incluye el ServiceProfile de OWL-S. Estos descriptores de servicios se utilizan, generalmente, para el descubrimiento automatizado de servicios Web. Existe una lista inicial de descriptores que permiten identificar el nombre del servicio, el autor, la descripción textual, la versión etc., pero se contempla la posibilidad de extender esta lista inicial de acuerdo con las exigencias de cada momento.

2. *Ontología de modelo de procesos*: extiende la ontología de procesos genérica propuesta por PSL para hacer disponibles los conceptos que son útiles en el contexto de los Servicios Web. Se añaden dos elementos fundamentales sobre PSL: la noción estructurada de proceso atómico, tal y como se especifica en OWL-S, y una infraestructura para especificar varias formas de flujos de datos. FLOWS-Core contiene la extensión más básica sobre PSL que proporciona una representación abstracta para servicios (Web), su impacto en el mundo y la transmisión de mensajes entre ellos. Sobre el modelo de procesos de FLOWS-Core, se pueden añadir numerosas extensiones que aumentan la expresividad del lenguaje. Los conceptos principales de la ontología de FLOWS-Core son: Servicio, Proceso Atómico, Compuesto De, Mensaje y Canal.
3. *Grounding*: los conceptos de SWSO para la descripción de servicios y las instanciaciones de estos conceptos para describir un servicio particular, son especificaciones abstractas, de forma que no se ofrecen detalles relacionados con el formato de los mensajes, los protocolos de transporte ni las direcciones de red que son necesarios para hacer un uso efectivo de un Servicio Web. El propósito del *Grounding* es relacionar estos elementos abstractos con detalles más concretos. Para ello, se ha considerado el uso de WSDL, ya que este lenguaje de descripción de servicios es ampliamente utilizado y contiene los medios apropiados para especificar los detalles necesarios. Por tanto, el *Grounding* define correspondencias entre ciertos conceptos de la SWSO a construcciones WSDL que describen la realización concreta de esos conceptos.

Web Service Semantics (WSDL-S)

WSDL-S (WSDL-S, 2005) ha supuesto un cambio radical con respecto a las perspectivas "tradicionales" para la incorporación de semántica a los Servicios Web. WSDL-S define un mecanismo para asociar anotaciones semánticas con Servicios Web que han sido descritos utilizando WSDL. A diferencia de los lenguajes descritos

anteriormente, WSDL-S asume la existencia de modelos semánticos del dominio relevantes para cada servicio, mantenidos fuera del ámbito de los documentos WSDL y que pueden ser referenciados desde un documento WSDL a través de elementos de extensibilidad ideados como parte de la propuesta WSDL-S (ver Fig. 30).

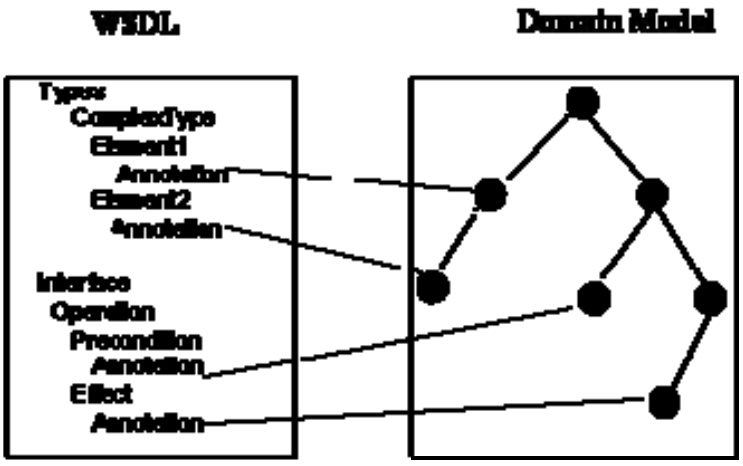


Fig. 30. Anotación semántica de los elementos de WSDL (WSDL-S, 2005)

El fundamento conceptual de esta aproximación se basa, al igual que en los casos anteriores, en el hecho de que el estándar WSDL actual opera a un nivel sintáctico que carece de la expresividad semántica necesaria para representar los requisitos y capacidades de los servicios. De este modo, la incorporación de semántica puede mejorar la reutilización de software y el descubrimiento, facilitar la composición de servicios Web y permitir la integración de aplicaciones heredadas como parte de la integración de procesos de negocio. La información semántica que se considera por parte de WSDL-S incluye las definiciones de precondiciones, entradas, salidas y efectos de las operaciones de servicios Web.

Más concretamente, WSDL 2.0 incorporaba, a día del envío de la recomendación al W3C, los siguientes constructores para representar descripciones de servicios: '*interface*', '*operation*', '*message*', '*binding*', '*service*' y '*endpoint*'. De entre estos, sólo '*interface*', '*operation*' y '*message*' tratan con la definición abstracta de un servicio y son, por tanto, los necesarios para añadir las anotaciones semánticas sobre las descripciones de servicios que permite el descubrimiento, composición e invocación dinámica de servicios. En resumen, los elementos y atributos de extensibilidad planteados por esta propuesta son los siguientes:

- Un atributo de extensión denominado "*modelReference*" para especificar la asociación entre una entidad WSDL y un concepto en algún modelo semántico. Este atributo puede añadirse a un tipo complejo, elemento, operación, así como a los elementos de extensión "*precondition*" y "*effect*".
- Un atributo de extensión denominado "*schemaMapping*", añadido a los elementos XSD y a los tipos complejos, para manejar diferencias estructurales entre los elementos de diseño de un Servicio Web y sus correspondientes conceptos del modelo semántico.
- Dos elementos, denominados "*precondition*" y "*effect*", situados como descendientes de "*operation*". Estos nuevos elementos se utilizan, principalmente, para el descubrimiento de servicios, y no son estrictamente necesarios para la invocación de un servicio dado.
- Un atributo de extensión en el elemento "*interface*" denominado "*category*", que consiste en información de categorización de servicios que puede ser utilizado cuando se publica un servicio en un registro de Servicios Web como UDDI. Mediante este atributo se permite afinar el proceso de búsqueda de servicios en los registros.

Entre las ventajas de esta novedosa aproximación sobre otras, como OWL-S y WSMO, se pueden destacar las siguientes:

1. Los usuarios pueden describir, de forma incremental, todos los detalles, tanto semánticos como a nivel de operaciones en WSDL, un lenguaje que es familiar para la comunidad de desarrolladores.
2. Al externalizar los modelos semánticos del dominio, WSDL-S permanece independiente del lenguaje de representación de ontologías que se desee utilizar. Esto permite a los desarrolladores de Servicios Web anotar sus servicios con el lenguaje ontológico de su elección (p.ej., UML, OWL, etc.). Esto supone una ventaja adicional, porque reutilizar modelos del dominio existentes expresados con lenguajes de modelado como UML puede acelerar la incorporación de anotaciones semánticas.
3. Es relativamente fácil modificar las herramientas existentes actualmente alrededor de la especificación WSDL para incorporar los elementos propuestos por esta

aproximación. En todo caso, es más rápido y fiable que el desarrollo de herramientas implementadas desde cero.

Semantic Annotations for Web Services Description Language (SAWSDL)

Con la misma misión que WSDL-S surgió, en Abril de 2006, el grupo de trabajo SAWSDL[10] como parte de la "*W3C Web Services Activity*"[11]. El objetivo de este grupo de trabajo es el desarrollo de un mecanismo que permita la anotación semántica de descripciones de Servicios Web.

Actualmente, la propuesta del grupo de trabajo, "*Semantic Annotations for WSDL and XML Schema*" (SAWSDL, 2007), se encuentra en estado de "*Working Draft*" (borrador) a la espera de una revisión final. En este documento, se definen un conjunto de atributos de extensión para WSDL y XML Schema que permite la descripción de semántica adicional de los componentes de WSDL. Esta anotación semántica se consigue, al igual que ocurría en WSDL-S, utilizando en WSDL referencias sobre modelos semánticos como, por ejemplo, ontologías. Así, SAWSDL no especifica un lenguaje concreto para representar los modelos semánticos, sino que incorpora mecanismos por medio de los cuales es posible referenciar desde documentos WSDL a conceptos de modelos semánticos definidos externamente utilizando anotaciones.

De los elementos que incorpora la especificación WSDL 2.0 para representar descripciones de servicios (como, '*Type Definition*', '*Interface*', '*Interface Operation*', '*Interface Fault*', '*Binding*', '*Service*' y '*Endpoint*'), SAWSDL se centra en la anotación semántica de aquellos que ofrecen una definición abstracta de un servicio (esto es, '*Type Definition*', '*Interface*', '*Interface Operation*' e '*Interface Fault*') para permitir el descubrimiento, composición e invocación dinámica de servicios. A continuación, se describen los dos constructores básicos para anotación semántica propuestos por SAWSDL:

- Un atributo de extensión denominado "*modelReference*" para especificar la asociación entre un componente WSDL o XML Schema y un concepto en algún modelo semántico. Es utilizado para anotar definición de tipos complejos en XML Schema, definición de tipos simples, declaraciones de elementos y declaraciones de atributos, además de interfaces, operaciones y defectos en WSDL.

[10] http://www.w3.org/2002/ws/sawsdl/

- Dos atributos de extensión denominados "*liftingSchemaMapping*" y "*loweringSchemaMapping*", que se añaden a la declaración de elementos en XML Schema, definición de tipos complejos y simples para especificar correspondencias entre datos semánticos y XML. Estas correspondencias se usarán posteriormente durante la invocación de servicios.

I.4.3.2. Entornos de Ejecución de Servicios Web Semánticos

Junto con algunas de las aproximaciones mencionadas anteriormente, se han desarrollado herramientas que permiten interactuar con Servicios Web Semánticos, automatizando las tareas de descubrimiento, selección, composición, invocación y monitorización. En los siguientes apartados, se da información sobre de cuatro de las más importantes de dichas herramientas.

Herramientas OWL-S: OWL-S Virtual Machine

No existe para la aproximación OWL-S una herramienta única que sirva de entorno de ejecución de Servicios Web Semánticos. Sin embargo, existe una colección de herramientas individuales, con funcionalidades complementarias, que se centran en diferentes aspectos de su modelo conceptual. Entre las herramientas más destacadas, se encuentran las siguientes[12]: "*OWL-S Protégé-based Editor*", "*OWL-S Editor*", "*OWL-S Matcher*", "*OWL-S Virtual Machine*", "*WSDL2OWL-S converter*" y "*OWL-S2UDDI converter*".

Tanto "*OWL-S Protégé-based Editor*" (Elenius et al., 2005) como "*OWL-S Editor*" (Scicluna et al., 2004) permiten la creación de anotaciones semánticas sobre Servicios Web de forma sencilla, ocultando los complejos constructores de que se compone esta ontología para servicios.

El "*OWL-S Matcher*"[13] (OWLSM) implementa un algoritmo que devuelve diferentes grados de correspondencia para los elementos individuales de las descripciones OWL-S. En particular, este algoritmo considera los elementos del perfil del servicio. La

[11] http://www.w3.org/2002/ws/

[12] http://www.daml.org/services/owl-s/tools.html

[13] http://owlsm.projects.semwebcentral.org/

ordenación de los servicios devueltos se hace según un criterio que permite seleccionar un servicio de entre un conjunto potencialmente extenso de resultados.

El "*WSDL2OWL-S converter*" (Paolucci et al., 2003b) permite la transición entre descripciones de servicios en WSDL a descripciones semánticas en OWL-S. Los resultados de esta transformación son una especificación completa del *Grounding* y una especificación parcial del modelo de procesos y del perfil en OWL-S. Por su parte, y complementando la funcionalidad ofrecida por la herramienta descrita previamente, el "*OWL-S2UDDI converter*"[14] convierte las descripciones de perfil de OWL-S en los anuncios UDDI correspondientes que pueden ser entonces publicados en un registro UDDI.

La herramienta más completa de entre las desarrolladas para OWL-S es el "*OWL-S Virtual Machine*" (OWL-S VM, antes DAML-S VM) (Paolucci et al., 2003a) que proporciona un cliente de Servicios Web de propósito general para la invocación de servicios basado en el modelo de procesos provisto por OWL-S. La principal funcionalidad que ofrece OWL-S VM es controlar la interacción entre Servicios Web basado en la descripción de un proceso definido mediante OWL-S. La arquitectura del sistema, así como su relación con los Servicios Web, se presenta en la siguiente figura (ver Fig. 31):

[14] http://projects.semwebcentral.org/projects/owl-s2uddi/

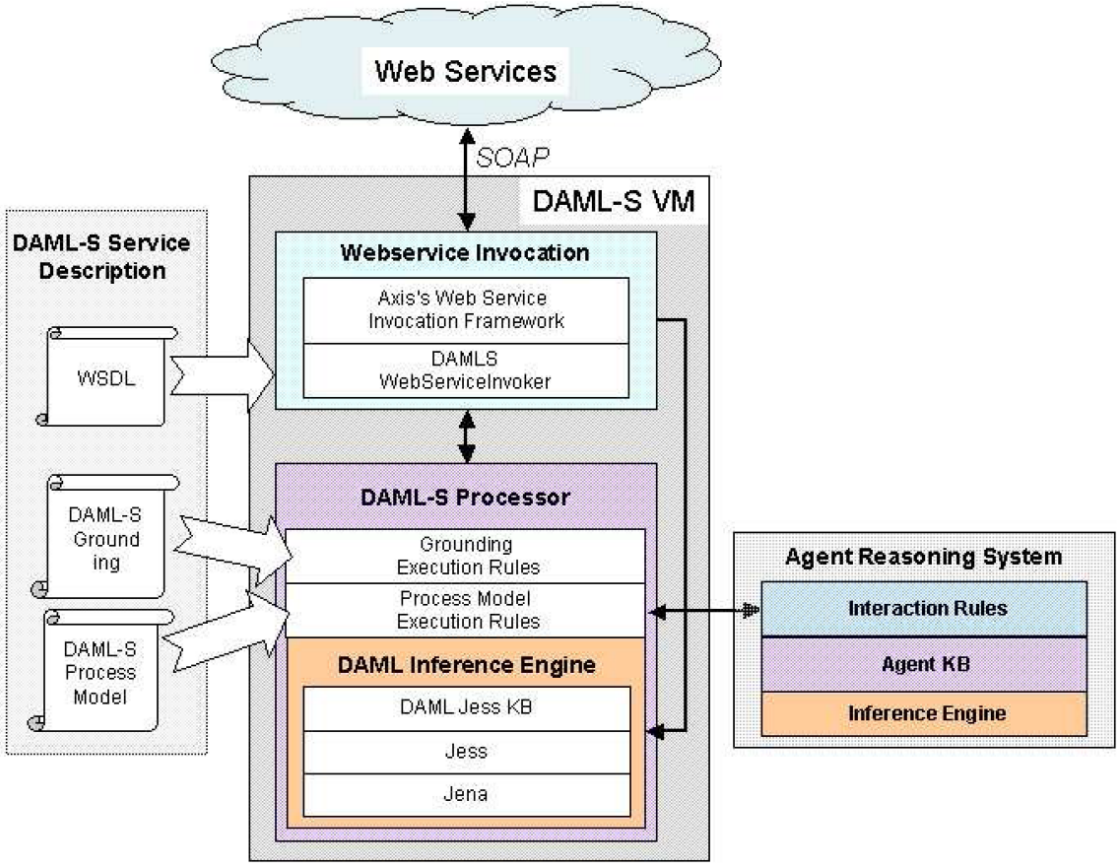


Fig. 31. Arquitectura de la OWL-S VM (Paolucci et al., 2003a)

El sistema está compuesto por tres elementos fundamentales: el "*DAML-S Service Description*", el "*DAML-S VM*" y el "*Agent Reasoning System*". En primer lugar, el "*DAML-S Service Description*" especifica el conocimiento que utiliza el "*DAML-S VM*" para controlar la interacción con otros Servicios Web. Este conocimiento está compuesto por el modelo de procesos, el *Grounding*, y la descripción WSDL de los vínculos. El "*DAML-S VM*" está dividido en dos módulos lógicos, el "*DAML-S Processor*" y el "*Web service Invocation*". El "*DAML-S Processor*" utiliza el "*DAML Inference Engine*" y un conjunto de reglas para implementar la semántica operacional del modelo de procesos de DAML-S, y el *grounding* para gestionar la interacción con el proveedor. Por su parte, el módulo "*Web service Invocation*" es responsable de la transferencia de información con el proveedor. Por último, el "*Agent Reasoning System*" es el encargado de realizar los procesos de razonamiento y de toma de decisiones durante la interacción.

Web Service Execution Environment (WSMX)

El "*Web Service Execution Enviornment*" (WSMX, 2005) es un entorno de ejecución para el descubrimiento, selección, composición, mediación, invocación y monitorización dinámica de Servicios Web Semánticos. Así, WSMX es una implementación de referencia basada en el modelo conceptual descrito en WSMO (WSMO, 2005) y que sirve como arquitectura para evaluar la idoneidad de los elementos que este modelo conceptual incorpora. De forma muy general, la funcionalidad de WSMX se resume en lo siguiente: WSMX puede satisfacer el objetivo de un usuario a través de la selección dinámica de un Servicio Web apropiado, la mediación de los datos que necesitan ser comunicados a este servicio y su invocación En la siguiente figura (ver Fig. 32), se muestran los distintos componentes que constituyen WSMX.

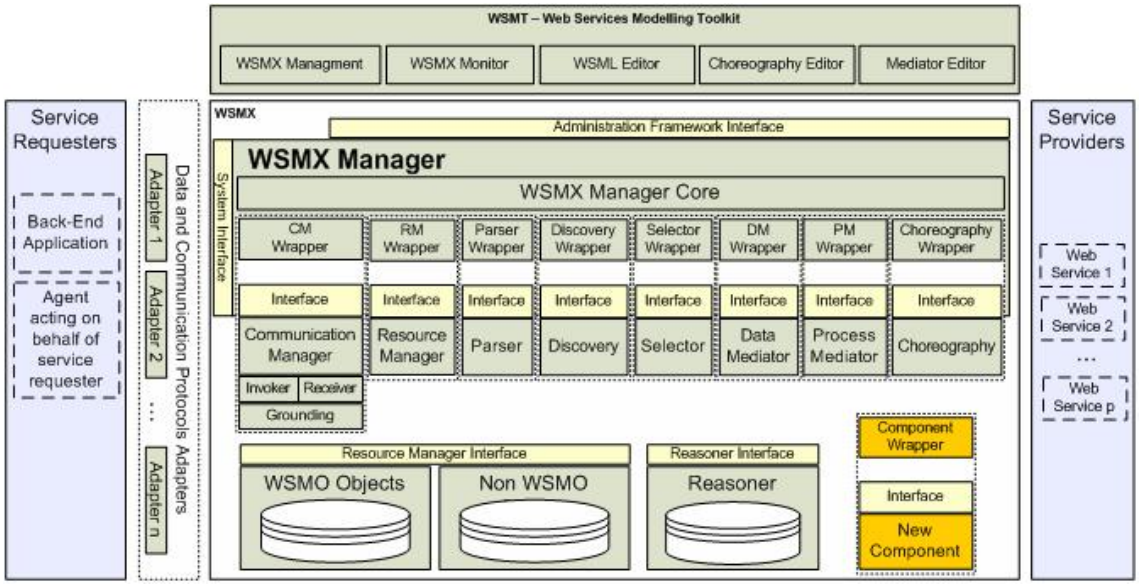


Fig. 32. Arquitectura de WSMX (WSMX, 2005)

La funcionalidad global del sistema se encuentra dispersa entre sus componentes constituyentes. El "*WSMX Manager Core*" coordina las actividades de los restantes componentes del sistema y las interacciones que se producen entre ellos. Los datos que se manejan dentro de WSMX se representan por medio de un formato interno en forma de un evento con un tipo y un estado. El citado componente, núcleo de la arquitectura de WSMX, gestiona desde un punto de vista lógico el procesamiento de todos lo eventos, trasladándolos a los diferentes componentes de WSMX. El componente "*Discovery*" es el encargado de proporcionar la funcionalidad necesaria para hacer corresponder la

descripción de los Servicios Web Semánticos con los objetivos indicados por los usuarios.

Por su parte, el componente "*Selection*" se utiliza para encontrar los servicios más apropiados, en caso de que se hayan encontrado numerosos servicios que satisfacen el objetivo establecido por el usuario. El componente "*Data Mediator*" proporciona los medios para transformar datos basados en conceptos de una ontología en datos basados en conceptos de otra ontología. El proceso de mapeo está basado en reglas definidas entre los conceptos de las ontologías fuente y destino. Si se da el caso que se precisa mediación de datos y estos datos no se encuentran en formato XML, se hace uso del conversor XML para traducir los resultados del mediador en XML. Esto es necesario porque las invocaciones de Servicios Web se realizan vía SOAP, y el formato de los mensajes para SOAP es XML. El componente "*Process Mediator*" ayuda en la resolución de los problemas de heterogeneidad que pueden aparecer durante la invocación de Servicios Web basados en coreografía. Este componente se encarga de asegurar que los procesos públicos del servicio invocador y del invocado se corresponden a la perfección.

El componente "*Parser*" está basado en un compilador y un parseador de mensajes. El compilador parsea mensajes WSML recibidos desde el WSMO Editor en la capa de interfaz con el usuario, valida estos mensajes contra WSMO y, entonces, almacena los elementos del mensaje en el repositorio de WSML. Los elementos compilados por WSMX son los meta-datos de servicios Web, ontologías y mediadores. Una vez que todos estos elementos han sido compilados a WSMX, están disponibles para su uso durante la ejecución de los objetivos enviados a WSMX. Por otro lado, el parseador de mensajes parsea los mensajes WSML que contienen los objetivos enviados a WSMX. Cada objetivo es parseado y almacenado de forma persistente. La diferencia entre el parseador de mensajes y el compilador es que, mientras este último maneja los meta-datos de servicios Web, ontologías y mediadores, el parseador funciona sobre instancias de objetivos.

Los "*Adapters*" permiten a aplicaciones que no pueden comunicarse directamente con las interfaces provistas por WSMX comunicarse con el sistema. El "*Choreography Engine*" media entre los patrones de comunicación del solicitante y los del proveedor del servicio. El "*Reasoner*" proporciona servicios de razonamiento que permiten realizar los procesos de descubrimiento, mediación, validación de posibles

composiciones de servicios o determinación de si un servicio compuesto en un proceso es ejecutable en un contexto dado. El "*Communication Manager*" hace posible que WSMX se comunique con Servicios Web externos disponibles en la red, enviando los mensajes de solicitud necesarios y recibiendo la repuesta. Este componente está a su vez compuesto por tres elementos: "*Grounding*", que permite la trasformación entre los datos semánticos disponibles en WSMX y los datos en formato XML o RDF utilizados en las comunicaciones externa y viceversa; "*Invoker*", utilizado cuando se precisa ejecutar un Servicio Web externo; y "*Receiver*", que recibe la respuesta y la vuelve al formato semántico interno de WSMX. Por último, el "*Resource Manager*" es el responsable de gestionar los repositorios, donde se almacenan las definiciones de objetivos, servicios Web, ontologías y mediadores en WSMX.

Internet Reasoning Service (IRS): IRS-I, IRS-II, IRS-III

El proyecto IRS[15] tiene el propósito general de dar soporte a la construcción automática o semi-automática de sistemas mejorados semánticamente sobre Internet. Mientras IRS-I (Crubezy et al., 2003) soportaba la creación de sistemas intensivos en conocimiento estructurados conforme al framework UPML e IRS-II (Motta et al., 2003) integraba el framework UPML con la tecnología de Servicios Web, IRS-III (Cabral et al., 2006) incorpora y extiende la ontología WSMO (WSMO, 2005). De este modo, IRS-III es un framework para Servicios Web Semánticos que da soporte a la publicación, localización, composición y ejecución de Servicios Web en base a su descripción semántica que se ajusta al modelo conceptual definido por WSMO.

El marco de trabajo IRS está compuesto por tres componentes principales (ver Fig. 33): "*IRS Server*", "*IRS Publisher*" e "*IRS Client*", que se comunican a través de mensajes SOAP.

[15] http://kmi.open.ac.uk/projects/irs/

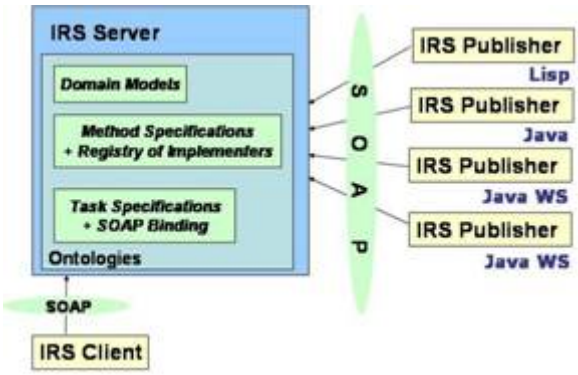


Fig. 33. Framework IRS

El "*IRS Server*" contiene descripciones de objetivos, mediadores, servicios Web y ontologías, de forma que el descubrimiento, la composición, la mediación, el razonamiento y la invocación de servicios son controlados por este componente. El "*IRS Publisher*" lleva a cabo dos funciones principales: enlazar Servicios Web con sus descripciones semánticas almacenadas en el "*IRS Server*", y generar automáticamente una envoltura que permita que código independiente en Lisp o Java pueda ser invocado como un Servicio Web a través de su descripción WSDL. Finalmente, el "*IRS Client*" proporciona una interfaz de usuario para la invocación de servicios basada en objetivos.

El framework IRS-III sigue esta arquitectura distribuida, compuesta por el servidor IRS-III, las plataformas de publicación y los clientes, todo ello comunicado por medio del protocolo SOAP (ver Fig. 34). El servidor maneja la gestión de la ontología y la ejecución de los modelos de conocimiento definidos para WSMO. Además, el servidor recibe las solicitudes SOAP desde las aplicaciones de clientes a través del API, tanto para la invocación de servicios basada en objetivos, como para crear y editar descripciones de objetivos, servicios Web y mediadores. Por su parte, las plataformas de publicación permiten a los proveedores de servicios asociar descripciones semánticas a sus servicios y proporcionan componentes para invocar servicios en un lenguaje o plataforma específicos. Cuando un Servicio Web se publica en IRS-III, la información acerca de la plataforma de publicación se asocia con la descripción del servicio Web para su uso posterior en la invocación.

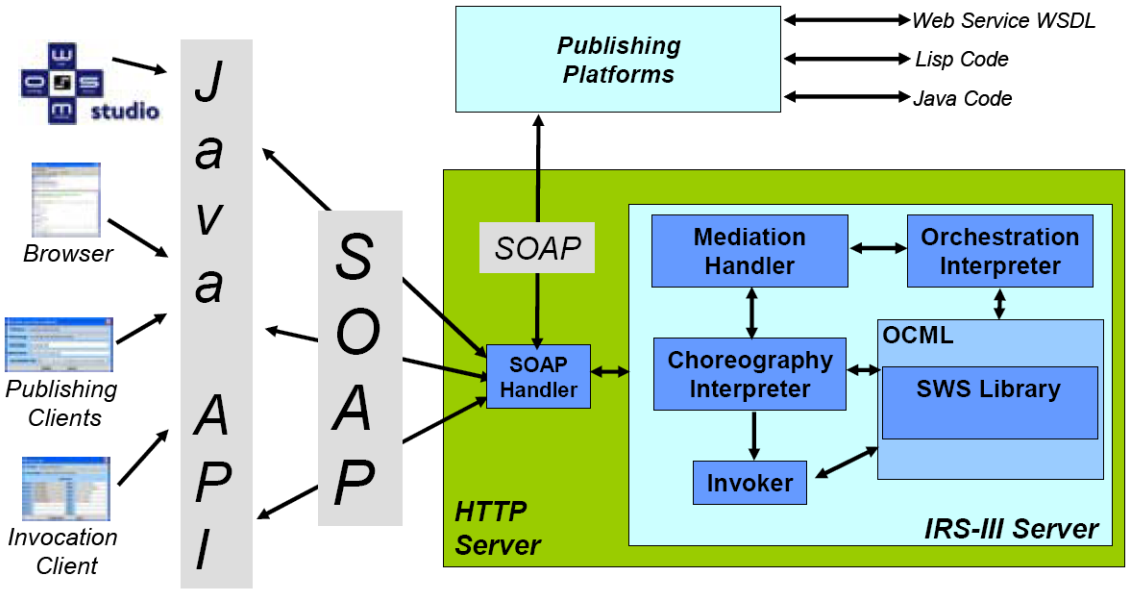


Fig. 34. Arquitectura del framework IRS-III (Cabral et al., 2006)

Los principales componentes de la arquitectura IRS-III son los siguientes (Cabral et al., 2006):

- "*SWS Library*": almacena las descripciones semánticas en el lenguaje de representación OCML. Está estructurada en los modelos de conocimientos que sugiere WSMO: objetivos, servicios Web, mediadores y ontologías.
- "*Choreography Interpreter*": interpreta el *grounding* y las transiciones protegidas de la descripción de la coreografía cuando es requerido por el "*Mediation Handler*".
- "*Orquestration Interpreter*": interpreta el flujo de trabajo de la descripción de la orquestación cuando es requerido por el "*Mediation Handler*".
- "*Mediation Handler*": existen componentes de mediación específicos que dan soporte a las actividades de selección, composición e invocación de servicios. Estas actividades pueden requerir la ejecución de un servicio de mediación o de reglas de mapeo declaradas en la descripción de un mediador.
- "Invoker": este componente del servidor se comunica con la plataforma de publicación enviando las entradas del cliente y llevando los resultados de vuelta al cliente.

METEOR-S: Semantic Web Services and Processes

El objetivo principal del proyecto METEOR-S[16] es la definición y el soporte al ciclo de vida completo de los Servicios Web Semánticos: (i) desarrollo de Servicios Web Semánticos, (ii) publicación y descubrimiento de servicios, (iii) composición de procesos Web, (iv) optimización y análisis de restricciones, y (v) ejecución de Servicios Web Semánticos. Para ello, se ha hecho uso de cuatro categorías de semántica: semántica de datos, semántica funcional, semántica no funcional y semántica de ejecución. La semántica de datos describe los datos (entradas y salidas) de los Servicios Web. La semántica funcional describe la funcionalidad de los Servicios Web (esto es, qué hacen) mientras que la semántica no funcional describe aspectos no funcionales de los servicios como la calidad del servicio o las reglas de negocio. Finalmente, la semántica de ejecución modela el comportamiento de Servicios Web y de procesos.

Una representación de alto nivel de los elementos que constituyen la arquitectura de METEOR-S (Verma et al., 2005) se muestra en la siguiente figura (ver Fig. 35):

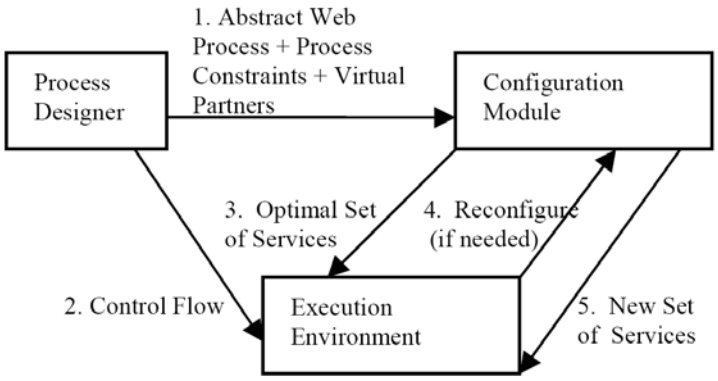


Fig. 35. Arquitectura de alto nivel de METEOR-S (Verma et al., 2005)

El "*Process Designer*" es una interfaz gráfica de usuario para diseñar procesos Web abstractos. Éstos consisten en procesos con Servicios Web virtuales, no reales. El proceso abstracto se representa mediante WS-BPEL, el estándar industrial de facto para representar flujos de trabajo con Servicios Web. El "*Configuration Module*" es el responsable de encontrar un conjunto óptimo de Servicios Web reales para el proceso basado en las restricciones establecidas. Finalmente, el "*Execution Environment*" se encarga de manejar la interacción entre el proceso y los Servicios Web (ver Fig. 36).

[16] http://lsdis.cs.uga.edu/projects/meteor-s/

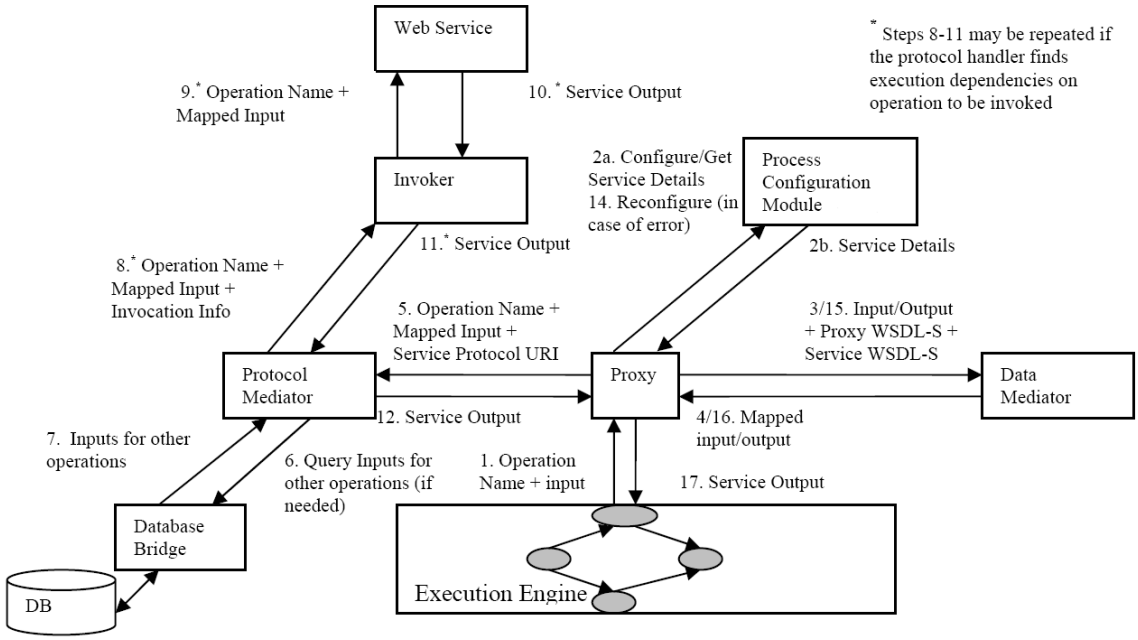


Fig. 36. "*Execution Environment*" (Verma et al., 2005)

El motor de ejecución invoca una operación específica de un servicio. La solicitud de invocación es enviada al Proxy de ese servicio, que comienza utilizando el mediador de datos para transformar los datos de entrada en el formato esperado por el Servicio Web. El Proxy entonces comprueba con el mediador de protocolos que dicha operación puede ser invocada. Si se da el caso de que otras operaciones deben de ejecutarse previamente, el manejador de protocolos genera un plan de ejecución. Todas las operaciones en el plan de ejecución se invocan concurrentemente utilizando el invocador. Si no se dispone de todas las entradas necesarias para invocar los servicios, esta información se extrae de una base de datos. Finalmente, el resultado de la operación se envía de vuelta a través del Proxy al motor de ejecución.

I.4.4. Problemas y Carencias de los Servicios Web Semánticos

Desde su concepción, la Web Semántica en general y los Servicios Web Semánticos en particular han estado asociados al concepto de agente (Hendler, 2001; McIlraith et al., 2001; Gibbins et al., 2003; Paolucci & Sycara, 2003). Refiriéndose a agente, en este ámbito, como aquella entidad software autónoma (o sea, sin necesidad de intervención humana) que es capaz de sacar provecho de los datos que han sido anotados semánticamente como lo podría hacer un usuario humano.

Por tanto, centrándose en la tecnología de Servicios Web Semánticos, el marcado semántico de la funcionalidad ofrecida por los servicios, permite que agentes software

con las habilidades adecuadas sean capaces de hacer uso de estos servicios para satisfacer objetivos de alto nivel (Fensel & Bussler, 2002; OWL-S, 2004). Desde un punto de vista más integral, y teniendo en cuenta las posibilidades ofrecidas por todas y cada una de las aproximaciones existentes en la actualidad para la provisión de Servicios Web Semánticos, los agentes deben ser capaces de realizar de forma autónoma las tareas de descubrimiento, selección, composición, ejecución y monitorización de los servicios, adaptándose, en cada momento, a los cambios que se pueden producir en el entorno debido al dinamismo propio de este tipo de sistemas.

En resumen, el "problema" de la tecnología de Servicios Web Semánticos, desde su propia concepción, es su dependencia de entidades software de nivel superior con capacidades cognitivas que posean la capacidad de acceder al contenido semántico de las descripciones de los servicios, procesarlo y entenderlo de modo que puedan hacer un uso efectivo y apropiado de la funcionalidad que estos servicios ofrecen.

I.5. Problema a Resolver

Tanto la tecnología de agentes como los Servicios Web Semánticos pueden utilizarse de forma separada para el desarrollo de aplicaciones con unas propiedades extraordinarias. Una infraestructura basada en agentes puede beneficiarse de propiedades tales como autonomía, flexibilidad, dinamismo, pro-actividad, reactividad, aprendizaje, inteligencia, etc. Por su parte, un entorno de trabajo basado en Servicios Web (Semánticos) proporciona un alto grado de interoperabilidad entre plataformas, sistemas operativos y lenguajes de programación, además de permitir el desarrollo de componentes débilmente acoplados y altamente reutilizables. Sin embargo, y como se ha destacado en secciones anteriores, ambas tecnologías poseen una serie de inconvenientes que limitan su aplicabilidad en entornos de ejecución reales.

De forma resumida, se puede destacar que el mayor problema de la tecnología de agentes y que, en mayor medida, ha restringido su aplicación a entornos meramente de investigación, es la utilización de protocolos de comunicaciones no estándar. El uso de estos protocolos propietarios como RMI o el protocolo definido por la OMG para CORBA, IIOP ('*Internet Inter-Orb Protocol*'), impide la posibilidad de que dos agentes situados en distintas localizaciones puedan comunicarse con éxito si, entre medias, se interpone un cortafuegos. El uso del protocolo HTTP para la comunicación en el mundo

de los Servicios Web permite a los servicios una comunicación fluida y eficiente salvando este escollo.

Por otra parte, los Servicios Web Semánticos surgieron con el objetivo de permitir la utilización de los servicios por parte de los sistemas software, sin necesidad de la intervención humana. La descripción semántica de las capacidades de los servicios permite a entidades software determinar las posibilidades de dicho servicio de forma similar a como lo podría hacer un humano, de forma que los procesos de descubrimiento, selección, composición, invocación y monitorización de tales servicios se pueden hacer de forma automática. Por tanto, desde su concepción los Servicios Web Semánticos se han ligado con los agentes software que, en este ámbito, serían las entidades capaces de procesar las descripciones semánticas de los servicios.

Teniendo en cuenta lo indicado en este apartado, el objetivo último de la tesis en cuestión es el desarrollo de una plataforma-marco de trabajo para la integración de la tecnología de agentes y la de Servicios Web Semánticos de forma que las ventajas aportadas por cada una de las mismas por separado puedan ser explotadas al máximo, mientras que sus problemas son aliviados. Para esto, se ha hecho uso de ontologías como medio de comunicación entre ambas tecnologías de forma que se mantiene su independencia y se evita, por tanto, la necesidad de modificar los estándares existentes para cada una de ellas en la actualidad.

I.6. Resumen

En este capítulo, se presenta un exhaustivo estudio del estado del arte de los tres pilares fundamentales de la investigación que se relata en esta tesis doctoral: Ingeniería Ontológica, Tecnología de Agentes y los Servicios Web Semánticos. Si bien el concepto de ontología podría, lógicamente, haberse incluido como parte del estudio de los Servicios Web Semánticos, se ha optado por separarlo en una sección propia con el fin de enfatizar la importancia de este elemento en la solución propuesta en esta tesis.

Así, después de la primera sección, introducción al capítulo, se muestra un profundo estudio de diversos aspectos relacionados con las ontologías. En primer lugar, se define el término "ontología", citando las numerosas definiciones que se han propuesto para este ambiguo término y constantando la ausencia de consenso en este sentido. La definición adoptada en este trabajo es la establecida por Studer et al. (1998): "*Una*

ontología es una especificación formal y explícita de una conceptualización compartida". A continuación, se enumeran los distintos tipos de ontologías, clasificándolas tanto por el conocimiento que contienen como por la motivación de las mismas. Posteriormente, se describen brevemente los formalismos lógicos que constituyen la base de los lenguajes ontológicos, a saber, lógica de primer orden, lógica descriptiva y programación lógica. Finalmente, se detallan los principales lenguajes para la representación de ontologías existentes actualmente, en particular aquellos concebidos tras la aparición de la Web Semántica: RDF, RDF-S, y el lenguaje ontológico estándar propuesto por el W3C, esto es, OWL. En este apartado también se hace referencia a WSML, un lenguaje para la representación de ontologías que tiene un especial interés por su relación con los Servicios Web Semánticos.

El segundo bloque principal de este capítulo es el relativo a la tecnología de agentes. En este bloque, primeramente, se define el concepto de "agente" y se enumeran las propiedades que éste debe poseer para considerarse "inteligente" entre las que se encuentran, básicamente, la reactividad, la pro-actividad y la habilidad social. En concreto, la definición de agente inteligente que se aplica en esta tesis doctoral es la siguiente: "*Un agente inteligente es una entidad software situada en un entorno concreto, capaz de mostrar un comportamiento autónomo, reactivo y proactivo sobre este entorno y preparado para interaccionar con otros agentes con el único fin de satisfacer unos determinados objetivos establecidos por un ente (software o humano) que es representado por el agente*". A continuación, se introduce el término de "Sistema Multi-Agente", enfatizando en la realidad de que, en la mayoría de los casos, los sistemas están compuestos por varios agentes, actuando sobre un mismo entorno, que deberán cooperar para conseguir objetivos comunes o competir por el uso de recursos escasos. Esto nos conduce lógicamente a la noción de "negociación". Un mecanismo de negociación es necesario cuando dos o más agentes colaboran para conseguir un objetivo. Además, se indican los componentes que forman parte de todo mecanismo de negociación, básicamente protocolo y estrategia de negociación. Seguidamente, se pone de relieve la dificultad que constituye la elaboración de un Sistema Multi-Agente, y la necesidad, por tanto, de metodologías para guiar el proceso de desarrollo. En concreto, se estudia en profundidad la metodología INGENIAS (Gómez-Sanz & Fuentes, 2002; Pavón & Gómez-Sanz, 2003), que ha sido aplicada en esta tesis doctoral, aunque también se mencionan otras como, por ejemplo, MAS-CommonKADS, MaSE, ZEUS,

GAIA y MOBMAS. Al tiempo que surgían las metodologías, y también con el propósito de facilitar el desarrollo de Sistemas Multi-Agente, fueron apareciendo numerosas herramientas que permiten la interconexión y ejecución de los agentes de un Sistema Multi-Agente, denominadas "plataformas multi-agente". La plataforma JADE (Bellifemine et al., 2003) es descrita en detalle, mientras que se ofrece una breve recapitulación de las características de otras plataformas como FIPA-OS y ZEUS. Este bloque finaliza con la exposición de los problemas y carencias de los que adolece esta tecnología, destacando como principal causa de su escasa aplicación en el sector industrial la utilización de protocolos de comunicaciones no estándar.

Los Servicios Web Semánticos son examinados con exhaustividad en la siguiente sección. En primer lugar, se constata el hecho de que esta tecnología surgió como la combinación de la Web Semántica y los Servicios Web. Por tanto, para llegar a comprender los entresijos de la tecnología, es necesario conocer los detalles relativos a sus elementos constituyentes. Así, primeramente se introduce la Web Semántica, sus antecedentes y fundamentos tecnológicos. Además, se presenta su arquitectura en capas y se mencionan las ventajas e inconvenientes de la tecnología. A continuación, se hace lo propio con los Servicios Web, comentando los antecedentes de esta tecnología y describiendo los elementos que la conforman, básicamente, WSDL, SOAP y UDDI. Asimismo, se introduce el término de "Arquitectura Orientada a Servicios" como un nuevo paradigma de programación basado en Servicios Web, y se revelan las ventajas e inconvenientes de esta tecnología. Una vez aclarados los fundamentos tecnológicos que constituyen los Servicios Web Semánticos, se presenta una descripción completa de este nuevo campo, basado en la anotación semántica de las descripciones de las capacidades de los Servicios Web. En este apartado, se enumeran las diversas propuestas que han sido realizadas al organismo estandarizador del W3C para su consideración a la hora de consensuar una solución universal a este problema. Además, se describen algunos de los entornos de ejecución de Servicios Web Semánticos que se han desarrollado para demostrar la utilidad de cada una de las aproximaciones. Por último en esta sección, se relatan los problemas y carencias de esta tecnología, fundamentados básicamente en la necesidad de disponer de una entidad software, de orden superior, capaz de sacar provecho de las descripciones semánticas de los servicios para automatizar las tareas de gestión de los mismos.

Finalmente, se hace acopio de los problemas y dificultades que ponen trabas a las tecnologías de agentes y de Servicios Web Semánticos por separado, y se plantea como solución (y objetivo de esta tesis doctoral) *el desarrollo de una plataforma-marco de trabajo para la integración de la tecnología de agentes y la de Servicios Web Semánticos (haciendo uso de las ontologías) de forma que las ventajas aportadas por cada una de las mismas por separado puedan ser explotadas al máximo, mientras que sus problemas son aliviados*.

CAPÍTULO II. INTEGRACIÓN DE AGENTES INTELIGENTES Y SERVICIOS WEB SEMÁNTICOS – TRABAJO RELACIONADO

II.1. Introducción

Debido a los problemas que surgen de la aplicación por separado de las tecnologías de agentes y Servicios Web Semánticos mencionados en el capítulo anterior, numerosos investigadores han propuesto la aplicación conjunta de ambas tecnologías para el desarrollo de aplicaciones avanzadas (Hendler, 2001; McIlraith et al., 2001; Gibbins et al., 2003; Paolucci & Sycara, 2003; Buhler & Vidal, 2005; Masuoka et al., 2005; Gómez et al., 2006; Shafiq et al., 2006). Sin embargo, no existe todavía acuerdo en cómo se debe alcanzar este objetivo de integración de las tecnologías.

James Hendler (2001) describió por primera vez cómo los lenguajes ontológicos asociados a la Web Semántica podrían dar lugar a sistemas basados en agentes más potentes que fueran capaces de utilizar los servicios ofrecidos en la Web. Su trabajo constituyó en realidad el origen de lo que hoy se conocen como Servicios Web Semánticos. El autor propuso un método para describir el modo en que un agente debe de realizar la invocación de los servicios por medio de un lenguaje ontológico como DAML+OIL. Una vez que las características referentes a la invocación de un servicio están descritas semánticamente, los agentes serían capaces de determinar automáticamente la información necesaria para realizar la invocación de dicho servicio. Posteriormente, se ha propuesto una gran variedad de soluciones que se diferencian, en algunos casos, en matices conceptuales de gran envergadura. Blacoe y Portabella (2005) distinguen tres posibles escenarios en los que se pueden integrar lo que denominan "servicios basados en agentes" y "servicios basados en Web":

1. Los Servicios Web proporcionan los niveles más básicos de funcionalidad, mientras que los agentes proporcionan funcionalidad de alto nivel haciendo uso y combinando los Servicios Web disponibles, de forma que se consiguen funciones de valor añadido.

2. La comunicación en Servicios Web y agentes se hace equivalente, de forma que la distinción entre ambas tecnologías desaparece (p.ej. implementar agentes con tecnología de Servicios Web).
3. Ambos tipos de servicios permanecen separados creando un espacio de servicios heterogéneo en el que unos y otros puedan interoperar a través de gateways y procesos de traducción.

En este capítulo, se estudian algunos de los trabajos de investigación en los que se ha hecho uso de agentes y Servicios Web (semánticos) de un modo integrado. Se ofrecen detalles de cada una de estas propuestas y se destacan los inconvenientes de cada una de las mismas.

II.2. GODO (Goal-Oriented DiscOvery for Semantic Web Services)

El sistema GODO[17] (Gómez et al., 2004; Gómez et al., 2006) se puede definir como un agente software situado entre los distintos entornos de ejecución de Servicios Web Semánticos (p.ej. WSMX, METEOR-S, OWL-S Virtual Machine, etc.) y los usuarios que desean utilizar dicho sistema (ver Fig. 37). El objetivo último de este agente es evitar la necesidad de que los usuarios sean familiares con los formalismos propios de los entornos de ejecución y, de este modo, facilitarles su utilización. En particular, GODO tiene en cuenta el hecho de que la mayor parte de los entornos de ejecución deben recibir como entrada un documento especificado en un formalismo concreto (p.ej. un objetivo en formato WSML para WSMX, una descripción en OWL-S para la OWL-S Virtual Machine, etc.). Así, GODO transforma la entrada del usuario (en general a través de sentencias en lenguaje natural) en el formalismo adecuado para cada uno de los entornos de ejecución, e interactúa con estos de forma que libera al usuario de todo este arduo trabajo.

[17] GODO fue concebido en una colaboración entre Juan M. Gómez, actualmente profesor visitante de la Universidad Carlos III de Madrid (en el momento en que se desarrolló el sistema, era investigador de DERI –Digital Enterprise Research Institute), Mariano Rico, profesor ayudante de la Universidad Autónoma de Madrid, y el autor de esta tesis doctoral, durante una estancia en el centro de investigación de DERI en Galway (Irlanda).

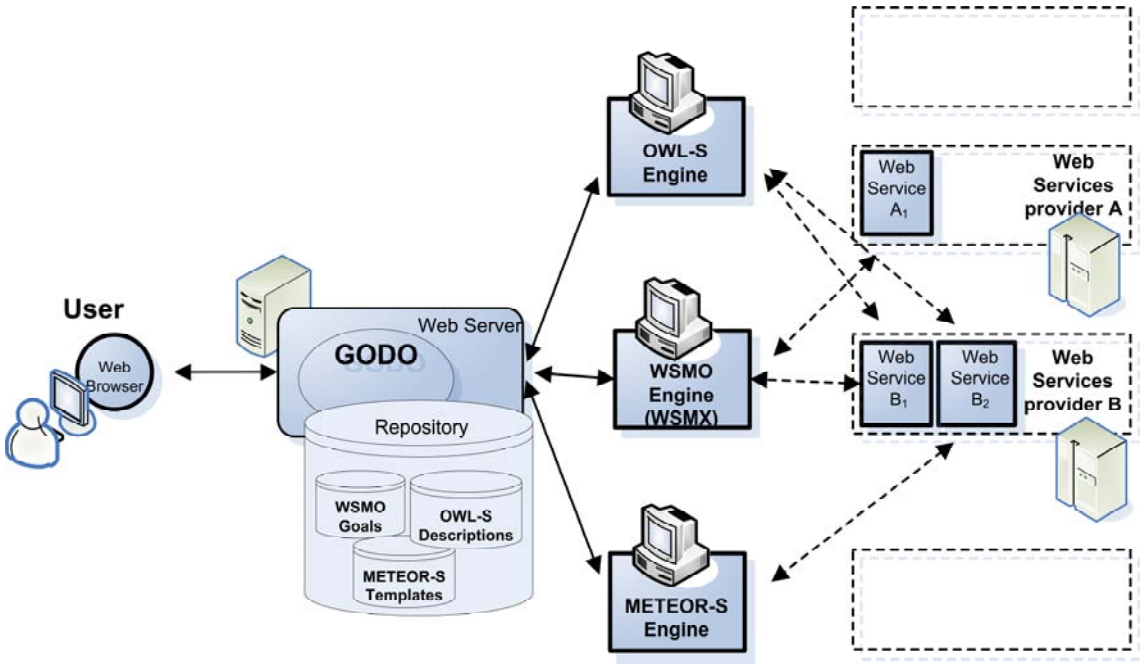


Fig. 37. GODO entre los entornos de ejecución y los usuarios (Gómez et al., 2006)

La arquitectura del sistema con los componentes principales del mismo se muestra en la siguiente figura (Fig. 38):

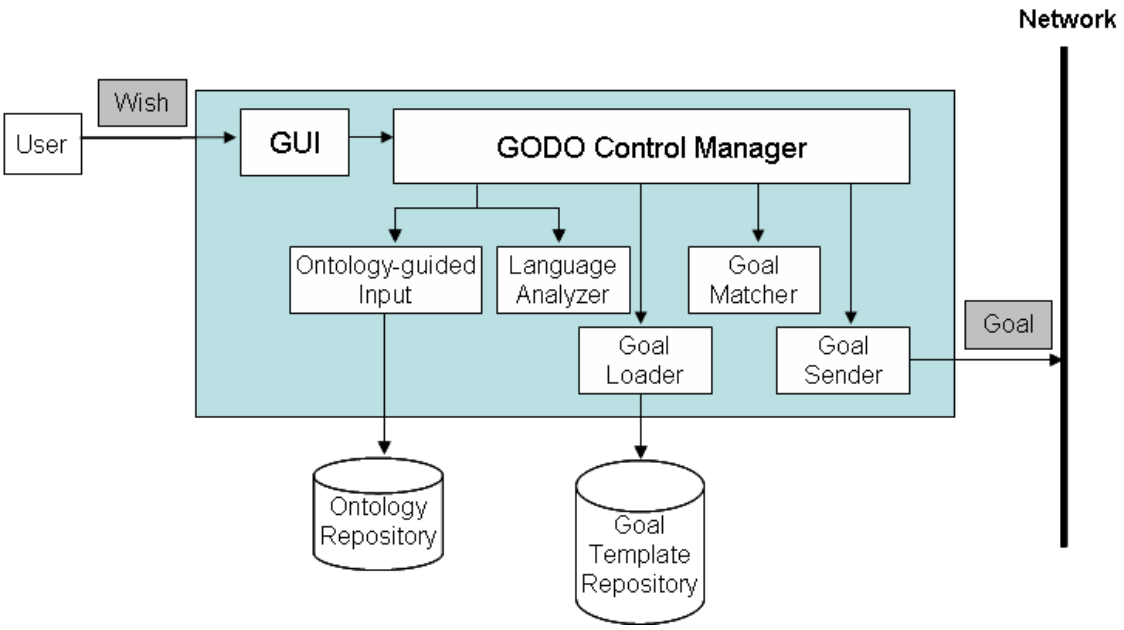


Fig. 38. Arquitectura de GODO (Gómez et al., 2006)

El *Control Manager* es el principal componente de la arquitectura, ya que controla las interacciones que tienen lugar entre los restantes componentes. Mediante la interfaz *GUI*, los usuarios pueden escribir sus consultas y éstas llegan al *Control Manager*. Tanto el *Ontology-guided Input* como el *Language Analyzer* gestionan la entrada del usuario. Por un lado, el *Language Analyzer* es el componente encargado de, mediante la aplicación de una herramienta de reconocimiento de lenguaje natural, generar una ontología a partir de una sentencia en lenguaje natural. Por otro lado, el *Ontology-guided Input* guía la entrada del usuario, presentándole distintas posibilidades disponibles de acuerdo con una ontología de base. Por su parte, el *Goal Loader* es el responsable de cargar todas las plantillas de objetivos disponibles que se encuentran almacenadas en un repositorio. El componente *Goal Matcher* se encarga de encontrar correspondencias entre la ontología que representa la necesidad definida por el usuario y las plantillas de objetivos que se encuentran en el repositorio. Por último, el *Goal Sender* envía de forma secuencial al entorno de ejecución adecuado los objetivos ya completados que se hayan considerado necesarios para satisfacer la necesidad definida por el usuario.

El flujo de ejecución del sistema es como sigue: 1) el usuario indica, a partir de una sentencia en lenguaje natural, lo que desea obtener por medio de la ejecución de servicios a través de la interfaz *GUI* (alternativamente, este primer paso puede consistir en la utilización del componente *Ontology-guided Input*, que limitaría la entrada del usuario y permitiría obviar el paso 2); 2) el *Control Manager* solicita al *Language Analyzer* que genere la ontología correspondiente a la sentencia de entrada; 3) una vez disponible la ontología con los deseos del usuario, el *Control Manager* insta al *Goal Loader* a cargar todas las plantillas de objetivos que se encuentren disponibles en el repositorio; 4) cuando se dispone de todas las plantillas de objetivos en los distintos formatos y la ontología con los deseos del usuario, el *Control Manager* transfiere estos dos elementos al *Goal Matcher* para que encuentre aquellas plantillas de objetivos que puedan satisfacer la necesidad definida por el usuario; 5) finalmente, las plantillas se completan con información proveniente de la ontología de necesidades del usuario y los objetivos son enviados por el *Goal Sender* de forma secuencial a los entornos de ejecución apropiados para su ejecución.

El mayor inconveniente asociado con el agente software GODO es que éste interacciona con infraestructuras de Servicios Web Semánticos, y no con éstos directamente. Debido a esto, muchas de las ventajas asociadas a los SMAs son desaprovechadas. En particular, no es posible realizar procesos de negociación entre las partes ni aplicar técnicas de agentes para tareas relacionadas con la gestión de Servicios Web Semánticos, como el descubrimiento, la composición y la invocación. En su lugar, deben implementarse soluciones *ad hoc* para todas estas tareas en la infraestructura, no beneficiándose así de las posibilidades que proporcionan en la actualidad las plataformas multi-agente.

II.3. SWF (Semantic Web Fred)

El proyecto SWF (http://swf.deri.at/) (Stollberg et al., 2004a; Stollberg et al., 2004b; Stollberg et al., 2005) combina la tecnología de agentes, ontologías y Servicios Web Semánticos para desarrollar un sistema que permite la realización de forma cooperativa y automatizada de tareas. Para esto, se identifica de forma unívoca la utilidad de cada una de estas tecnologías en el ámbito del proyecto:

- Las ontologías proporcionan una semántica muy expresiva y procesable por el ordenador que permite el intercambio de información semánticamente correcto entre aplicaciones.
- La tecnología de agentes permite la ejecución automatizada (sin necesidad de intervención humana) de tareas. Los agentes deben ser capaces de procesar una ontología para permitir el intercambio de información mejorado semánticamente y proporcionar apoyo para las tareas de descubrimiento, invocación y ejecución de Servicios Web.
- Por su parte, los Servicios Web (Semánticos) permiten la reusabilidad e interoperabilidad entre funcionalidad computacional.

En la siguiente figura, se presenta la arquitectura global del sistema desarrollado (ver Fig. 39):

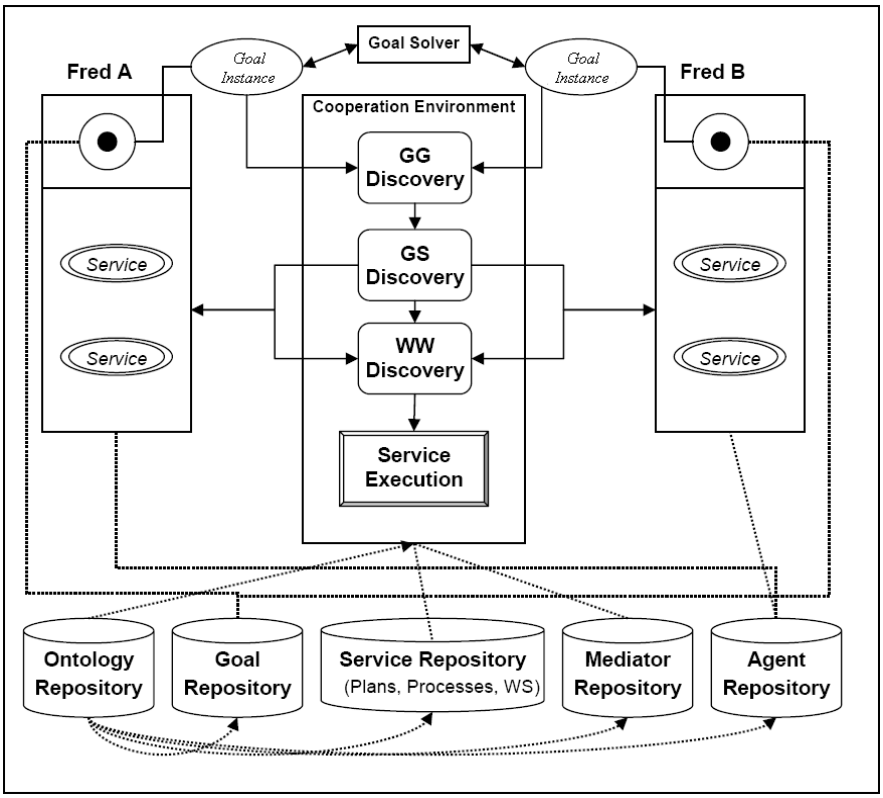


Fig. 39. Arquitectura general de SWF (Stollberg et al., 2004a)

Las ontologías en SWF se utilizan para definir la terminología de la aplicación. Se distinguen dos tipos de ontologías: las ontologías del dominio, las cuales definen los conceptos integrantes del dominio del discurso, y las ontologías del sistema que definen conceptos referentes a los componentes tecnológicos (objetivos y servicios) para permitir un procesado mejorado semánticamente. Otro de los componentes principales de la arquitectura de SWF son los objetivos (‘*goal*’). Un objetivo expresa un *deseo* (qué es lo que el usuario quiere conseguir) que un usuario delega a un agente (que en este proyecto se denomina *Fred*) para su resolución automatizada. Un objetivo define qué problema es el que se tiene que resolver sin especificar nada acerca del cómo ni del quién ni dónde se ha de resolver dicho problema. El objetivo asignado a un agente se utiliza para determinar qué otros agentes pueden ser potencialmente cooperadores en el desempeño de tal objetivo y qué servicios son necesarios para su consecución automática. El tercer componente principal de la arquitectura lo conforman los servicios. El concepto ‘servicio’ se utiliza para identificar todos los tipos de recursos de resolución de problemas que se encuentran disponibles en el sistema. En particular,

SWF distingue entre tres tipos de servicios: (1) *planes*, que identifican a programas Java internos; (2) *procesos*, se refieren a servicios que requieren múltiples pasos y donde cada actividad puede ser resuelta por un objetivo o por otro servicio, permitiendo, de este modo, la definición de servicios complejos y anidados; y (3) *Servicios Web externos*, un servicio Web, en el sentido 'tradicional' del concepto, que puede ser invocado accediendo a su descripción WSDL. Los repositorios constituyen el último componente de la plataforma. Como se puede apreciar en la figura, existen distintos tipos de repositorios: repositorio de ontologías, de agentes, de servicios, de objetivos y de mediadores.

Además de los componentes identificados, dentro del proyecto SWF se han elaborado diversos métodos y mecanismos relacionados con la consecución cooperativa de objetivos a través de servicios. De esta forma, existe un método para el descubrimiento Objetivo-Objetivo ('*GG Discovery*'), un primer paso hacia la resolución cooperativa de un objetivo, que detecta potenciales socios cooperativos determinando la compatibilidad de sus respectivos objetivos. Después, se hace necesario el descubrimiento Objetivo-Servicio ('*GS Discovery*'), que detecta los servicios adecuados para la resolución automatizada de objetivos de forma separada para cada socio identificado en la fase de descubrimiento Objetivo-Objetivo. El resultado de este método es un conjunto de servicios que un socio puede utilizar para la cooperación automatizada.

Finalmente, para determinar si es posible la elaboración de un plan que permita la consecución con éxito del objetivo inicial, se debe estudiar la compatibilidad entre los servicios que cada uno de los socios ha detectado mediante el mecanismo de descubrimiento Objetivo-Servicio. Para esto, se dispone de un nuevo método, denominado descubrimiento Servicio-Servicio ('*WW Discovery*'), que devuelve como resultado un modelo global de interacción de los servicios de los socios que permite la ejecución automatizada de la cooperación. Cuando los tres mecanismos de descubrimiento se han completado con éxito, se crea un "contrato de cooperación", que contiene la información necesaria para la resolución cooperativa del objetivo. Una vez establecido el contrato, los socios que forman el consorcio ejecutan los servicios de acuerdo con la información del contrato obteniendo el objetivo deseado.

SWF es una solución muy completa para la interacción agente-servicio, ya que proporciona los medios para realizar tareas tales como el descubrimiento, la composición y la invocación de servicios. Sin embargo, una de las mayores deficiencias de esta herramienta es su estrecha relación con la aproximación de Servicios Web Semánticos WSMO. De hecho, se considera el sistema SWF como una implementación de esta aproximación. Dado que no se ha alcanzado un estándar para esta tecnología, ligar una solución a una aproximación concreta limita su aplicabilidad en el futuro. Por otro lado, existen muchos aspectos que no son considerados en esta propuesta, principalmente desde el punto de vista de los agentes. No hay una diferenciación clara entre entidades proveedoras de servicios y entidades consumidoras. Además, los usuarios no tienen medios para establecer un conjunto de preferencias que podrían ser utilizadas en los procesos de descubrimiento y selección de servicios, así como en un posible proceso de negociación.

Por último, es interesante comentar el proceso de descubrimiento propuesto en el proyecto SWF. En particular, se observa la introducción de una etapa "descubrimiento Objetivo-Objetivo" que no es frecuente observar en las soluciones tradicionales a este problema. En esta fase, el sistema trata de encontrar los agentes que poseen objetivos complementarios y, por tanto, cumplen las condiciones idóneas para que tenga sentido la cooperación. En las aproximaciones más tradicionales, donde se distingue con mayor precisión los roles de proveedores de servicios y consumidores de servicios, esta fase es obviada asumiendo que, si un servicio satisface un objetivo, la entidad que proporciona dicho servicio es compatible con la que desea utilizarlo.

II.4. Servicios Web y Comportamientos de Agente

Otro trabajo donde el objetivo es utilizar de forma conjunta las tecnologías de agente y de Servicios Web es el presentado por P. A. Buhler y J. M. Vidal (Buhler & Vidal, 2003; Vidal et al., 2004; Buhler & Vidal, 2005). Aquí, los autores destacan el comportamiento pasivo de los Servicios Web y proponen envolverlos en agentes proactivos para evitar este limitante. Este trabajo se basa en el estudio de Huhns (2002) referente a las diferencias existentes entre agentes y Servicios Web. En este estudio, se

indica que, a pesar de compartir algunos puntos similares, los agentes superan a los Servicios Web en diversas características: i) un Servicio Web sólo sabe de su existencia mientras que los agentes son en general conscientes de la existencia de otros agentes y de los usuarios; ii) los Servicios Web no están diseñados para utilizar y entender ontologías; iii) los Servicios Web son entidades pasivas, a diferencia de la actitud proactiva y autónoma de los agentes.

El objetivo de esta propuesta es la de conseguir el desarrollo de un '*workflow*' (flujo de trabajo) dispuesto dinámicamente utilizando agentes inteligentes que forman un SMA y que hacen uso, en forma de comportamientos externos, de Servicios Web (semánticos). La idea es, por tanto, añadir la propiedad de proactividad propia de los SMAs a un entorno o flujo de trabajo compuesto por Servicios Web, de forma que éste se transforme en un mecanismo cooperativo de resolución de problemas. Para esto, se parte de la descripción de un proceso en BPLE4WS (*Business Process Execution Language for Web Services*) que impone un orden inicial en la ejecución de los distintos servicios para un propósito determinado. Dado que BPEL4WS describe las relaciones entre los Servicios Web en el flujo de trabajo, los agentes que representan a dichos servicios sabrán sus relaciones a priori. La estrategia propuesta por los autores es la de representar el flujo de trabajo mediante Redes de Petri, de forma que cada agente (en el SMA) represente una funcionalidad diferente, consiguiendo así un flujo de trabajo distribuido.

En la siguiente figura (ver Fig. 40) se presenta un ejemplo donde una organización multi-agente se dispone de forma que es capaz de realizar un flujo de trabajo determinado. En esta figura, se puede comprobar la existencia de distintos mecanismos de comunicación: (1) los agentes se comunican entre sí a través del lenguaje FIPA-ACL; (2) para la comunicación con los servicios, los agentes hacen uso de mensajes SOAP; y (3) el espacio de variables del proceso BPEL4WS se accede a través de los protocolos o interfaces que proporcione el espacio de datos utilizado.

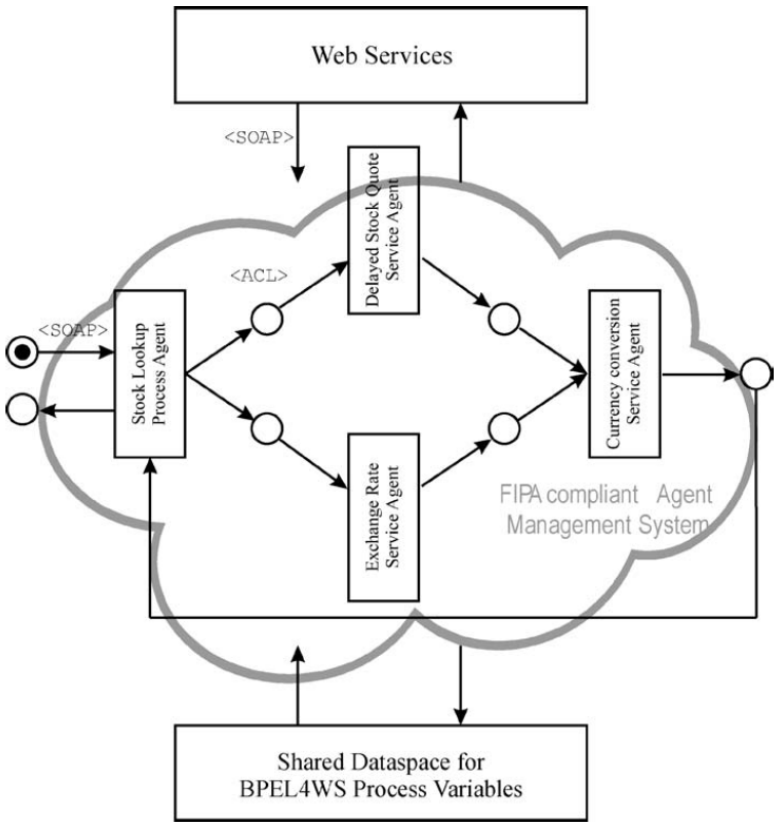


Fig. 40. Arquitectura del SMA para un flujo de trabajo de ejemplo (Buhler & Vidal, 2005)

Esta propuesta posee dos limitaciones importantes. En primer lugar, aunque los autores consideran que el surgimiento de la Web Semántica y la descripción semántica de los Servicios Web puede facilitar la gestión de flujos de trabajo basados en agentes en entornos dinámicos y en tiempo real, no se plantean, por el momento, la utilización de tales técnicas, por considerar poco maduro el estado de la tecnología. De esta forma, no se hace uso real de Servicios Web Semánticos y, por tanto, no es posible sacar provecho de los beneficios asociados a su utilización. La segunda mayor desventaja de esta aproximación, provocada como consecuencia del primer inconveniente mencionado, es la definición estática de los flujos de trabajo. Uno de los problemas asociados a la no utilización de la semántica, es la imposibilidad de que los agentes sean capaces de realizar de forma automática las tareas de descubrimiento, selección, composición e invocación de tales servicios. Al contrario, cada agente está asociado a un único Servicio Web para el que actúa de envoltorio. Así, cuando se intenta invocar un servicio, el agente es capaz de realizar tareas de alto nivel o "inteligentes" que

mejoran la funcionalidad proporcionada por el Servicio Web, pero las relaciones entre los agentes están predeterminadas a priori, no pudiendo generar flujos de trabajo dinámicos.

II.5. Task Computing

Task Computing (Song et al., 2004; Masuoka et al., 2005) es un marco de trabajo orientado al usuario que permite a usuarios finales llevar a cabo tareas complejas en entornos ricos desde el punto de vista de aplicaciones, dispositivos y servicios. La definición propuesta por los autores para el término "*Task Computing*" es la de una computación que rellena el hueco existente entre las tareas (qué es lo que los usuarios quieren hacer) y los servicios (funcionalidades que están disponibles para los usuarios). Para hacer esto, el marco de trabajo hace uso de la tecnología de Servicios Web Semánticos.

La arquitectura del sistema se muestra en la siguiente figura (ver Fig. 41). El entorno está compuesto de cuatro capas. En la capa más profunda de la arquitectura, "*Realization Layer*", se encuentran los componentes (como dispositivos, servicios, aplicaciones, contenido) que proporcionan realmente la funcionalidad que se ofrece al usuario final. En la siguiente capa, "*Service Layer*", se sitúan los servicios que permiten acceder a los recursos de la capa inferior. Estos servicios pueden ser tomados como interfaces que permiten acceder a la funcionalidad proporcionada por dichos recursos. Cada servicio tiene asociado, al menos, una descripción semántica que puede haber sido creada dinámicamente junto con el servicio. La descripción semántica se incluye con el propósito de evitar que el usuario se enfrente directamente con la complejidad de las fuentes de funcionalidad que están por debajo y hacer más sencillo para éste el empleo de tales fuentes para la consecución de tareas complejas.

La capa inmediatamente superior, "*Middleware Layer*", es la que contiene los mecanismos y técnicas capaces de manejar estos servicios y su semántica. Tareas como el descubrimiento de servicios, determinar cómo éstos se pueden componer, su invocación, la monitorización de la ejecución y otra serie de tareas de gestión (p.ej. creación y publicación de servicios descritos semánticamente) se realizan en este nivel

de la arquitectura. Finalmente, la capa superior de la arquitectura, "*Presentation Layer*", constituye una de las partes más importantes de esta aproximación. Tiene como objetivo presentar al usuario las tareas permisibles, abstrayéndole de la complejidad de las capas subyacentes. Los autores han desarrollado una interfaz multi-modal para la interacción con los usuarios donde se mezclan elementos de voz, textuales y gráficos. En su conjunto, esta capa de presentación presenta al usuario un entorno donde la funcionalidad es transitoria y se puede crear dinámicamente, así como componerse en tiempo real, para realizar las tareas indicadas por los usuarios.

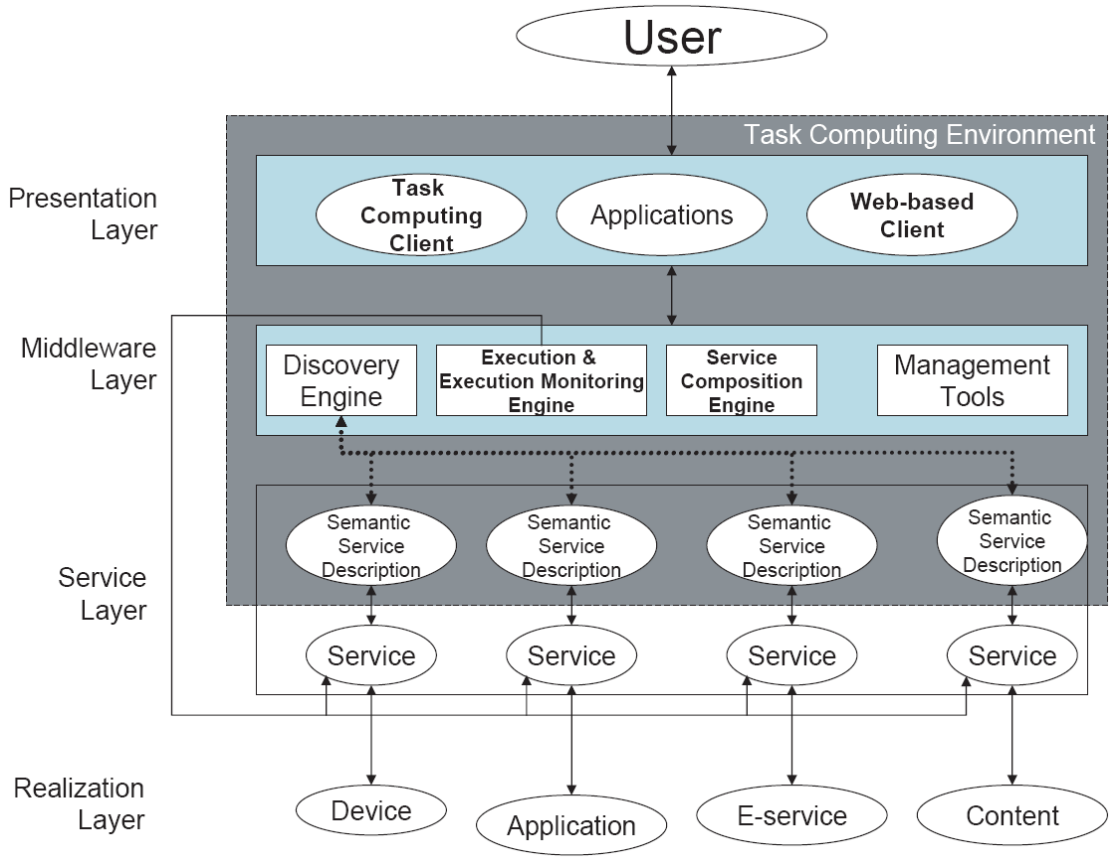


Fig. 41. Arquitectura del entorno de trabajo Task Computing (Song et al., 2004)

Para probar la utilidad de este marco de trabajo, sus autores lo aplicaron en entornos ubicuos, donde numerosos recursos (proyectores, impresoras, etc.) se ponían a la disposición de los usuarios para su explotación (ver Fig. 42). En este dominio, los autores destacan la utilidad del Task Computing a la hora de facilitar al usuario distintos tipos de tareas, como la de compartir y mostrar una presentación, programar una

presentación para el futuro, consultar e imprimir las direcciones para llegar al aeropuerto, etc.

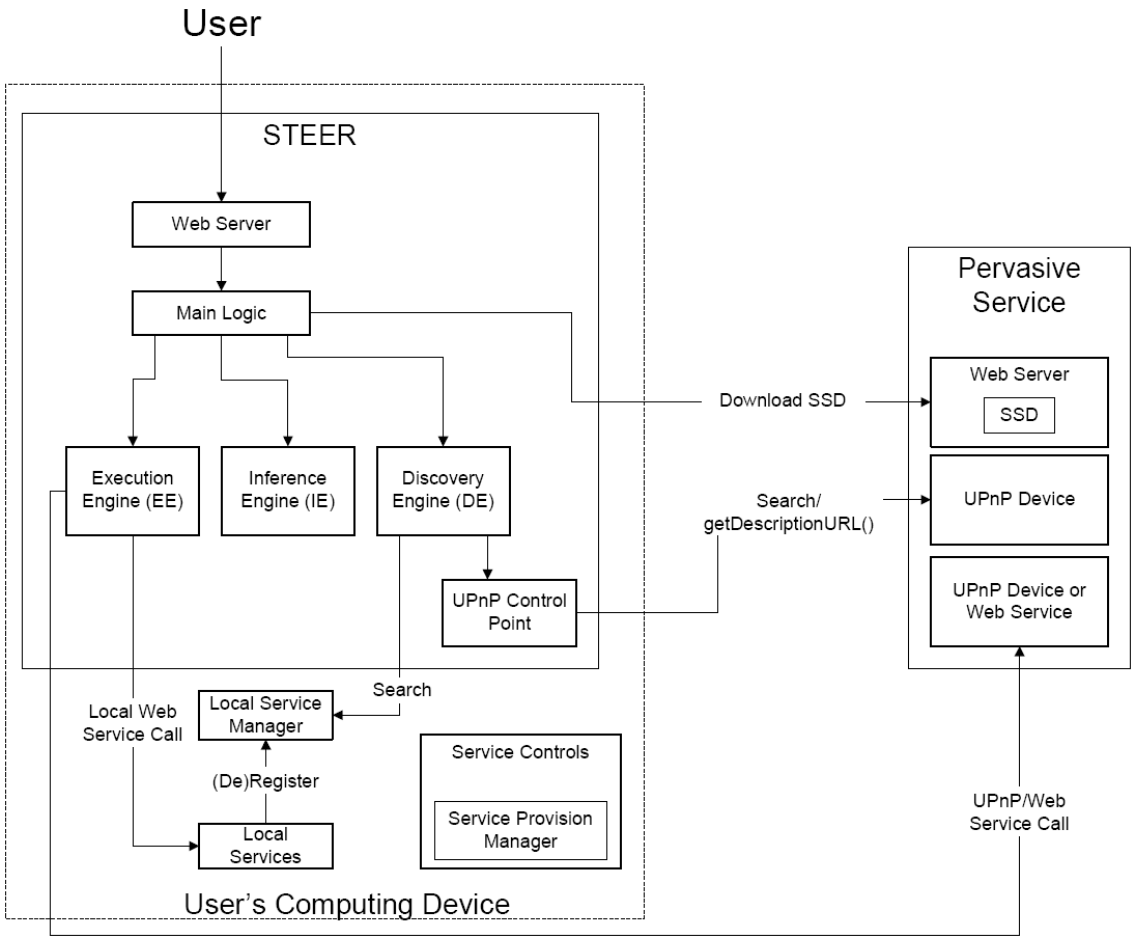


Fig. 42. Task Computing en computación ubicua (Masuoka et al., 2003)

La visión del Task Computing es bastante ambiciosa, integrando en un mismo entorno recursos tanto físicos como funcionales, aplicaciones, Servicios Web, contenido y dispositivos hardware. Sin embargo, esta visión se ve mermada por la carencia en la arquitectura de una entidad que posea las características de los agentes inteligentes (autonomía, pro-actividad, aprendizaje, etc.), lo cual limita las posibilidades que, de otro modo, podrían incorporarse al entorno. Uno de los puntos donde esto se comprueba es en la decisión tomada por los diseñadores de limitar la autonomía del sistema en beneficio de la interactividad con el usuario. En particular, se deja al usuario final la tarea de componer los servicios disponibles. La descripción semántica de los servicios, que comúnmente se dirige a entidades software autónomas para su aprovechamiento sin

necesidad de intervención humana, se utiliza entonces para ofrecer a los usuarios una visión más abstracta de la funcionalidad de los mismos de forma que les sea más fácil la manipulación de los mismos. Por otro lado, si bien existen las tareas más convencionales relacionadas con la manipulación de servicios como descubrimiento, composición, invocación y monitorización, en ningún momento se especifica otro elemento de igual importancia en estos entornos, como es el de la selección de servicios. Para esta tarea, un proceso de negociación puede ayudar a determinar el servicio, de entre todos que satisfacen una necesidad, que ofrece las mejores condiciones para su ejecución. En las últimas décadas, se han realizado extensas investigaciones relacionadas con la negociación en entornos multi-agente. El Task Computing podría beneficiarse, si hiciera uso de la tecnología de agentes, de las soluciones ya propuestas en este sentido. Finalmente, se ha comprobado la utilidad de la plataforma sobre un dominio de computación ubicua en el ámbito de una oficina, donde son muy limitados el conjunto de servicios disponibles. Una solución de estas características podría tener mayores problemas aplicado en otros entornos más abiertos como comercio electrónico, organizaciones virtuales, negocio electrónico, etc., donde se multiplica el número de servicios disponibles y, por tanto, presentar al usuario esta lista de servicios se vuelve intratable.

II.6. Otras metodologías

Existen otras metodologías que, si bien no están tan relacionadas con la propuesta en esta tesis, tratan de explotar por medio de la tecnología de agentes las nuevas posibilidades surgidas tras la emergencia de la Web Semántica. En este apartado se estudian, en menor grado de detalle, algunas de ellas.

II.6.1. Interoperabilidad entre Agentes Software y Servicios Web

Los marcos de trabajo y entornos descritos en los apartados anteriores se podrían clasificar, en su mayoría, dentro de la categoría 1 de acuerdo con la tipificación propuesta por Blacoe y Portabella (2005) (los Servicios Web proporcionan la funcionalidad más básica y, en un nivel superior, los agentes capaces de generar servicios de valor añadido). Se han elaborado, por el contrario, soluciones que

pretenden permitir una interoperabilidad entre agentes y Servicios Web sin barreras ni diferencias de nivel (tipo 3 de la mentada clasificación). Esta es la finalidad de uno de los grupos de trabajo dentro de FIPA denominado "*Agents and Web Services Interoperability Working Group (AWSI WG)*" (http://www.fipa.org/subgroups/AWSI-WG.html). Además del objetivo de rellenar el vacío de comunicación existente entre agentes y Servicios Web de forma que los agentes sean capaces de localizar, negociar e interactuar con Servicios Web y viceversa, el grupo de trabajo AWSI WG persigue las siguientes metas (AWSI WG, 2005):

1. Mantener la compatibilidad con las especificaciones de FIPA previas siempre que sea posible.
2. Conseguir añadir valor a la tecnología de Servicios Web introduciendo conceptos propios de agentes como diálogos, semántica de comunicación y protocolos de interacción.
3. No modificar los estándares actuales en relación a Servicios Web ni tener impacto sobre las implementaciones de referencia.
4. Hacer uso de tecnologías de la Web Semántica para introducir conceptos de agentes en Servicios Web.

Con estos propósitos en mente, se planteó la necesidad de un componente software o middleware situado entre la tecnología de agentes y la de Servicios Web que sirva de puente entre ambas reconciliando las diferencias entres sus respectivas especificaciones (ver Fig. 43).

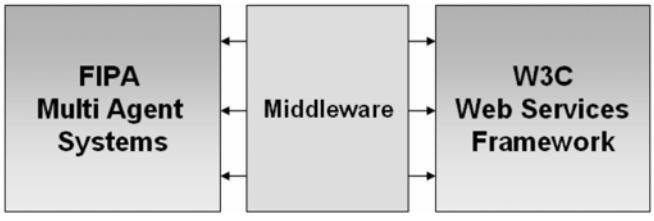


Fig. 43. Middleware que sirve de puente entre agentes y Servicios Web (Shafiq et al., 2005; Shafiq et al., 2006)

Los problemas con los que debe de enfrentarse una solución de esta índole se basan en tres aspectos fundamentales de las tecnologías: utilización de diferentes protocolos de comunicación (ACL vs. SOAP), lenguajes de descripción de servicios (DF-Agent-Description vs. WSDL) y registros de servicios (DF vs. UDDI). Por tanto, se han

dispuesto mecanismos para tratar cada uno de estos problemas por separado, y se han integrado en un mismo componente software denominado "*AgentWeb Gateway*" (ver Fig. 44).

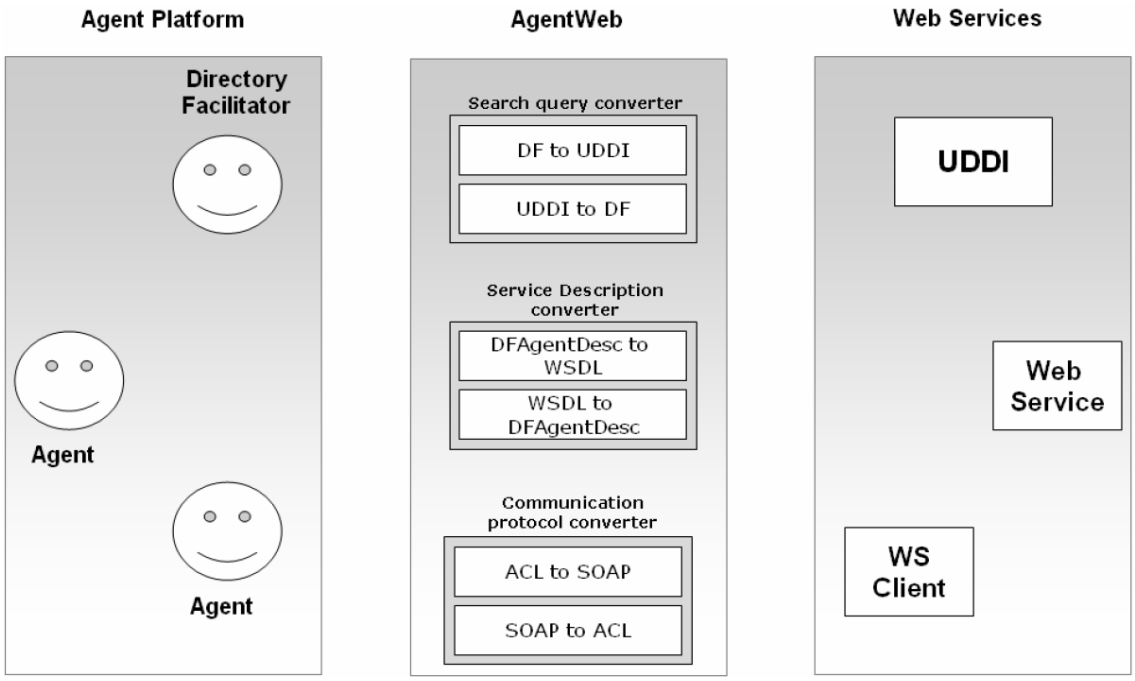


Fig. 44. Detalle de la estructura interna de "AgentWeb" (Shafiq et al., 2005; Shafiq et al., 2006)

En la figura, se pueden apreciar los tres conversores existentes: (i) conversor para descubrimiento de servicios (de una consulta de búsqueda en UDDI a una consulta de búsqueda en DF y viceversa); (ii) conversor para descripción de servicios (de una descripción en WSDL a una descripción en DF-Agent-Description y viceversa); y (iii) conversor del protocolo de comunicación (de SOAP a ACL y viceversa) –se puede encontrar la descripción detallada de cada uno de estos mecanismos de conversión en (Shafiq et al., 2006). Así, con la ayuda del "AgentWeb Gateway" y sin la necesidad de modificar ninguna de las especificaciones de FIPA (en cuanto a agentes) ni del W3C (en cuanto a Servicios Web), se puede conseguir la siguiente funcionalidad (Shafiq et al., 2005):

1. Agentes software pueden descubrir Servicios Web en un registro UDDI.
2. Agentes software pueden publicar sus servicios en un registro UDDI.
3. Agentes software pueden invocar Servicios Web.

4. Clientes de Servicios Web pueden descubrir agentes software en el DF ("*Directory Facilitator*") de una plataforma de agentes.
5. Los Servicios Web pueden ser publicados en el DF de una plataforma de agentes.
6. Clientes de Servicios Web pueden invocar agentes software.

Con esta solución, se consigue satisfacer la necesidad de combinar agentes y Servicios Web en un entorno integrado sin tener que modificar las especificaciones de FIPA y, por tanto, sin perder la compatibilidad con los SMAs preexistentes. Con todo, existe un problema con esta aproximación, de naturaleza conceptual más que tecnológica. En este sentido, las tecnologías de agentes y la de Servicios Web surgieron con distinto propósito y, a pesar de que en algunas ocasiones puedan superponerse, en la mayoría de los casos ofrecen una funcionalidad que se complementa. Si bien los agentes inteligentes surgieron con el objetivo de lograr aplicaciones en las que entidades software pudieran decidir por sí mismas qué hacer para satisfacer los objetivos de diseño imitando el comportamiento humano, los Servicios Web tienen el propósito de transformar la Web de una fuente de contenidos a una fuente de funcionalidad que pueda ser aprovechada tanto por usuarios humanos como por componentes software autónomos (esto es, agentes software). Siguiendo esta visión, parece evidente la complementariedad de las propuestas y las diferencias conceptuales de ambas tecnologías que derivan en la idoneidad de tener SMAs que hagan uso de la funcionalidad ofrecida por los Servicios Web para satisfacer las necesidades de los usuarios (ver Fig. 45).

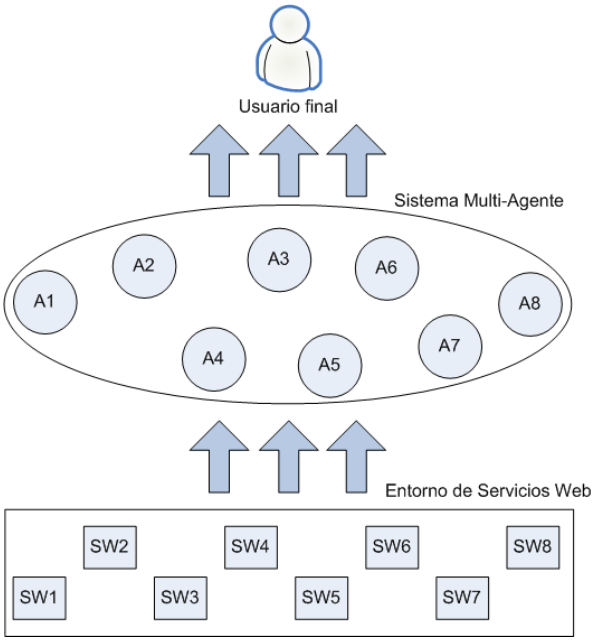


Fig. 45. Interacción Agente-Servicio Web a distinto nivel

Por último, el "*Web Services Integration Gateway*" (WSIG) (Greenwood et al., 2005) es una solución similar a la que busca el grupo de trabajo AWSI WG, desarrollada para su aplicación en el marco de trabajo para desarrollo de SMAs, JADE, y que permite a los agentes descubrir e invocar Servicios Web.

II.6.2. Servicios Web para Implementar Sistemas Multi-Agente

La segunda opción que contemplan Blacoe y Portabella (2005) en su clasificación de cómo integrar agentes y Servicios Web es la que abre la posibilidad a que agentes y Servicios Web se vuelvan una misma entidad. El modo en que generalmente se lleva esto a cabo, es mediante la utilización de los Servicios Web como infraestructura tecnológica de base para el desarrollo de SMAs. Estas aproximaciones tratan de incorporar características y comportamientos típicamente asociados al mundo de agentes e Inteligencia Artificial (p.ej., autonomía, comportamiento dirigido por objetivos, negociación flexible, etc.) dentro de Servicios Web. Así, en general se establece que un Servicio Web encapsula el comportamiento de un agente racional. Este es el caso de la solución propuesta en (Walton, 2005). El autor en este trabajo destaca las ventajas de la tecnología de Servicios Web y propone superar los inconvenientes de la misma (p.ej. inflexibilidad), adhiriendo técnicas procedentes de la tecnología de

agentes. De esta forma, se puede conseguir un entorno que posea las características de un SMA sin la necesidad, para ello, de hacer uso de marcos de trabajo específicos para el desarrollo de este tipo de sistemas como FIPA-OS y JADE. Además, de este modo, es posible conseguir que diversos Servicios Web puedan agruparse y actuar de forma coordinada para la resolución de tareas conjuntas, imposible hasta el momento por la naturaleza inflexible de la arquitectura de los Servicios Web. A continuación, se muestra un ejemplo de entorno multi-agente tal y como lo concibe el autor (ver Fig. 46). En la figura, se puede apreciar la división de un agente en dos partes: un "*stub*", que se encarga de los procesos relacionados con la comunicación entre agentes, y un "*body*", que encapsula el proceso de razonamiento del agente.

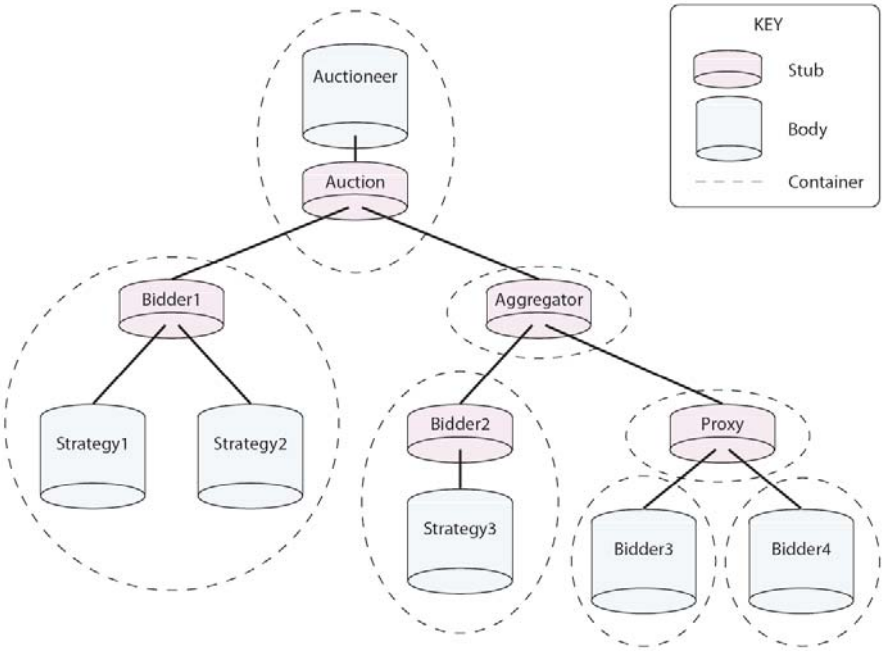


Fig. 46. Agentes compuestos por "*stub*" y "*body*" (Walton, 2005)

La principal aportación de la solución propuesta por este autor está dentro del campo de composición de Servicios Web. Son dos las principales limitaciones de la tecnología de Servicios Web en relación a la composición: (1) utilización de un mecanismo de comunicación de un sentido y sin estado; y (2) se precisa definir la topología del sistema antes de poder comenzar su ejecución. Partiendo de esta base, se propone la utilización de protocolos de interacción entre agentes (AIP, '*Agent Interaction Protocol*') para subsanar estos problemas. Por medio de este mecanismo, es posible representar patrones

de interacción complejos entre grupos de agentes así como permitir abstraerse lo suficiente como para no necesitar conocer a los agentes a priori. Por último, el autor destaca la mejora que supone esta aproximación sobre aquellas técnicas que permiten la interoperabilidad entre estas tecnologías por medio de puentes o *gateways*, justificada en la posibilidad de tratar a los Servicios Web como agentes propiamente.

Una solución similar es lo que Paolucci y Sycara (2003) denominaron Servicios Web Semánticos autónomos ("*Autonomous Semantic Web Services*"). Con este nombre, se identifican los Servicios Web que actúan como agentes independientes, autónomos y dirigidos por objetivos, capaces de seleccionar a otros agentes con los que interaccionar y negociar de forma flexible sus modelos de interacción. Estos servicios hacen uso de ontologías y páginas Web anotadas semánticamente para llevar a cabo sus tareas y transacciones. En realidad, lo que los autores presentan en este trabajo es muy parecido a lo propuesto por Walton, ya que se trata de incorporar una nueva capa al Servicio Web tradicional que incorpora las propiedades y características que se le suponen a un agente inteligente (capacidad de razonamiento y desarrollo de tareas basadas en conocimiento). En este caso, la capa adicional hace uso de la descripción semántica de los otros servicios (en DAML-S, predecesor de OWL-S) para encontrar aquellos con los que desea interaccionar y para poder invocar a los mismos. En la siguiente figura (ver Fig. 47), se muestran los módulos de que se compone esta nueva capa.

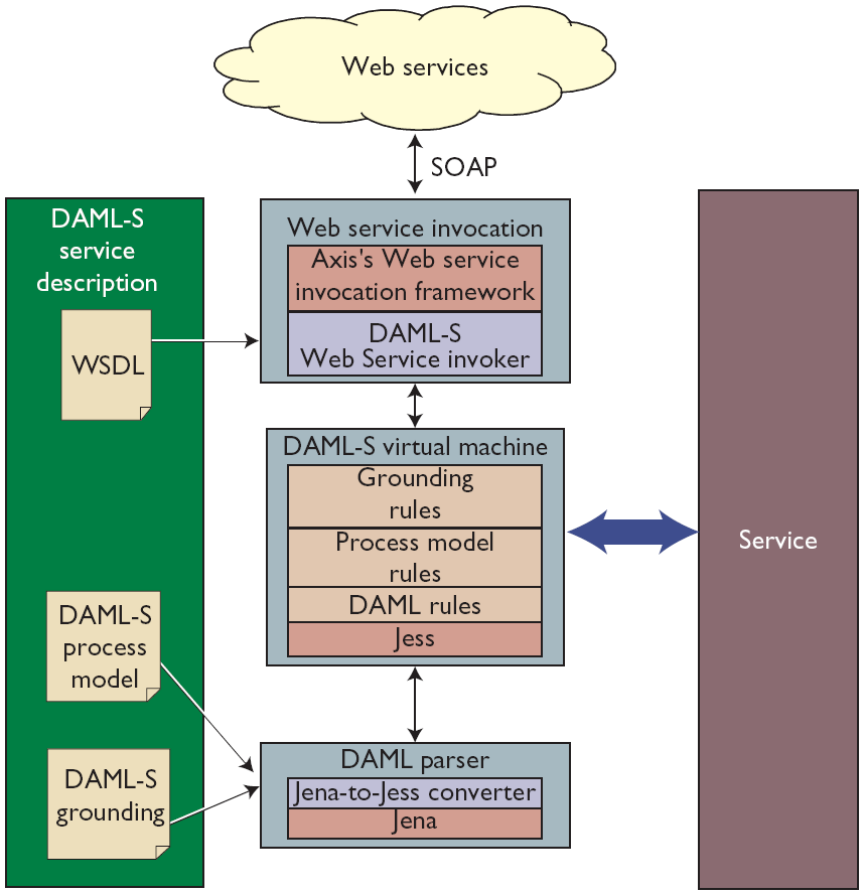


Fig. 47. Estructura de un "Servicio Web Semántico autónomo" (Paolucci & Sycara, 2003)

El servicio utiliza la información disponible sobre otros servicios (su descripción semántica) para interaccionar con ellos. Los tres módulos de control centrales se encargan de controlar esta interacción, mientras que la funcionalidad propia del servicio se encuentra disponible separadamente (parte derecha de la figura).

El principal problema asociado a este tipo de soluciones es que, tal y como se ha comprobado en las dos anteriores, se ignoran por completo todos los estándares relacionados con la tecnología de agentes y se han de implementar mecanismos *ad hoc* para el desarrollo de tareas como la negociación que, haciendo uso de los marcos de trabajo que satisfacen el estándar FIPA, ya estaría disponible a priori. En el caso particular de la segunda solución presentada, sólo se aprovechan las propiedades de la tecnología de agentes para el descubrimiento e invocación de otros servicios, dejando de lado otras muchas posibilidades que también podrían ser interesantes a la hora de

ofrecer un servicio (aprendizaje, autonomía y funcionamiento basado en objetivos, reactividad, etc.).

II.6.3. Agentes Inteligentes y la Web Semántica

Hasta este momento, a pesar de la carencia de estructura de la información en la Web actual, los agentes software han sido capaces de hacer uso de este tipo de información para actuar en lugar de los usuarios a los que representan y ayudar a los mismos en el desarrollo de tareas repetitivas y que requieren mucho tiempo. Así, por ejemplo, muchos sistemas de agentes registran los hábitos de navegación de los usuarios conforme éstos se desplazan de unas páginas a otras en la Web. Una vez disponen de un registro histórico con esta información, los agentes son capaces de sugerir a los usuarios nuevas páginas que les pueden resultar interesantes. En otros casos, los agentes pueden automatizar los accesos a las fuentes de información, replicando las consultas de los usuarios y analizando posteriormente las páginas con los resultados. Sin embargo, en todos estos casos, los agentes software deben de utilizar técnicas de reconocimiento de palabras clave o herramientas de reconocimiento de lenguaje natural para ser capaces de entender el contenido de las páginas Web.

Entre los principales objetivos que persigue la Web Semántica, se encuentra el de mejorar el modo en que los agentes software navegan y utilizan la información disponible en Internet. Con este propósito, a través de la Web Semántica se suministra un tipo de métodos estructurados de representación que permite expresar conceptos y relaciones por medio de ontologías. Ya antes de que el nombre de "Web Semántica" fuera acuñado por Berners-Lee y sus colegas, algunos investigadores advirtieron la utilidad y el beneficio que podrían llegar a tener los agentes Web que interaccionaran con contenido Web anotado con conocimiento semántico (Luke et al., 1997). En este trabajo, los autores describen SHOE ('*Simple HTML Ontology Extensions*'), predecesor de los lenguajes ontológicos para la Web Semántica actuales, y destacan los beneficios de describir las páginas Web de forma semántica. En particular, se centran en las ventajas que puede conllevar el uso de la información semántica por parte de agentes software que realizan búsquedas en la Web.

De igual forma, cuando los cimientos de la Web Semántica fueron asentados en el famoso artículo de Berners-Lee et al. (2001), los autores ya afirmaron que el potencial de la Web Semántica se podrá explotar al completo cuando agentes software sean capaces de recoger información de distintas fuentes, procesarla e intercambiar los resultados con otros programas haciendo uso del contenido semántico. En resumidas cuentas, gracias a la Web Semántica, los agentes pueden analizar las páginas Web para extraer la información relevante de las mismas. Asimismo, son capaces de entender y razonar sobre esta información y utilizarla para satisfacer las necesidades de los usuarios y proporcionar la ayuda que estos requieran.

Con el propósito de probar la validez de esta visión, se han desarrollado varios proyectos y numerosas herramientas que, teniendo como base la tecnología de agentes, hacen uso de las anotaciones semánticas de las páginas Web, tal y como promulga la Web Semántica. Entre los proyectos, se pueden destacar los siguientes: *RETSINA* (‘*Reusable Environment to Task-Structured Intelligent Networked Agents*’) *Calendar Agent*, que planifica la agenda de un usuario (Payne et al., 2002a; Payne et al., 2002b); *SERSE*, un SMA que puede realizar búsquedas sobre la Web Semántica (Tamma et al., 2004); e *ITTALKS*, un sistema basado en agentes y en la Web que permite la notificación automática e inteligente de charlas relacionadas con las tecnologías de la información (Cost et al., 2002).

Sin embargo, todavía es necesario solventar una serie de dificultades para que la utilización conjunta de estas tecnologías sea aplicable en entornos reales. En particular, uno de los principales problemas en relación a la Web Semántica es el uso de diversas ontologías al mismo tiempo. Dado que los desarrolladores elaboran las ontologías con vistas a la aplicación en la que se pretende utilizar, es muy común encontrarse con diversos modelos ontológicos para el mismo dominio de aplicación. El conflicto surge cuando se pretende que varias entidades software, haciendo uso de distintas ontologías para un mismo dominio, se comuniquen entre sí e interaccionen para conseguir sus respectivos objetivos. En este tipo de situaciones se hace necesario un mecanismo mediante el cual las diferencias, tanto sintácticas como semánticas, se reconcilien. Se han propuesto numerosas soluciones a este problema, como el mapeo (‘*mapping*’) y la fusión (‘*merging*’). La idea detrás de estas soluciones es la de encontrar, de forma

automática, los conceptos, atributos y relaciones que tengan algún tipo de correspondencia entre las ontologías que entran en conflicto.

II.6.4. Entornos de Ejecución de Servicios Web Semánticos

Tal y como se describió en detalle en el capítulo anterior, para algunas de las metodologías existentes en la actualidad para la anotación semántica de las capacidades de los Servicios Web, se han desarrollado herramientas capaces de hacer un uso efectivo de las anotaciones para efectuar las tareas pertinentes sobre los servicios publicados (esto es, descubrimiento, selección, composición, invocación y monitorización), permitiendo, de este modo, el desarrollo de aplicaciones que aprovechen las características de dinamismo de este tipo de entornos. OWL-S VM (Paolucci et al., 2003a), WSMX (WSMX, 2005), METEOR-S (Verma et al., 2005) e IRS-III (Cabral et al., 2006) son sólo algunos ejemplos de lo que se denominan *entornos de ejecución de Servicios Web Semánticos*.

Estas aplicaciones, a diferencia de las herramientas y soluciones presentadas hasta el momento, no hacen uso de la tecnología de agentes para explotar el contenido semántico disponible y manejar los servicios provistos. En su lugar, estos entornos de ejecución están basados en componentes software que, en algunos casos, se despliegan como Servicios Web, constituyendo aplicaciones débilmente acopladas. A pesar de las ventajas que esto conlleva, son numerosas las trabas que supone el ignorar los avances producidos en el campo de la Inteligencia Artificial y, en concreto, en lo relacionado con la tecnología de agentes y los SMAs. En primer lugar, se está desaprovechando la riqueza de las aplicaciones concebidas para el desarrollo de SMAs que cumplen los estándares establecidos por la comunidad investigadora. Nos referimos, naturalmente, a las plataformas multi-agente que, con distinto grado de madurez, se encuentran disponibles para la elaboración de sistemas inteligentes con capacidades comunicativas especiales, como las necesarias en los entornos de ejecución de Servicios Web Semánticos. Estas plataformas dan soporte al desarrollo de aplicaciones donde entidades individuales que poseen distintas habilidades y conocimientos se ejecutan sobre un mismo entorno, ya sea para colaborar en la consecución de un objetivo común, o competir para la obtención de recursos escasos.

En definitiva, se puede decir que las plataformas multi-agente permiten el desarrollo rápido y eficaz de sistemas basados en conocimiento donde entidades autónomas que asumen distintos roles dentro del sistema y tienen acceso a distintas formas de conocimiento que les permiten desarrollar tareas inteligentes (e incluso incorporar procesos de aprendizaje), al tiempo que son capaces de colaborar mediante el uso de protocolos de negociación. Algunas de estas características proporcionadas por los SMAs (p.ej., gestión del conocimiento, procesos comunicativos entre módulos del sistema) han sido incorporadas *ad hoc* a los entornos de ejecución de Servicios Web Semánticos desarrollados hasta el momento, mientras que otras características (como la capacidad de aprendizaje o los protocolos de negociación), que pueden mejorar la funcionalidad de estos sistemas, aún no han sido incorporadas a su núcleo funcional. Numerosos son, además, los algoritmos y técnicas desarrollados para algunas tareas de los SMAs cuyo uso puede ser provechoso para los entornos de ejecución, como son la publicación y registro de servicios ofrecidos, el descubrimiento de entidades que ofrecen los servicios deseados, la composición de servicios para consecución de un objetivo, etc.

II.7. Problemas de las soluciones disponibles actualmente

Como se ha señalado en los apartados anteriores, existen en la actualidad una gran cantidad de aplicaciones y plataformas que pretenden la automatización de las tareas relacionadas con el entorno de los Servicios Web a través de la anotación semántica de las capacidades de los mismos. Se ha comprobado, además, que, en la mayoría de los casos, estos sistemas software hacen uso de la tecnología de agentes para sacar el máximo provecho a los componentes desarrollados en este campo y la madurez de los mismos. Sin embargo, tal y como se ha detallado en las secciones anteriores, las tecnologías disponibles poseen una serie de problemas e inconvenientes que complican su aplicabilidad en la resolución de problemas en entornos reales.

En lo que respecta a soluciones concretas, y en relación con las descritas en secciones anteriores, el principal problema de GODO radica en su restricción a la hora de interactuar con los Servicios Web Semánticos, haciendo imposible sacar provecho de

las numerosas posibilidades que ofrece la tecnología de agentes para la consecución de tareas tales como descubrimiento, selección y composición de Servicios Web. Por su parte, SWF está demasiado vinculado con WSMO, no contemplando, así, el amplio abanico de aproximaciones de Servicios Web Semánticos existentes en la actualidad y la inexistencia de un estándar global. El uso de agentes como "envoltorio" de Servicios Web, como proponen Buhler y Vidal (2005), conlleva también serias limitaciones como la necesidad de definir un flujo de trabajo estático que contradice el dinamismo natural de este tipo de sistemas. El Task Computing, aunque es más cercano a la computación pervasiva, también contempla la utilización de servicios a partir de sus descripciones semánticas. Sin embargo, esta aproximación no hace uso de la tecnología de agentes, limitando, de este modo, la autonomía del sistema, así como obviando tareas tan esenciales en estos entornos como la negociación.

En general, ninguna de las soluciones tecnológicas definidas hasta el momento para la integración de la tecnología de agentes y Servicios Web Semánticos es capaz de aprovechar las ventajas que proporcionan estas tecnologías por separado ni superar los problemas asociados a las mismas. En esta tesis, se plantea un uso desacoplado de estas tecnologías haciendo uso de ontologías para permitir una comunicación efectiva entre las mismas. De este modo, se consigue mantener a cada entidad en un nivel de abstracción distinto, potenciando sus virtudes y limitando al máximo los inconvenientes que puede suponer su uso. Con esta aproximación, los agentes realizan actividades de alto nivel utilizando como base la funcionalidad ofrecida por los Servicios Web, situados en una capa inferior. Así, la tecnología de Servicios Web se ciñe a su cometido, esto es, proporcionar servicios accesibles por medio de protocolos estándar de Internet y, por su parte, la tecnología de agentes despliega todo su potencial aprovechando las anotaciones de los servicios y constituyendo una plataforma altamente autónoma y que ofrece un gran dinamismo.

II.8. Resumen

En este capítulo, se han estudiado algunos de los trabajos de investigación en los que se hace uso de agentes y Servicios Web (Semánticos) de un modo integrado desarrollados

hasta el momento. En primer lugar, se muestra en extenso análisis de las soluciones más cercanas a la propuesta en esta tesis doctoral. A continuación, se estudian, en menor grado de detalle, otras metodologías que resultan de interés por su relación con los objetivos establecidos en esta tesis. Finalmente, se presentan los principales problemas y carencias que padecen las soluciones actuales, y que dan lugar al planteamiento de esta tesis.

GODO ha sido la primera herramienta en ser analizada. Se trata de un agente de usuario que actúa como intermediario entre los usuarios y los entornos de ejecución de Servicios Web Semánticos. El objetivo último del agente de usuario es procesar las consultas de los usuarios para generar las solicitudes correspondientes en los entornos de ejecución que estén accesibles. El principal problema de GODO radica en su restricción a la hora de interactuar con los Servicios Web Semánticos, haciendo imposible sacar provecho de las numerosas posibilidades que ofrece la tecnología de agentes para la consecución de tareas tales como descubrimiento, selección y composición de Servicios Web.

Semantic Web Fred es un proyecto en el que se combinan las tecnologías de agentes, ontologías y Servicios Web Semánticos para el desarrollo de tareas de forma cooperativa y automatizada. En este sistema, agentes software, llamados 'Freds', realizan tareas en representación de sus propietarios e interaccionan entre sí en caso necesario. Para resolver una tarea, los agentes hacen uso de los servicios, recursos computacionales que permiten la resolución automática de tareas. Se distinguen tres tipos de servicios: planes (programas Java), procesos (servicios complejos y anidados), y Servicios Web externos (a través de la interfaz WSDL). El problema primordial de SWF es su estrecha vinculación con WSMO, que le aleja de otras especificaciones de Servicios Web Semánticos existentes en un campo en el que no se ha definido un estándar global.

Otra solución propuesta en este sentido, es el uso de agentes como "envoltorio" de Servicios Web. En esta propuesta se destaca el comportamiento pasivo de los Servicios Web, y se plantea la utilización de Agentes Inteligentes que actúen en representación de ellos en un flujo de trabajo. De este modo, cuando un Servicio Web se intenta ejecutar, el agente que lo representa toma el control e intenta mejorar el proceso haciendo uso de

sus capacidades "inteligentes". Con esto, se consigue que flujos de trabajo estáticos definidos mediante Servicios Web, se conviertan en flujos dinámicos que presentan propiedades como la autonomía y la proactividad. Esta solución conlleva, sin embargo, serias limitaciones como la necesidad de definir un flujo de trabajo estático, lo que contradice el dinamismo natural de este tipo de sistemas.

Task Computing, por último, es un marco de trabajo orientado al usuario que permite llevar a cabo tareas complejas en entornos ricos desde el punto de vista de aplicaciones, dispositivos y servicios. La visión del Task Computing es bastante ambiciosa, integrando en un mismo entorno recursos tanto físicos como funcionales, aplicaciones, Servicios Web, contenido y dispositivos hardware. Sin embargo, esta visión se ve mermada por la carencia en la arquitectura de una entidad que posea las características de los agentes inteligentes (autonomía, pro-actividad, aprendizaje, etc.), lo cual limita las posibilidades que, de otro modo, podrían incorporarse al entorno.

Las limitaciones patentes en las soluciones que se han desarrollado hasta el momento para integrar las tecnologías de agentes y Servicios Web Semánticos, han conducido a la definición de esta tesis doctoral, donde Agentes Inteligentes, Servicios Web Semánticos y Ontologías constituyen la base tecnológica.

CAPÍTULO III. SISTEMA BASADO EN TECNOLOGÍAS DEL CONOCIMIENTO PARA ENTORNOS DE SERVICIOS WEB SEMÁNTICOS

III.1. Introducción

En el Capítulo I se ha incluido una descripción detallada acerca del estado del arte en todo lo concerniente a la tecnología de agentes y los Servicios Web Semánticos. Así, agente inteligente ha sido definido como aquella entidad software situada en un entorno determinado capaz de realizar tareas de forma autónoma en este entorno para satisfacer sus objetivos de diseño. Los agentes están caracterizados por diversas propiedades, entre las que se incluyen la reactividad (esto es, responder a cambios en el entorno), pro-actividad (o sea, tomar la iniciativa) y habilidad social (la capacidad de interactuar con otros agentes). Sin embargo, cuando se emplean estos agentes para realizar tareas sobre la Web, deben enfrentarse al problema asociado con la falta de estructura de la información publicada.

A diferencia de los agentes inteligentes, que fueron concebidos con el propósito de disponer de programas con la capacidad de decidir por si mismos y resolver tareas mecánicas y repetitivas, la tecnología de los Servicios Web surgió en el contexto de la computación distribuida como la mejor solución para la ejecución remota de funcionalidad. A diferencia de otras tecnologías en este campo, los Servicios Web se basan en protocolos estándar de Internet, por lo que su adopción ha sido realmente rápida. Sin embargo, conforme la Web crece en tamaño y diversidad, existe una creciente necesidad de automatizar aspectos de los Servicios Web como el descubrimiento, selección, composición, invocación y monitorización.

Por su parte, la Web Semántica promulga la anotación de las páginas Web con el contenido semántico procedente de ontologías. Una ontología es una especificación formal y explícita de una conceptualización compartida. Con esto, si los recursos Web se describen a partir de ontologías, entidades software independientes podrán hacer uso de la información disponible en la Web de igual modo en que lo podría hacer un usuario

humano. Esta evolución de la Web supone, tanto para la tecnología de agentes como para la de los Servicios Web, la desaparición de sus respectivas limitaciones. Así, los agentes serán capaces de procesar automáticamente la semántica de la información publicada en la Web sin tener que enfrentarse a la falta de estructura de la Web tradicional. Por su parte, la descripción semántica de las capacidades de los Servicios Web hace posible la automatización de las tareas de descubrimiento, selección, composición e invocación de los mismos.

No obstante, a pesar de los indudables beneficios potenciales de estas tecnologías, tras la adopción de la Web Semántica, tanto los agentes inteligentes como los Servicios Web Semánticos siguen teniendo una serie de problemas que requieren nuevas soluciones. Las plataformas multi-agente no están, en general, preparadas para comunicarse entre sí cuando es preciso traspasar las fronteras de las compañías, dado que el uso de protocolos no estándar limita las posibilidades de que los mensajes de interacción superen los cortafuegos existentes en las puertas de entrada de los sistemas software de las organizaciones. Las soluciones basadas en el establecimiento de puentes de comunicación entre plataformas localizadas en distintos entornos organizacionales carecen de este problema, si bien, deben de establecer estos puentes de forma estática en tiempo de diseño, perdiendo así el dinamismo que caracteriza la Web.

Por otro lado, la adición de contenido semántico formal y comprensible por sistemas computerizados a la descripción de las capacidades de los Servicios Web se concibió con el objetivo de permitir a entidades software entender esta información como lo haría un humano, y cubrir, así, diversos aspectos relacionados con esta tecnología de forma automática. Para esto, sin embargo, se supone la existencia de componentes software que incorporen capacidades para procesar el contenido semántico y razonar sobre el mismo. Se han desarrollado entornos de ejecución de Servicios Web Semánticos basados en componentes y que dan soporte, en mayor o menor grado, a la realización de estas tareas. No obstante, el uso de técnicas y herramientas procedentes de la tecnología de agentes en el diseño e implementación de este tipo de aplicaciones podría mejorar considerablemente las capacidades de las mismas. Así, entre otras cosas, sería posible aplicar técnicas de aprendizaje para mejorar los resultados ofrecidos por la herramienta,

así como hacer uso de mecanismos de negociación para permitir la aplicación de estos marcos de trabajo en entornos reales competitivos.

Por su parte, en el Capítulo II se transmite el común entendimiento que se tiene por parte de la comunidad investigadora de que la combinación de agentes software y Servicios Web Semánticos dará lugar a la siguiente evolución de la Web. Existen tres posibles escenarios para conseguir esta integración: (1) los Servicios Web proporcionan la funcionalidad básica mientras que los agentes hacen uso de estos servicios para proveer de funciones de valor añadido; (2) incorporar capacidades de agentes a los Servicios Web; y (3) crear puentes de comunicación entre ambos tipos de entidades, obteniendo un espacio heterogéneo. Además, en este capítulo se enumeran algunas de las metodologías que se han elaborado hasta la actualidad en las que tecnologías de agentes software y de Servicios Web Semánticos interactúan en un entorno integrado. Todas y cada una de las soluciones descritas fallan en su propósito de sacar el máximo provecho a esta combinación. Esto se debe, principalmente, a su incapacidad para utilizar todo el potencial de cada una de dichas tecnologías.

En esta tesis doctoral, se parte del hecho de que, conceptualmente, agentes inteligentes y Servicios Web (semánticos) fueron concebidos con propósitos totalmente dispares y, como tal, se entiende (como hipótesis) que deben permanecer en dos niveles de abstracción diferentes. La principal idea que fundamenta la tecnología de agentes no es que los agentes inteligentes sean capaces de proporcionar servicios, sino que aquellos son concebidos como entidades autónomas que incorporan inteligencia y capacidades cognitivas que les permitan mostrar un comportamiento pro-activo orientado a objetivos y establecer procesos de interacción, ya sea competitivos o cooperativos, con otras entidades para satisfacer sus objetivos de diseño. Por su parte, los Servicios Web supusieron una nueva evolución en el mundo de la computación distribuida y tienen el único propósito de proveer funcionalidad a partir de componentes software accesibles globalmente. Estas diferencias conceptuales justifican la necesidad de disponer de ambas tecnologías en un entorno integrado y dan pistas sobre los beneficios que se pueden obtener si se aplican para tareas con distinto nivel de abstracción para el desarrollo de sistemas complejos. En base a esta hipótesis de partida, se ha desarrollado un marco de trabajo que hace uso de las tecnologías de agentes y de Servicios Web

Semánticos para la elaboración de aplicaciones que puedan tratar con el dinamismo de la Web al tiempo que poseen características tan sutiles como la autonomía y el aprendizaje.

En este capítulo se describe, paso a paso, el proceso seguido para el diseño de este marco de trabajo, desde los objetivos de diseño hasta la arquitectura del sistema final.

III.2. Objetivos de Diseño

La meta principal que se persigue en el diseño del marco de trabajo objeto de esta tesis doctoral es la *obtención de un sistema que, a partir del empleo de agentes inteligentes y el uso intensivo de ontologías, mejore el rendimiento en el consumo de los Servicios Web e incremente la automatización de las tareas de gestión de los mismos, de forma que se explote el dinamismo intrínseco de este tipo de entornos*. En otras palabras, el objetivo último es el desarrollo de un entorno de ejecución de Servicios Web Semánticos basado en tecnología de agentes. Este objetivo general se puede descomponer en una serie de subobjetivos del siguiente modo:

- Elaboración de un Sistema Multi-Agente con los siguientes tipos de agentes:
  - Agentes de gestión de Servicios Web, y
  - Agentes de gestión de la plataforma.
- Disponer de un mecanismo de gestión integral del conocimiento basado en ontologías y que tenga en cuenta, al menos, las siguientes bases de conocimiento (ver Fig. 78, pag. 168):
  - Capacidades de los Servicios Web publicados en Internet,
  - Conocimiento del dominio de la aplicación,
  - Conocimiento de los agentes del sistema,
  - Conocimiento de la aplicación, y
  - Conocimiento de mecanismos de negociación.
- Disponer de mecanismos para resolver problemas de interoperabilidad a distintos niveles:
  - Interoperabilidad de datos y su integración,
  - Interoperabilidad de procesos,

  - Interoperabilidad funcional, e
  - Interoperabilidad de protocolos.
- Disponer de un mecanismo para la gestión eficaz de servicios que cumpla las siguientes características:
  - El sistema tiene en cuenta la carga de cada servicio en todo momento y tiene como norma balancear esta carga.
  - Se hace uso de mecanismos de negociación para dar con el servicio que ofrece las mejores condiciones.
  - Manejar el dinamismo del entorno: unos servicios pueden dejar de estar disponibles a la vez que otros nuevos pueden aparecer (ser publicados en Internet).

Se han tenido en cuenta, a su vez, una serie de objetivos secundarios con los que se pretende conseguir que el sistema satisfaga los estándares de calidad industriales, de forma que sea posible su aplicación en entornos reales. Los objetivos de diseño secundarios son los siguientes:

- Independencia del dominio y de la aplicación (ver Fig. 48): se persigue el diseño de un marco de trabajo que permita el desarrollo de diversas aplicaciones (p.ej. comercio electrónico, integración de sistemas de información abiertos, negocio electrónico, etc.) en distintos dominios de aplicación (p.ej. informática, oncología, administración publica, etc.).
- Usabilidad: se distinguen tres perfiles de usuarios, desarrolladores software, proveedores de servicios y consumidores de servicios, divididos, a su vez, en consumidores individuales o colectivos (organizaciones). El diseño de la interfaz debe ser acorde al conocimiento que se le supone a cada uno de estos perfiles.
- Adaptabilidad y parametrización: el marco de trabajo debe de ser capaz de adaptarse a los cambios en el entorno de forma dinámica. De igual modo, se permitirá la configuración de diversos aspectos del sistema a partir de ficheros de configuración. En particular, el sistema deberá ofrecer la posibilidad, tanto a usuarios consumidores de servicios como a los proveedores de los mismos, de incluir su perfil y sus preferencias, que deberán de tenerse en cuenta a la hora de comenzar un proceso de negociación.

- Eficiencia y flexibilidad: se incorporarán algoritmos de eficiencia máxima. Será posible modificar los algoritmos de forma flexible, de modo que sea posible disponer en todo momento de los últimos avances en este campo.

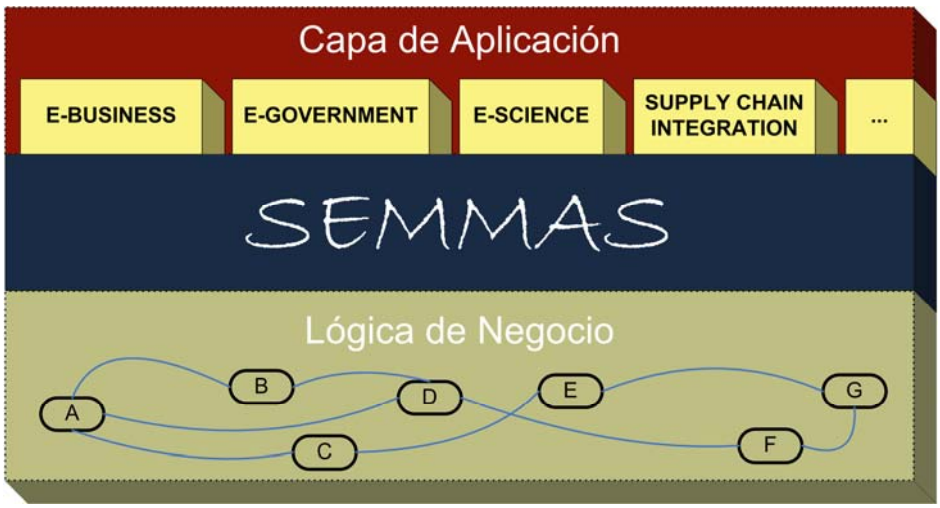


Fig. 48. Entorno de trabajo independiente de la aplicación y del dominio

III.3. Diseño y especificación

Tal y como se adelantó en la sección I.3.4, para el análisis y diseño del entorno de trabajo multi-agente se ha hecho uso de una metodología específica para el desarrollo de este tipo de sistemas como es INGENIAS[18]. La decisión de utilizar esta metodología entre las numerosas existentes para el desarrollo de SMAs se basó en tres aspectos fundamentales: (i) la metodología viene acompañada por un kit de desarrollo, el *INGENIAS Development Kit* (IDK), que permite la elaboración de los modelos de forma visual, (ii) tiene asociaciones para su integración con el Proceso Unificado, un marco de desarrollo software iterativo e incremental, y (iii) está bien alineada con la plataforma multi-agente *Java Agent DEvelopment Framework* (Jade), que es la utilizada en esta tesis, sobre la que se puede generar código directamente a partir de los modelos de la metodología.

[18] http://grasia.fdi.ucm.es/ingenias/

En la Tabla 3 (pag. 34), se pueden distinguir las fases y las tareas que las constituyen, y que se deben llevar a cabo para las etapas de análisis y diseño de las aplicaciones conforme a la metodología INGENIAS. Este flujo de trabajo, que parte de un conjunto de casos de uso generales, conlleva la elaboración de los siguientes cinco modelos que representan, con bastante exhaustividad, todos los requisitos y necesidades del sistema:

- Modelo de Organización: aclara la estructura del SMA, incluyendo roles, relaciones de poder y flujos de trabajo.
- Modelo de Agente: representa las tareas que los agentes deben realizar para alcanzar sus objetivos.
- Modelo de Objetivos/Tareas: identifica tanto los objetivos y tareas generales, como su descomposición en objetivos y tareas más concretas que se pueden asignar a los agentes.
- Modelo de Interacción: muestra la interacción entre agentes y roles.
- Modelo de Entorno: especifica las entidades y relaciones con el entorno del SMA.

En esta sección se muestra, de forma resumida, el proceso seguido a través de los modelos obtenidos. Partiremos de los casos de uso, identificados a partir de los objetivos de diseño detallados en la sección anterior, para continuar con los modelos de organización, agente, objetivos/tareas, interacción y, finalmente, entorno.

III.3.1. Casos de Uso

El caso de uso más general mostrado en la Fig. 49 representa la funcionalidad al más alto nivel de la plataforma: hacer posible que unos usuarios hagan realidad sus necesidades haciendo uso de los recursos que otros usuarios hacen disponibles. En la figura se pueden distinguir los dos actores principales que interactúan con la plataforma: proveedores de servicios y consumidores de servicios.

Fig. 49. Caso de uso (nivel 0)

La descomposición del caso de uso general mostrado en casos de uso más específicos ofrece una visión más completa acerca de las distintas tareas que ha de soportar el sistema. Un primer caso de uso específico muestra, de forma detallada, el modo en que los proveedores de servicios pueden publicar servicios en la plataforma (ver Fig. 50). Es posible comprobar cómo el proveedor de servicios puede especificar tanto una serie de condiciones generales, que se aplicarán en la ejecución de todos los servicios proporcionados por el mismo, como las condiciones particulares para cada uno de los servicios que publique, y que se tendrán en cuenta únicamente cuando se deba de ejecutar dicho servicio. Como complemento a este caso de uso, existe otro que permite, tanto a proveedores de servicios como a consumidores, modificar (añadir o eliminar) la lista de repositorios de servicios (esto es, repositorios donde se encuentran disponibles las descripciones semánticas de los Servicios Web disponibles en Internet), de forma que todos los servicios publicados en estos repositorios también estén disponibles para su uso (ver Fig. 51).

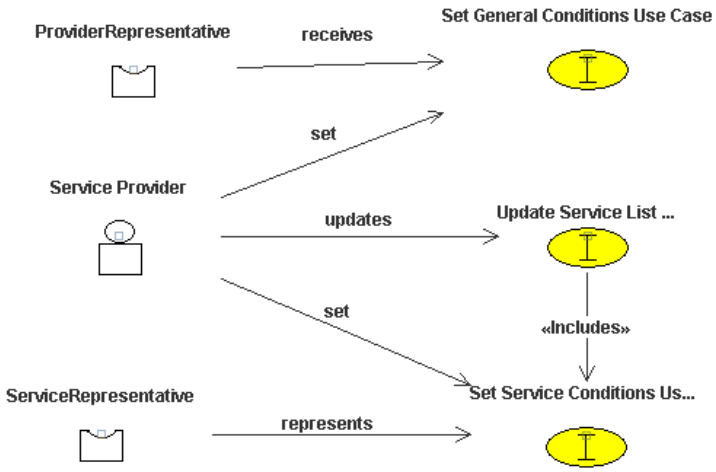


Fig. 50. Caso de uso (nivel 1) – publicación de servicios

Fig. 51. Caso de uso (nivel 1) – modificación de repositorios

Una de las tareas fundamentales de la plataforma es la gestión y manejo de los Servicios Web Semánticos. Previamente, se ha identificado un conjunto de tareas

básicas en este sentido: descubrimiento, selección, composición e invocación. En la Fig. 52, se muestra el caso de uso para el descubrimiento de servicios que está, a su vez, compuesto por la búsqueda de servicios que satisfagan las necesidades especificadas, selección y ordenación de éstos y, para cuando sea necesario, composición de servicios. Para cada una de estas sub-tareas, se identifica una serie de roles que se encargarán de su ejecución. De forma separada, se ha representado la invocación del servicio (ver Fig. 53), en el que participan dos roles, el representante del consumidor y el invocador. Adicionalmente a las tareas básicas para la gestión de los Servicios Web comentadas, se ha incorporado una tarea adicional, a saber, la monitorización, que se encarga de controlar la ejecución de los servicios para asegurar que se satisfacen las condiciones impuestas en la selección de dicho servicio (ver Fig. 54).

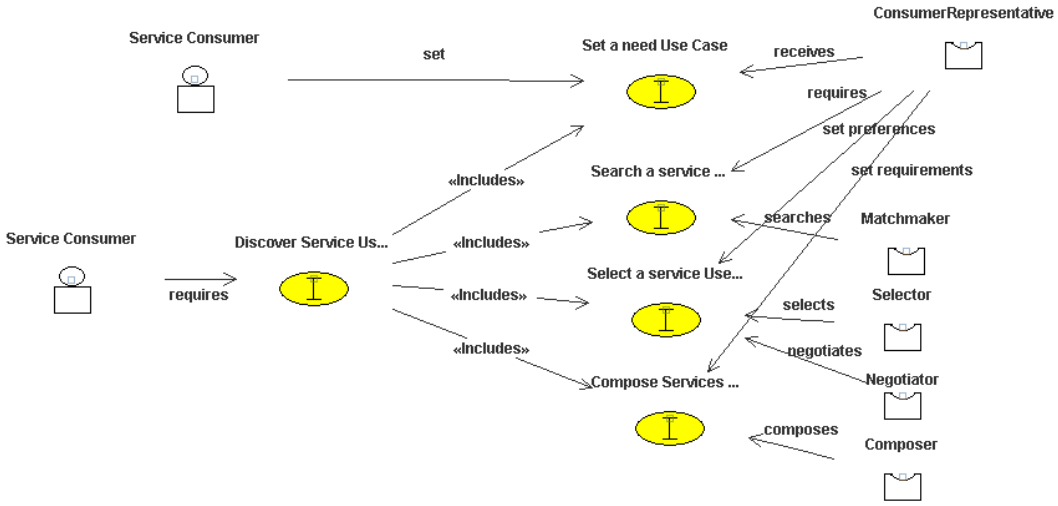


Fig. 52. Caso de uso (nivel 1) – descubrimiento, selección y composición de servicios

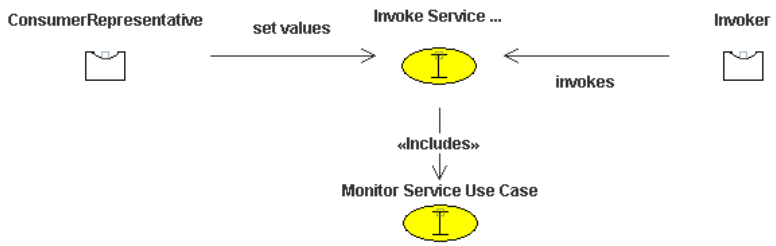


Fig. 53. Caso de uso (nivel 1) – invocación de servicios

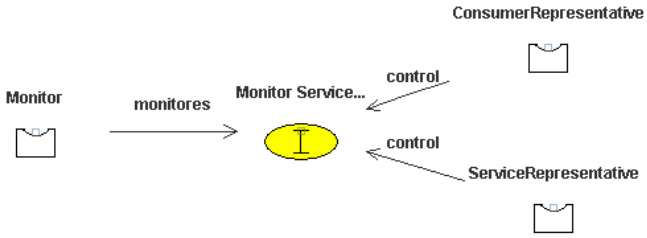


Fig. 54. Caso de uso (nivel 1) – monitorización

Otro aspecto clave que se debe tener en cuenta cuando se ha de interaccionar con múltiples recursos de Internet es la interoperabilidad. En estos entornos, pueden surgir distintos problemas de integración que requieran procesos de mediación para su correcto funcionamiento. En esta tesis doctoral, se ha intentado abarcar todo el espectro de problemas (ver Fig. 55): interoperabilidad de datos, interoperabilidad de procesos, interoperabilidad de protocolos e interoperabilidad funcional. En la sección III.3.4, se comenta en profundidad cada uno de estos problemas y se detallan los mecanismos de mediación utilizados para su resolución.

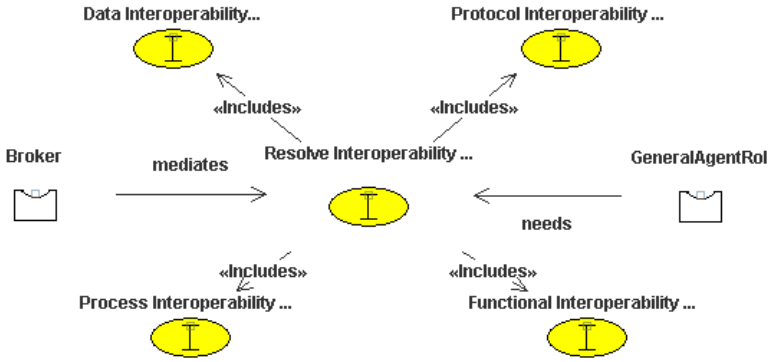


Fig. 55. Caso de uso (nivel 1) – mediación

Aparte de aquellos casos de uso dedicados a la gestión y explotación de los Servicios Web, se han añadido diversos casos de uso concernientes a la gestión de la plataforma en general con el objetivo de que éstos sirvan para que se tengan en cuenta elementos como la calidad, la eficiencia, el rendimiento, la gestión de recursos, etc. necesarios para el desarrollo de una aplicación con vistas a su explotación en el contexto industrial. Por un lado, se ha especificado un caso de uso que representa la monitorización de todos los recursos del sistema para permitir que el sistema sea consciente de posibles fallos internos y sea capaz de recuperarse (ver Fig. 56). Por otro lado, se ha incorporado un

caso de uso, aún más general, con la intención de controlar el consumo de recursos del sistema y que limitará el uso de la aplicación para que ésta mantenga, en todo momento, los estándares de eficiencia mínima esperados (ver Fig. 57).

Fig. 56. Caso de uso (nivel 1) – monitor del sistema

Fig. 57. Caso de uso (nivel 1) – gestión integral de la plataforma

III.3.2. Modelo de Organización

El modelo de organización describe cómo se agrupan los distintos componentes del sistema (agentes, roles, recursos y aplicaciones), la funcionalidad del sistema y las restricciones que hay que imponer sobre la interacción entre los agentes. En el caso particular de la plataforma desarrollada en esta tesis, se han distinguido dos puntos de vista, el del consumidor de servicios y el del proveedor. En la siguiente figura (ver Fig. 58), se representa el diagrama de organización para el objetivo global de consecución de las actividades requeridas. En dicha figura, se puede distinguir la división en "departamentos" entre proveedores y solicitantes, cada uno de los cuales incorpora ciertos roles y, en el caso del proveedor, un recurso adicional que se refiere al proceso de negocio interno, que será, a la postre, lo que permitirá la ejecución de los servicios.

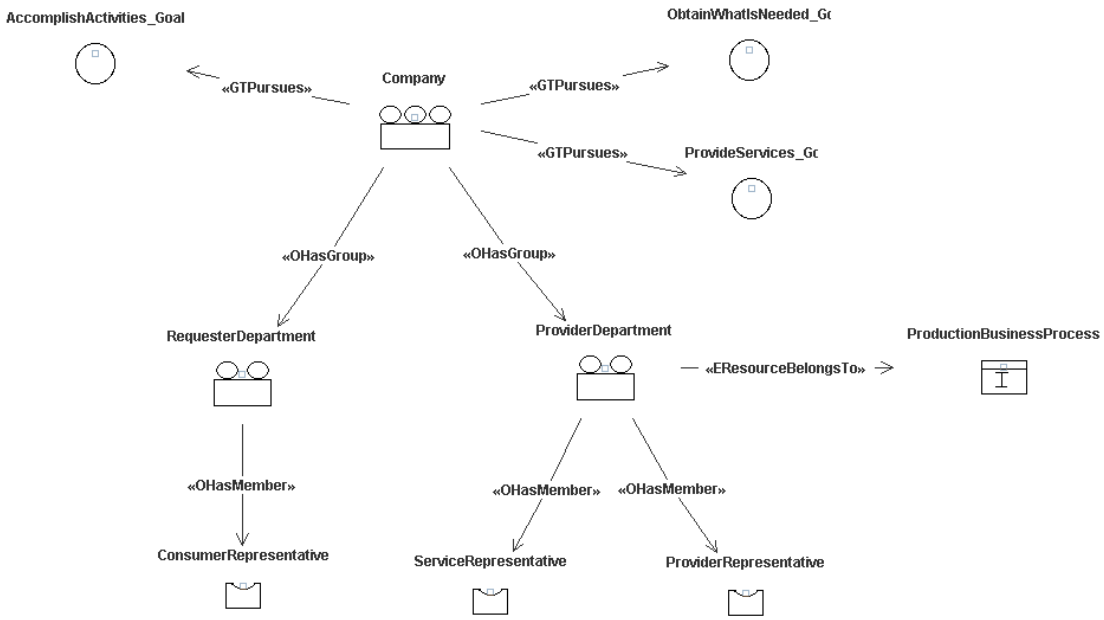


Fig. 58. Diagrama de Organización – entorno

En esta figura, '*Company*' representa a cualquier organización interesada en proporcionar servicios en todo el mundo. El departamento '*RequesterDepartment*' es el encargado de solicitar a otras entidades la ejecución de servicios que no están disponibles internamente (esto es, actúa como consumidor de servicios). Por su parte, el departamento '*ProviderDepartment*' es responsable de ejecutar las tareas requeridas, esto es, proporcionar los servicios que la organización provee.

Otro de los objetivos de primer nivel de la plataforma es la ya mencionada interoperabilidad. El siguiente diagrama (ver Fig. 59) de organización describe la estructura organizativa del sistema que se encarga de facilitar la comunicación entre las entidades que precisan de un servicio (esto es, consumidores de servicios) y las entidades que proporcionan servicios (o sea, proveedores de servicios).

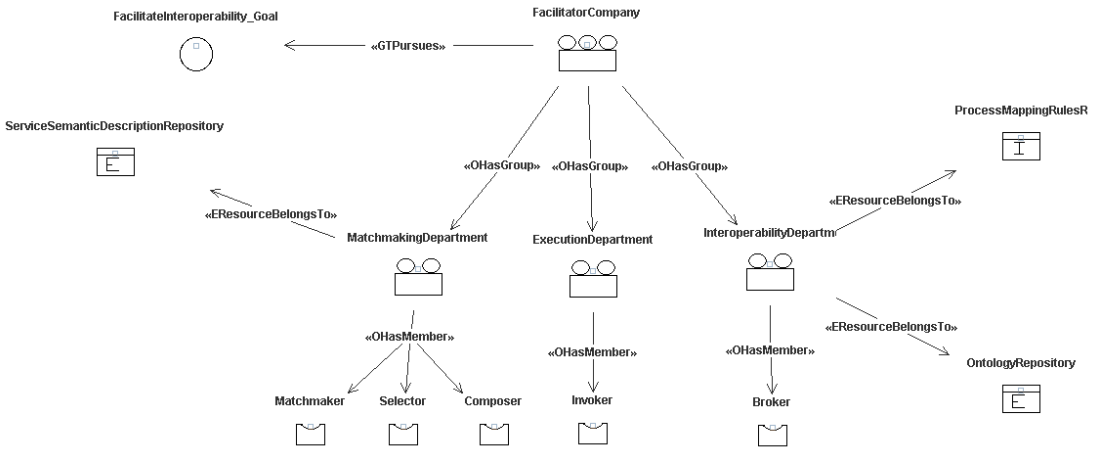


Fig. 59. Diagrama de organización – facilitador

En este diagrama se presentan los elementos necesarios para la búsqueda, selección, composición, mediación e invocación de los servicios, así como los recursos internos y externos que son necesarios para que estos procesos puedan llevarse a cabo de forma satisfactoria.

III.3.3. Modelo de Agentes

Mediante el modelo de agentes, es posible describir agentes particulares, sus tareas, objetivos, los roles que desempeñan y los estados cognitivos en que se encontrarán a lo largo de su vida. A continuación, se muestra el modelo de agentes para los agentes que interaccionan con las entidades externas (esto es, proveedores y consumidores de servicios), el '*Customer Agent*' y el '*Provider Agent*' (ver Fig. 60). El primero persigue dos objetivos principales, a saber, permitir que el cliente pueda establecer sus necesidades y satisfacer dichas necesidades. Complementariamente, el segundo agente tiene como objetivo proporcionar los servicios que le son indicados por la organización a la que representa.

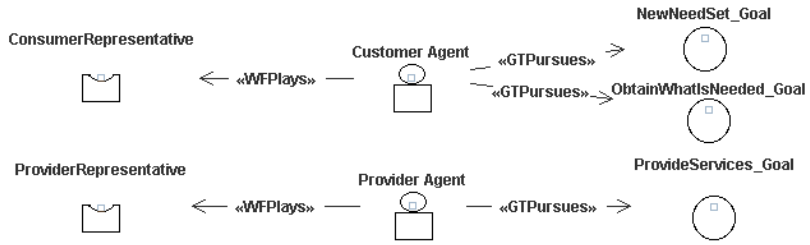


Fig. 60. Modelo de agentes – agentes consumidor y proveedor

Los agentes encargados de la búsqueda y selección de los servicios son el '*Discovery Agent*' y el '*Selection Agent*' (ver Fig. 61). En el caso particular del descubridor, se han tenido en cuenta dos roles, '*Matchmaker*' y 'Composer', que permiten a este agente encontrar el servicio o lista de servicios que pueden realizar una tarea concreta.

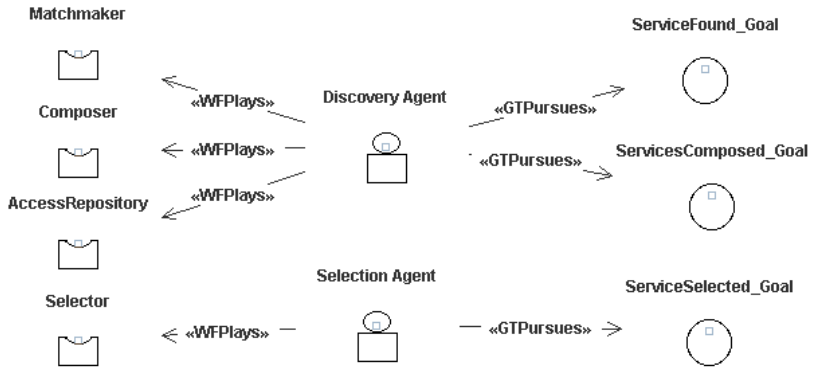


Fig. 61. Modelo de agentes – agentes descubridor y selector

La gestión y uso de los servicios concluye con la ejecución de los mismos. Este es uno de los propósitos perseguidos por el '*Service Agent*' que, a su vez, se debe de encargar de que se tengan en cuenta las condiciones establecidas por el proveedor a la hora de ejecutar el servicio correspondiente. Además, como ya se destacó anteriormente, un sistema de estas características requiere de un elemento mediador que se encargue de resolver los problemas de interoperabilidad que puedan surgir entre el resto de componentes. Esta es la misión del '*Broker Agent*' (ver Fig. 62).

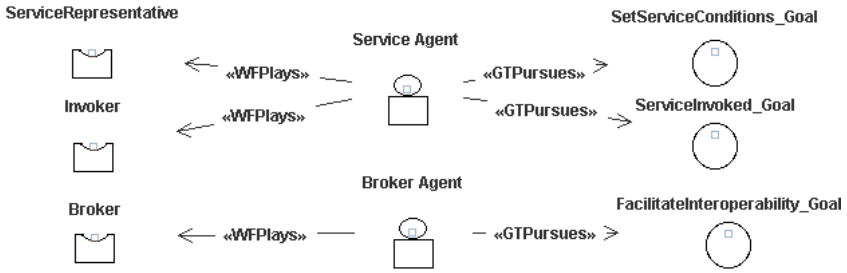


Fig. 62. Modelo de agentes – agentes de servicio y mediador

Finalmente, las tareas de gestión de la plataforma son realizadas por el '*Framework Agent*' (ver Fig. 63). En general, este agente se encarga de monitorizar el estado del resto de agentes y de controlar los recursos del sistema.

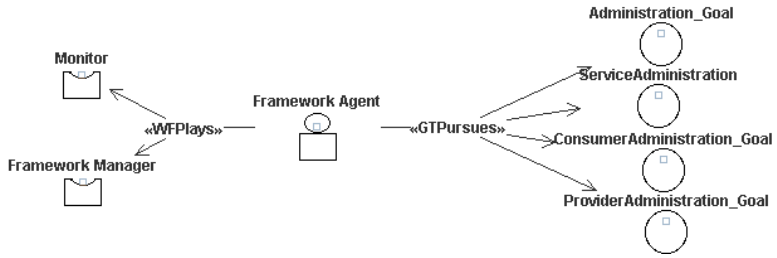


Fig. 63. Modelo de agentes – agente de gestión y monitorización de la plataforma

III.3.4. Modelo de Tareas y Objetivos

El modelo de tareas y objetivos define las relaciones entre objetivos y tareas, estructuras de objetivos y estructuras de tareas. También permite indicar las entradas y salidas de cada tarea y cuales son los efectos de las mismas, tanto en el entorno como en el estado cognitivo de los agentes. Como se comentó previamente, el objetivo principal que pretende que satisfaga la plataforma es el de realizar actividades para cumplir los deseos de los usuarios. Este objetivo general se ha dividido en cuatro subobjetivos de gran importancia (ver Fig. 64): administrar el sistema, obtener lo que es necesario, facilitar la interoperabilidad y proporcionar servicios. El agente encargado de, en un sentido genérico, la administración del sistema es el '*Framework Agent*'. Por su parte, el objetivo de obtener lo que es necesario, descompuesto a su vez en un objetivo más primitivo como es el de establecer una necesidad, es asumido por el propio usuario

consumidor de servicios. De los asuntos de interoperabilidad, se encarga el '*Broker Agent*' y, finalmente, el proveedor de servicios es el que, a este nivel de abstracción, persigue el objetivo de proveer servicios.

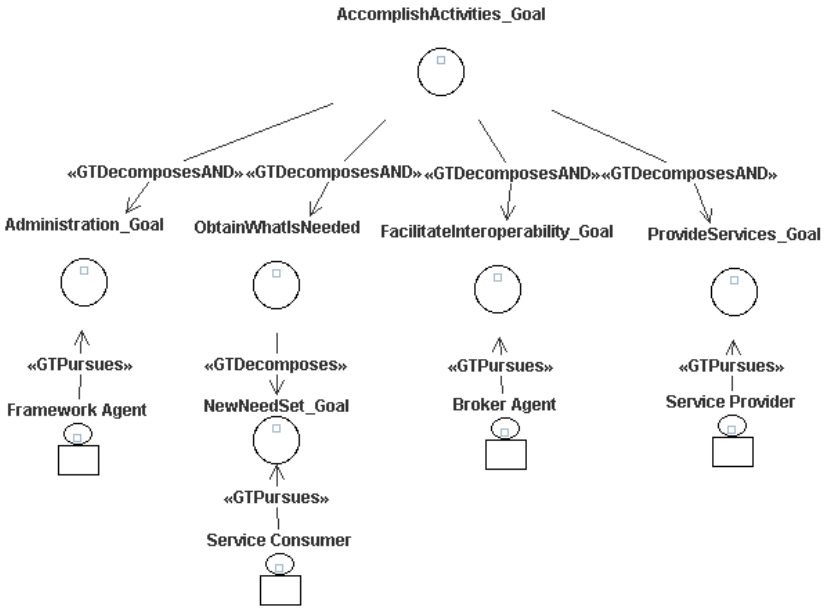


Fig. 64. Modelo de Tareas y Objetivos (nivel 0)

El objetivo de administración del sistema se puede dividir, a su vez, en tres subobjetivos (ver Fig. 65): administración de proveedores, administración de consumidores y administración de servicios. La administración de proveedores y consumidores se basa, fundamentalmente, en la suscripción y re-suscripción de éstos en el sistema. De forma similar, la administración de servicios está compuesta por la publicación de nuevos servicios (para lo que será necesario establecer las condiciones, insertar el servicio y hacerlo disponible) y la eliminación de los mismos de la plataforma, todo esto llevado a cabo por parte del proveedor de servicios.

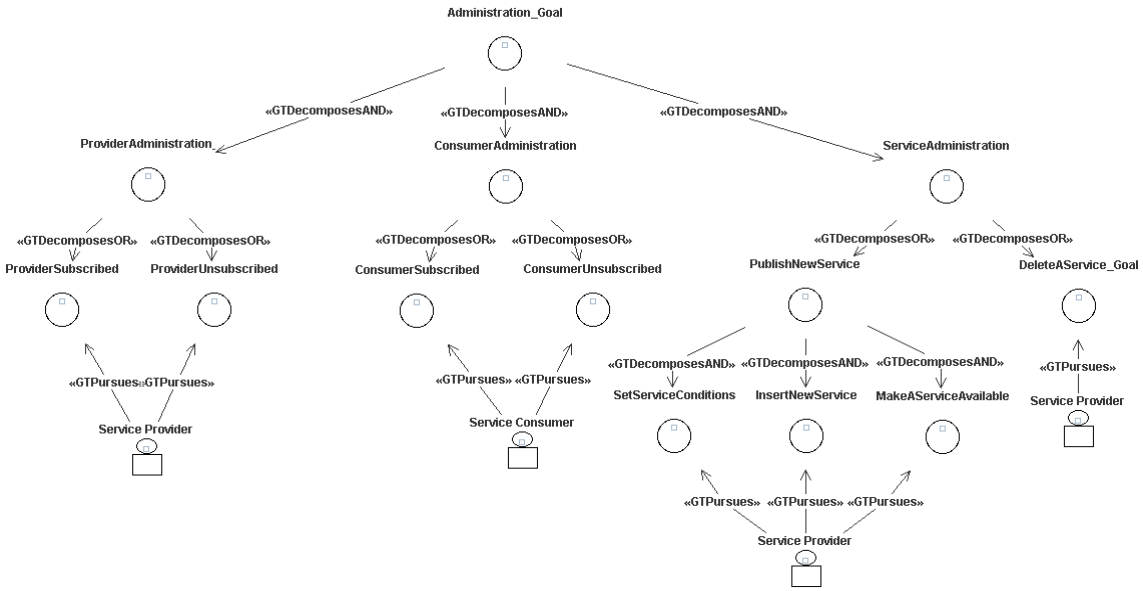


Fig. 65. Modelo de Tareas y Objetivos (nivel 1) – tareas de gestión

La provisión de servicios se realiza atendiendo a dos subobjetivos (ver Fig. 66): invocación y monitorización de servicios. Para la invocación, es necesario encontrar el servicio (tarea de la que se encarga el agente descubridor), seleccionar el servicio más apropiado (tarea del agente seleccionador), componer servicios (misión del agente descubridor), y ejecutar el servicio (perseguida por el agente de servicio). Por su parte, la monitorización de que se satisfacen las condiciones impuestas para la ejecución del servicio es llevada a cabo por el '*Framework Agent*'.

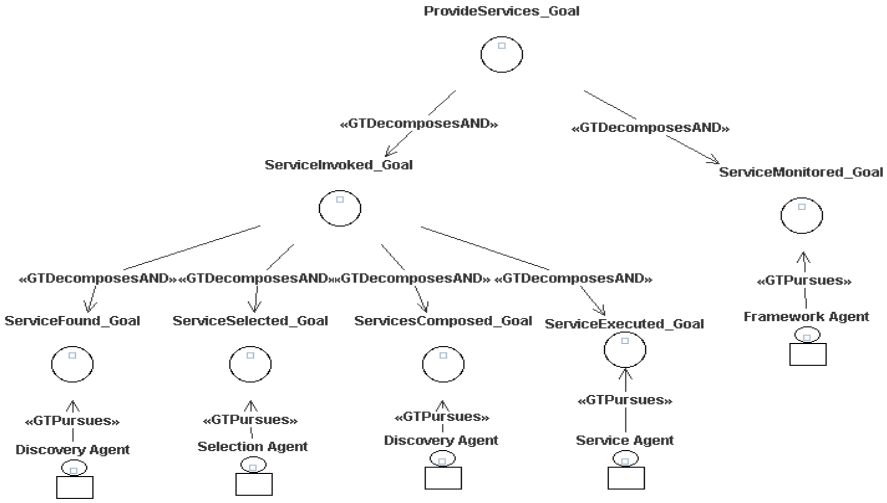


Fig. 66. Modelo de Tareas y Objetivos (nivel 1) – provisión de servicios

III.3.5. Modelo de Interacción

Mediante los modelos de interacción, se permite detallar cómo se coordinan y comunican los agentes. Cada declaración de interacción incluye los actores, el objetivo perseguido y el protocolo que sigue la interacción. A continuación, se muestran los diagramas que relacionan cada caso de uso con su respectivo modelo de interacción: casos de uso de gestión de servicios (ver Fig. 67), casos de uso de gestión de proveedores (ver Fig. 68) y casos de uso de gestión de consumidores (ver Fig. 69).

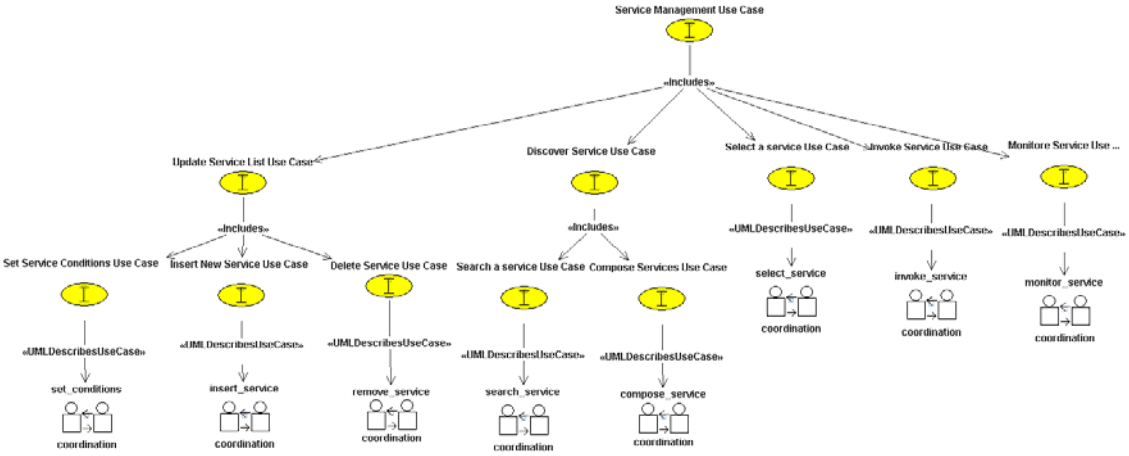


Fig. 67. Gestión de servicios – Caso de Uso & Interacción

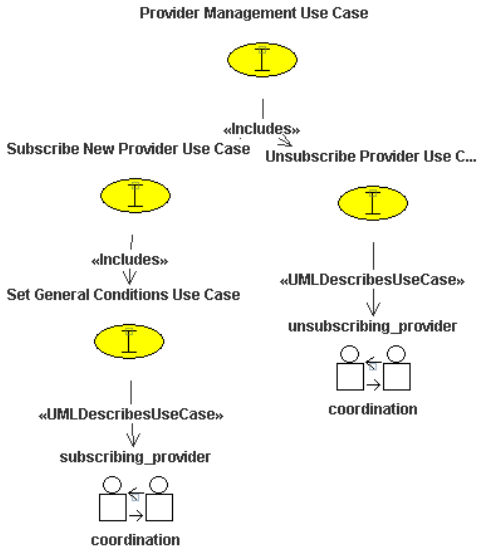


Fig. 68. Gestión de proveedores – Caso de Uso & Interacción

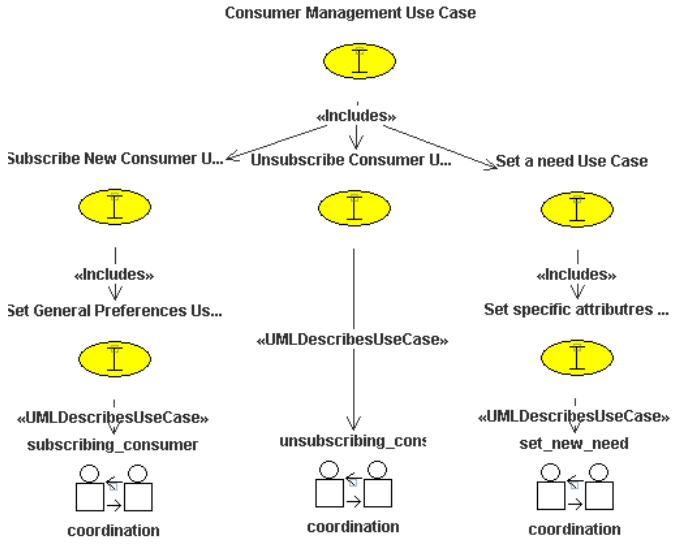


Fig. 69. Gestión de consumidores – Caso de Uso & Interacción

A modo de ejemplo, en la Fig. 70 se muestra el modelo de interacción correspondiente a la introducción de necesidades al sistema por parte de los usuarios consumidores de servicios. El objetivo que se persigue con esta interacción es el de disponer de un mecanismo para añadir una nueva necesidad. El iniciador de la interacción es el propio usuario que, dado que precisa la obtención de ciertos resultados, solicita al sistema que ejecute los servicios necesarios para conseguirlos. Para que la interacción tenga éxito, es necesaria la colaboración del rol '*ConsumerRepresentative*', que, como se indicó anteriormente, es asumido por el agente de consumidor.

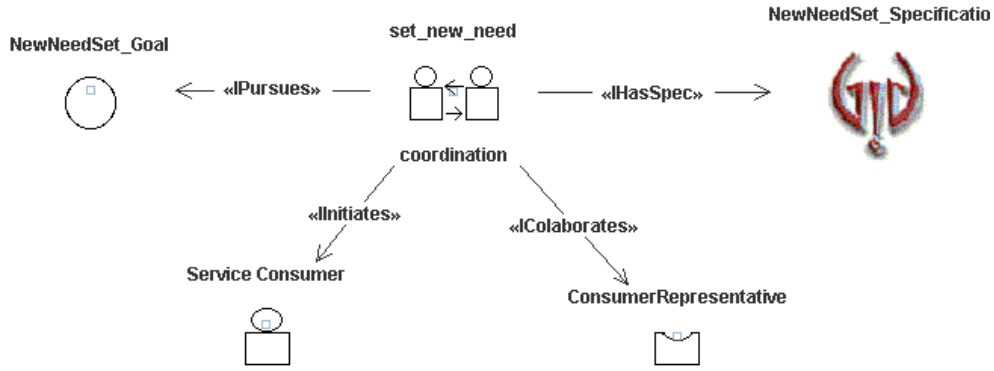


Fig. 70. Modelo de Interacción – Establecer necesidades

Para completar el modelo de interacción anterior, se ha elaborado un diagrama de colaboración, que establece los pasos a seguir para llevar a cabo la interacción y su precedencia (ver Fig. 71).

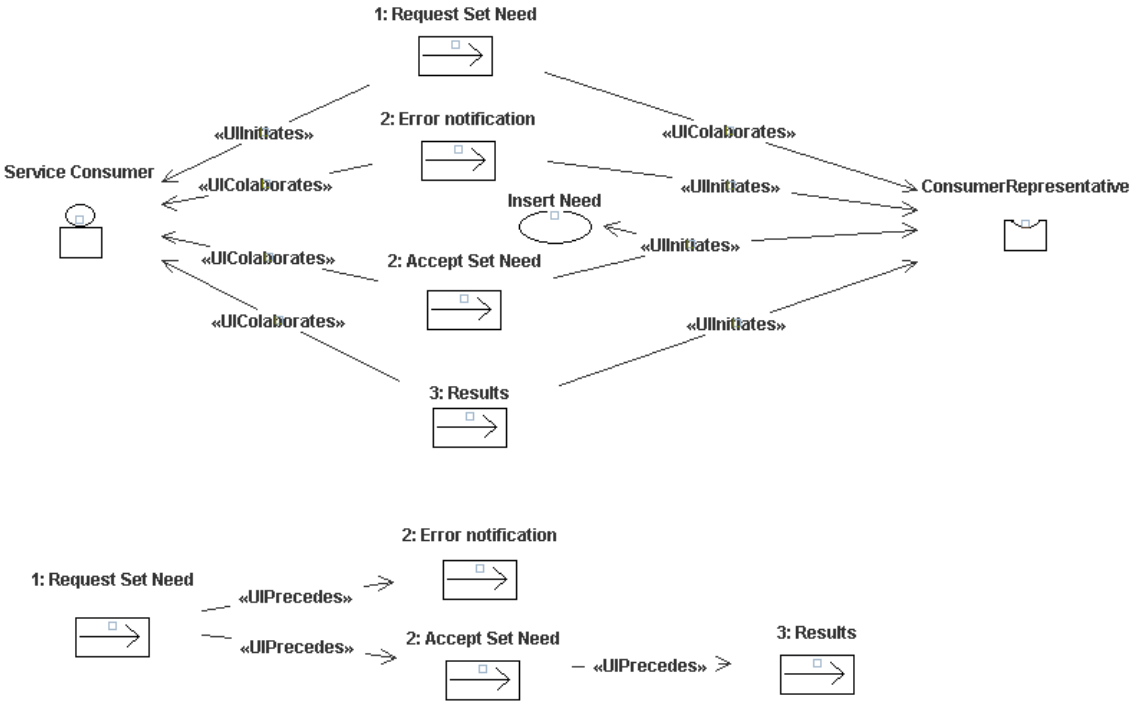


Fig. 71. Modelo de Colaboración – Establecer necesidades

III.3.6. Modelo de Entorno

El modelo de entorno define qué existe alrededor del nuevo sistema y cómo lo percibe cada agente. También identifica los recursos del sistema y quién es el responsable de la gestión de los mismos. Para el diseño de este marco de trabajo, se han identificado diversos recursos, tanto externos como internos al sistema. Entre los recursos que se encuentran de forma externa a la plataforma (ver Fig. 72), se pueden destacar los repositorios de descripciones semánticas de Servicios Web, que están dispersos por Internet (se trata, en general, de gestores de datos en RDF), el servidor de aplicaciones, que contiene la aplicación Web por medio de la que, tanto proveedores como consumidores, interactúan con el sistema, y los servicios en sí que son proporcionados por las entidades proveedoras.

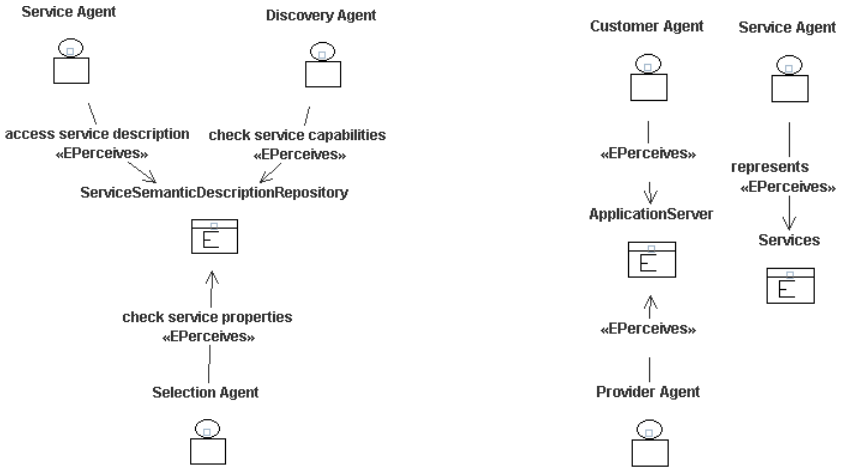


Fig. 72. Modelo de Entorno – recursos externos a la plataforma

Entre los recursos internos de los que hace uso la plataforma, se encuentran múltiples bases de conocimiento. En primer lugar, se puede destacar la base de conocimiento de mecanismos de negociación (ver Fig. 73). Esta contiene diversos protocolos y estrategias de negociación que los agentes de la plataforma debe seleccionar a la hora de proceder a un proceso de negociación. En particular, son los agentes '*Service Agent*' y '*Selection Agent*' los que se encargan de realizar el proceso de negociación cuando sea preciso elegir de entre una lista de servicios aquel que mejor cumpla las condiciones establecidas por el usuario.

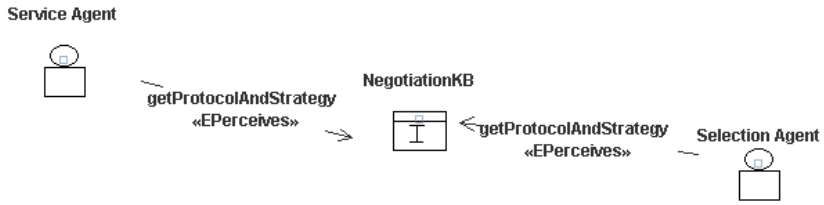


Fig. 73. Modelo de Entorno – recursos internos a la plataforma

En la Fig. 74, se muestran otros tres repositorios de datos utilizados en el marco de trabajo. En una base de conocimiento general, se almacena el conocimiento preciso para diversas tareas como la composición de servicios, el aprendizaje para la selección de servicios y la mejora en la ejecución de los mismos, y el conocimiento local de cada agente. Por otra parte, existe una base de datos que contiene los perfiles de los usuarios que están registrados en la plataforma. Este perfil se utiliza, si es el caso, en los

procesos de negociación que tengan lugar para la selección de servicios. Por último, las reglas para la resolución de los problemas de interoperabilidad están almacenadas en una base de conocimiento dedicada.

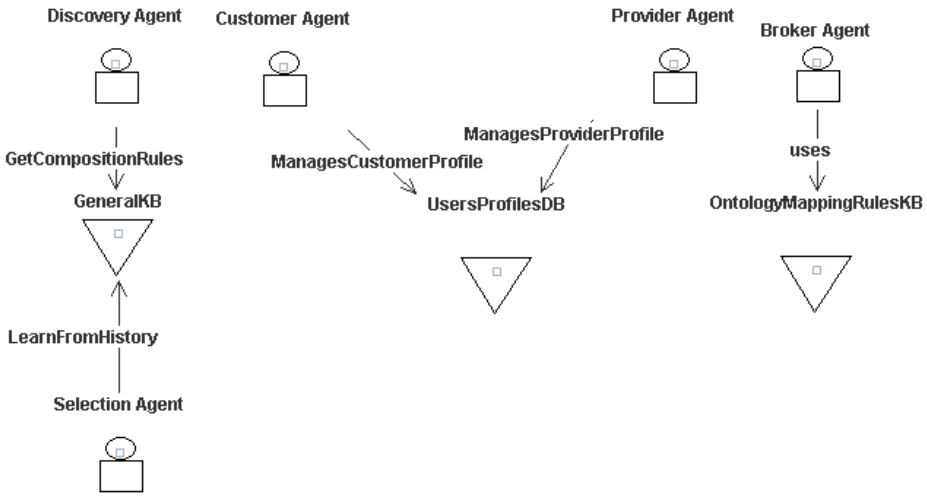


Fig. 74. Modelo de Entorno – bases de conocimiento

III.4. Arquitectura

En base a los modelos y diagramas producidos en las fases de análisis y diseño de INGENIAS, se ha desarrollado un sistema basado en conocimiento para entornos de Servicios Web Semánticos. La arquitectura general del sistema se presenta de forma gráfica en la Fig. 75.

El sistema está constituido por los siete agentes identificados durante la fase de diseño, cuatro bases de conocimiento que contienen las ontologías necesarias para el correcto funcionamiento de la plataforma y tres interfaces para interaccionar con entidades externas al sistema: consumidores de servicios, proveedores de servicios y desarrolladores software.

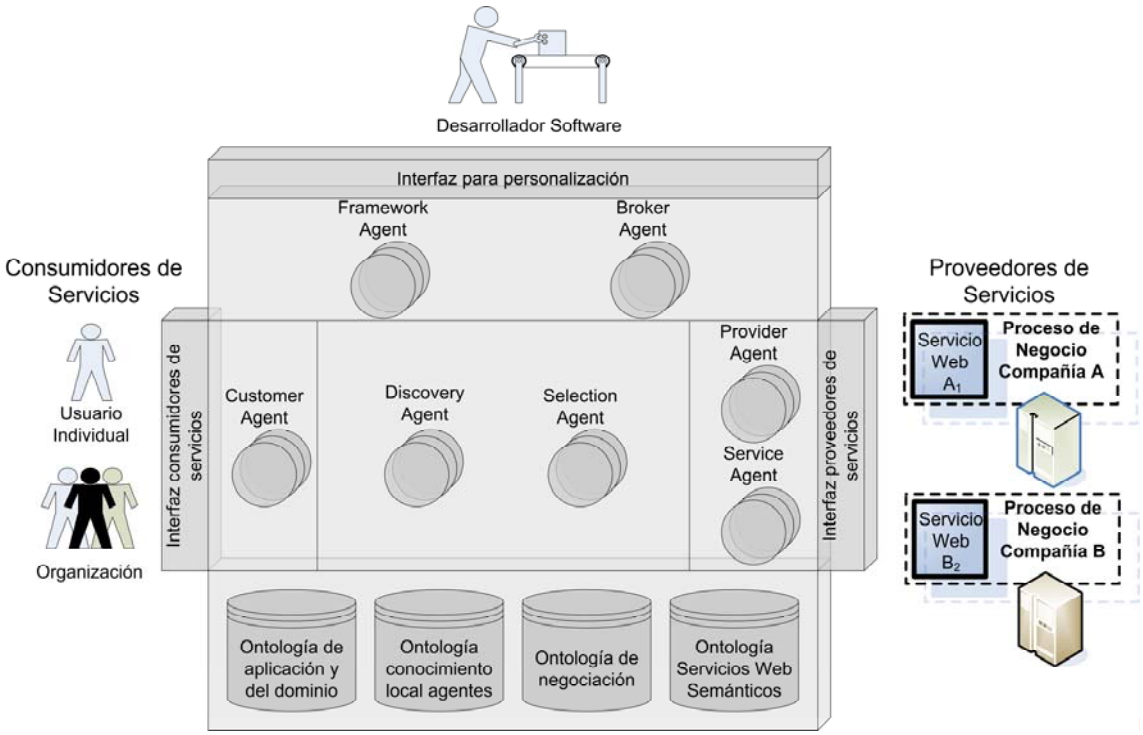


Fig. 75. Arquitectura general del sistema

En esta sección, se ofrece una descripción detallada de cada uno de los componentes que conforman la arquitectura. En primer lugar, se examinan los elementos del SMA, agentes, roles y bases de conocimiento. A continuación, se explican los mecanismos utilizados para la gestión de los Servicios Web y su integración en la plataforma. En tercer lugar, se describe cómo se hace frente a los problemas de interoperabilidad en la plataforma y, por último, se definen las características de cada una de las interfaces externas.

III.4.1. Sistema Multi-Agente

El SMA presentado en la Fig. 75 está constituido por cuatro bases de conocimiento y un conjunto de agentes, en el que cada uno de los cuales juega diferentes roles. En los siguientes apartados, se explica de forma detallada la funcionalidad de cada uno de estos agentes en relación a sus objetivos de diseño y las interrelaciones que han de tener lugar entre ellos. Asimismo, se describen todos los roles que se han diseñado y que implementan las funciones que han de cumplir cada uno de los agentes. Finalmente, se

comentan brevemente las bases de conocimiento incluidas en la plataforma, su propósito y su contenido.

III.4.1.1. Agentes Inteligentes involucrados

De acuerdo con las fases de análisis y diseño elaboradas conforme a la metodología INGENIAS, se determinó la necesidad de elaborar hasta un total de siete agentes distintos, cada uno de ellos con una misión y unos objetivos claramente definidos. Se pueden distinguir tres grupos principales de agentes: (i) agentes que actúan en nombre de los propietarios de servicios (p.ej. '*Provider Agent*' y '*Service Agent*'), (ii) agentes que actúan por cuenta de los consumidores de servicios (p.ej. '*Customer Agent*', '*Discovery Agent*' y '*Selection Agent*') y (iii) agentes que se encargan de las tareas de gestión (p.ej. '*Framework Agent*' y '*Broker Agent*'). Los agentes que representan a los proveedores de servicios gestionan el acceso a los servicios y se aseguran de que los contratos que se establezcan con los clientes sean satisfechos por ambas partes. Por su parte, los agentes que representan a los clientes o consumidores de servicios tienen que localizar los servicios, establecer los contratos pertinentes para su ejecución, y recibir y presentar los resultados. En último lugar, los agentes de gestión controlan tanto los recursos con que cuenta la plataforma para evitar problemas de sobrecarga, como las interacciones entre los restantes agentes del sistema para impedir incompatibilidades y resolver problemas de interoperabilidad en sus comunicaciones.

A continuación, se ofrece una descripción de cada uno de los agentes de la plataforma (En la Tabla 5, se muestra resumidamente información acerca de la relación entre los distintos tipos de agente y los objetivos que éstos persiguen. Por su parte, la Tabla 6 muestra en detalle las entradas, salidas esperadas e interacciones de cada uno de estos agentes)

1 ***Customer Agent:*** este agente actúa como representante de los usuarios consumidores de servicios. En primer lugar, debe situarse como el receptor de las preferencias que establezcan los usuarios tanto individuales (o sea, clientes particulares) o colectivos (esto es, organizaciones, compañías). A continuación, el agente procesa la entrada del usuario que expresa lo que desea obtener (o sea, el objetivo) y lo representa en el formato interno necesario. Esta entrada puede darse

en diversos formatos, como lenguaje natural o mediante una representación ontológica. Por último, este agente es el encargado de realizar las interacciones necesarias con los restantes agentes para llevar a cabo, de forma exitosa y al menor costo posible, el objetivo del usuario (el proceso que se produce internamente es totalmente transparente al usuario final, aunque es posible que se requiera de la participación del usuario en alguno de los pasos intermedios). Los resultados son presentados de vuelta al usuario en el formato más apropiado posible.

2 ***Discovery Agent:*** este agente es el encargado de buscar en los repositorios de Servicios Web Semánticos aquellos que satisfagan un objetivo dado. En caso de no encontrar servicios que, de forma individual, cumplan por completo las condiciones establecidas en el objetivo, el agente busca alguna composición de servicios (esto es, flujo de trabajo compuesto por servicios) que lo consiga.

3 ***Selection Agent:*** este agente es responsable de seleccionar los servicios (individuales o compuestos) más apropiados de entre aquellos pertenecientes a una lista de servicios dada. Más concretamente, se hace una ordenación de los servicios o conjuntos de servicios de la lista de acuerdo con un criterio de utilidad. Este criterio de utilidad se genera en base a las preferencias de los usuarios finales y a las condiciones expuestas por los mismos a la hora de establecer el objetivo.

4 ***Provider Agent:*** este agente actúa como representante de los proveedores de servicios. La organizaciones o entidades proveedoras de servicios establecen una serie de condiciones o preferencias de acuerdo con los objetivos estratégicos de dicha organización que deben de satisfacerse a la hora de ejecutar alguno de los servicios que este proveedor hace públicos. La misión de este agente es que estas condiciones sean tenidas en cuenta durante el proceso de negociación que tiene lugar entre consumidores de servicios y proveedores, para así cumplir los objetivos estratégicos de la organización.

5 ***Service Agent:*** este agente actúa como representante de los servicios disponibles. El agente tiene acceso a las preferencias o condiciones (esto es, propiedades no funcionales) establecidas por parte del proveedor de servicios para el servicio que este agente representa, y tiene en cuenta dicha información a la hora de entablar procesos de negociación con potenciales consumidores de servicios. En una fase

posterior, si el servicio que este agente representa resulta seleccionado por algún cliente para su ejecución, el agente será el encargado de realizar la invocación pertinente, asegurándose, en cada momento, de que las condiciones y preferencias mencionadas anteriormente son satisfechas en su totalidad durante todo el proceso de ejecución del servicio.

6 ***Broker Agent:*** este agente es el responsable de resolver los problemas de interoperabilidad que puedan surgir en las comunicaciones que tienen lugar entre los restantes agentes. Son cuatro los niveles de interoperabilidad a considerar que necesitan procesos de mediación: interoperabilidad de datos, de protocolos, de procesos y funcional. Estos cuatro problemas se describen con más detalle en la sección III.4.3. junto con los mecanismos concebidos para solventarlos. El agente en cuestión tiene como misión tratar, principalmente, la interoperabilidad de datos, aunque puede incorporar, a su vez, facilidades para permitir la mediación funcional y de procesos. En particular, la mediación de datos consiste en determinar una entidad ontológica (esto es, concepto, atributo, relación, etc.) perteneciente a una ontología "destino" a partir de otra entidad ontológica proveniente de una ontología "origen".

7 ***Framework Agent:*** este agente es el encargado de monitorizar el estado de los agentes en todo momento, así como controlar las interacciones entre los restantes componentes del sistema. Además, es responsable de gestionar el uso de los recursos disponibles, y se encarga de balancear la carga en todo momento para que el sistema sea lo más eficiente y escalable posible.

Los elementos descritos en la lista anterior son los tipos de agentes que contempla la plataforma y pueden haber varios agentes de un mismo tipo ejecutándose a la vez. Esto puede parecer obvio para los agentes que representan tanto a consumidores como a proveedores y servicios, pero también se produce este efecto en los restantes agentes. En particular, es posible que, por ejemplo, existan varias instancias del agente '*Discovery Agent*', cada una de ellas encargada de explotar la semántica de alguna de las distintas aproximaciones a los Servicios Web Semánticos (OWL-S, WSMO, SWSF, WSDL-S). Esto implica que se puede asegurar que la plataforma es totalmente

independiente de la tecnología de Servicios Web Semánticos elegida, siendo capaz, incluso, de adaptarse a un estándar que pueda originarse en el W3C.

Tipos de agente	Objetivos de diseño
Representantes de consumidores de servicios	
Customer Agent	*O1*. Reconocer las necesidades de los usuarios. *O2*. Reconocer las preferencias de los usuarios. *O3*. Obtener los resultados de las consultas realizadas por los usuarios. *O4*. Aprender por refuerzo (o sea, por observación) el modo de actuar de los usuarios para predecir consultas y mejorar los resultados. *O5*. Asegurar que se cumplen las condiciones establecidas en los contratos para la ejecución de servicios.
Discovery Agent	*O1*. Encontrar los servicios que cumplen un objetivo dado. *O2*. Encontrar una composición de servicios que cumplen un objetivo dado.
Selection Agent	*O1*. Negociar y determinar las condiciones exigidas para la ejecución de los servicios. *O2*. Ordenar de mayor a menor una lista de servicios de acuerdo con la utilidad esperada (utilidad calculada en base a las preferencias del usuario).
Representantes de proveedores de servicios	
Provider Agent	*O1*. Reconocer las preferencias (condiciones estratégicas) de los proveedores para la ejecución de los servicios que éste proporciona. *O2*. Indicar las condiciones estratégicas y de alto nivel por las que un proveedor accede a ejecutar alguno de los servicios que éste provee.
Service Agent	*O1*. Reconocer las preferencias (condiciones específicas) de los proveedores para la ejecución del servicio que este agente representa. *O2*. Negociar y establecer las condiciones exigidas para la ejecución del servicio al que representa de acuerdo con las pretensiones y preferencias del proveedor. *O3*. Invocar el servicio representado. *O4*. Aprender por refuerzo (o sea, por observación) la funcionalidad del servicio para, si es posible, automatizar la respuesta. *O5*. Asegurar que se cumplen las condiciones establecidas en los contratos para la ejecución de servicios.
Gestión de la plataforma	
Broker Agent	*O1*. Facilitar la interoperabilidad entre los distintos agentes de la plataforma. *O1.1*. Tratar la interoperabilidad de datos. *O1.2*. Tratar la interoperabilidad de procesos. *O1.3*. Tratar la interoperabilidad funcional.
Framework Agent	*O1*. Administrar y gestionar los recursos de la plataforma *O1.1*. Gestionar la suscripción y de-suscripción de proveedores. *O1.2*. Gestionar la suscripción y de-suscripción de consumidores. *O1.3*. Gestionar el registro y de-registro de servicios. *O1.4*. Crear los '*Service Agent*' que representan a cada uno de los posibles servicios a invocar. *O2*. Monitorizar el estado de los agentes y las interacciones entre los mismos.

Tabla 5. Objetivos que persigue cada tipo de agente

Tipos de agente	Entradas – Salidas – Interacciones
Customer Agent	
Entradas:	• Preferencias del usuario • Objetivos del usuario (multi-modal) • Retroalimentación del usuario
Salidas:	• Lista de servicios disponibles que satisfacen un objetivo dado (intermedio) • Resultados de la ejecución de los servicios
Interacciones:	• '*Discovery Agent*' o (→): objetivo para el que se debe de buscar servicios que lo realicen o (←): lista de servicios que cumplen un objetivo dado • '*Selection Agent*' o (→): objetivo del usuario o (→): preferencias del usuario final o (→): lista de servicios a ordenar de acuerdo con la utilidad esperada o (←): lista de servicios ordenados de acuerdo con su utilidad esperada en relación a las preferencias del usuario final • '*Service Agent*' o (→): descripción detallada del servicio a ejecutar (esto es, método, parámetros, localización, etc.) o (←): resultados de la ejecución del servicio
Discovery Agent	
Entradas:	• Objetivos del usuario
Salidas:	• Lista de servicios (o composición de servicios) disponibles que satisfacen el objetivo dado
Interacciones:	• '*Customer Agent*' o (←):objetivo para el que se debe de buscar servicios que lo realicen o (→):lista de servicios que cumplen un objetivo dado • Ontología de Servicios Web Semánticos o (→): consulta en base al objetivo buscado o (←): datos relativos a los servicios que satisfacen la consulta
Selection Agent	
Entradas:	• Preferencias del usuario • Objetivos del usuario • Lista de servicios (o composición de servicios) que satisfacen el objetivo dado
Salidas:	• Lista ordenada de servicios (o composición de servicios) de acuerdo con un valor de utilidad calculado con respecto a las preferencias del usuario y al objetivo inicial

Interacciones:

- '*Customer Agent*'
  - (←): objetivo del usuario
  - (←): preferencias del usuario final
  - (←):lista de servicios a ordenar
  - (→):lista de servicios ordenados de acuerdo con su utilidad esperada en relación a las preferencias del usuario final y el objetivo buscado
- '*Framework Agent*'
  - (→): lista de servicios sobre los que se deben de crear sus agentes representantes (o sea, '*Service Agent*')
  - (←): identificadores AID de todos los agentes que representan a los servicios
- '*Service Agent*'*
  - (→): petición de condiciones (p.ej. coste, calidad de servicio, horizonte temporal, etc.) para ejecución
  - (←): condiciones por las cuales el servicio se va a ejecutar

Provider Agent

Entradas:

- Preferencias de la entidad proveedora de servicios

Salidas:

- (Efecto) La ejecución de los servicios provistos por la entidad a la que representa este agente se hace de acuerdo con los objetivos estratégicos de la misma

Interacciones:

- '*Service Agent*'
  - (←): petición de condiciones/preferencias (de acuerdo con los objetivos estratégicos de la organización)
  - (→): condiciones/preferencias para la ejecución de servicios provistos por la entidad a la que representa este agente

Service Agent

Entradas:

- Preferencias del servicio al que representa (esto es, propiedades no funcionales)
- Descripción detallada del servicio a ejecutar (esto es, método, parámetros, localización, etc.)

Salidas:

- Resultado obtenido de la invocación del servicio
- (Efecto) La ejecución del servicio representado por este agente se hace de acuerdo con los objetivos estratégicos de la entidad proveedora y las propiedades no funcionales particulares del servicio.

Interacciones:

- '*Selection Agent*'
  - (←): petición de condiciones (p.ej. coste, calidad de servicio, horizonte temporal, etc.) para ejecución
  - (→): condiciones por las cuales el servicio se va a ejecutar – se tienen en cuenta tanto las propiedades no funcionales como objetivos estratégicos
- '*Provider Agent*'
  - (→): petición de condiciones/preferencias de la entidad proveedora del servicio
  - (←):condiciones/preferencias de acuerdo con los objetivos estratégicos de la entidad proveedora del servicio
- '*Customer Agent*'
  - (←): descripción detallada del servicio a ejecutar (esto es, método, parámetros, localización, etc.)
  - (→): resultados de la ejecución del servicio

Broker Agent

Entradas:

- Ontología "origen"
- Ontología "destino"
- Elemento X de la ontología "origen"

Salidas:

- Elemento Y de la ontología "destino" que se corresponde con el elemento X de la ontología "origen"

Interacciones:

- Cualquier agente** (*mediación de datos*)
  - (←): URI de la ontología "origen"
  - (←): URI de la ontología "destino"
  - (←): elemento de la ontología "origen"
  - (→): elemento de la ontología "destino"

Framework Agent

Entradas:

- Estado de los agentes
- Estado de los recursos del sistema
- Estado de las interacciones entre agentes del sistema

Salidas:

- Plan de choque y acciones a tomar ante los problemas que puedan haber surgido

Interacciones:

- Cualquier agente**
  - (→): petición de estado del agente
  - (←): informe de estado del agente (p.ej. bloqueado, en espera, en proceso, etc.)
- Cualquier agente **
  - (→): petición de estado de las interacciones
  - (←): informe de estado de las interacciones activas de este agente con el resto
- '*Selection Agent*'
  - (←): lista de servicios
  - (→): identificadores AID de todos los '*Service Agent*' creados que representan a los servicios recibidos

* A todos los agentes representantes de los servicios incluidos en la lista recibida como entrada
** La interacción con los restantes agentes es la misma y no se añade porque es redundante

Tabla 6. Entradas, salidas e interacciones de los agentes

En la Tabla 6, se muestran algunas de las interacciones que se producen entre los distintos agentes que constituyen el entorno. En la siguiente figura, se muestran gráficamente las interacciones que tienen lugar entre agentes y aquellas que tienen lugar entre agentes y los usuarios externos a la plataforma, consumidores de servicios y proveedores de servicios (ver Fig. 76).

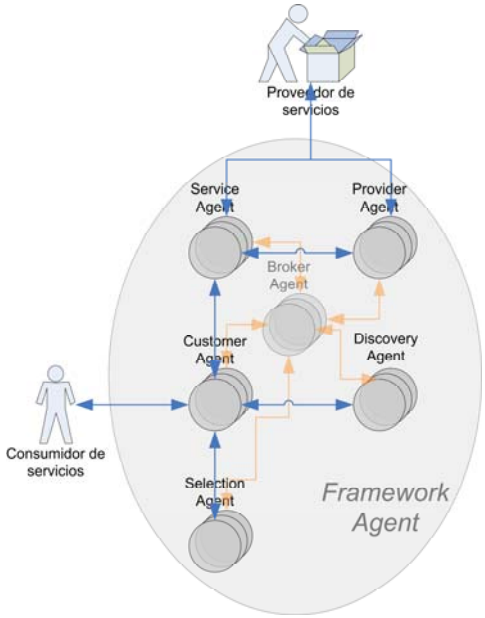


Fig. 76. Interacciones de los agentes

En la figura, se puede apreciar cómo el '*Framework Agent*' controla todo el proceso que tiene lugar entre los restantes agentes como si se tratara de una entidad superior. A su vez, el '*Broker Agent*' concibe interacciones con todos los agentes, dado que los problemas de interoperabilidad pueden ocurrir en cualquiera de ellos. Los usuarios consumidores de servicios sólo deben tratar con el '*Customer Agent*', mientras que los proveedores de servicios se relacionan tanto con el '*Provider Agent*' como con el '*Service Agent*'.

III.4.1.2. Roles

La descripción de los agentes que componen la plataforma presentada en el apartado anterior no es del todo exhaustiva. Con el propósito de alcanzar la flexibilidad pretendida por este marco de trabajo, se ha hecho uso de un mecanismo mediante el cual

la funcionalidad concreta y el modo de operación de cada uno de los agentes depende, en gran medida, de los roles que asuman estos agentes en tiempo de ejecución. Los roles, en el contexto de la Inteligencia Artificial y la tecnología de agentes, se pueden definir como encapsulaciones de propiedades y comportamientos dinámicos que pueden ser asumidas o "jugadas" por agentes. Los roles presentan ciertas propiedades que conllevan numerosas ventajas (Zhao et al., 2004): (i) son dinámicos y flexibles; (ii) están guiados por responsabilidades; y (iii) son sensibles al contexto. Con esta aproximación, se consigue disgregar la funcionalidad en múltiples componentes en forma de comportamientos, y se ofrece la posibilidad de que los agentes asuman nuevos roles en tiempo de ejecución de forma dinámica.

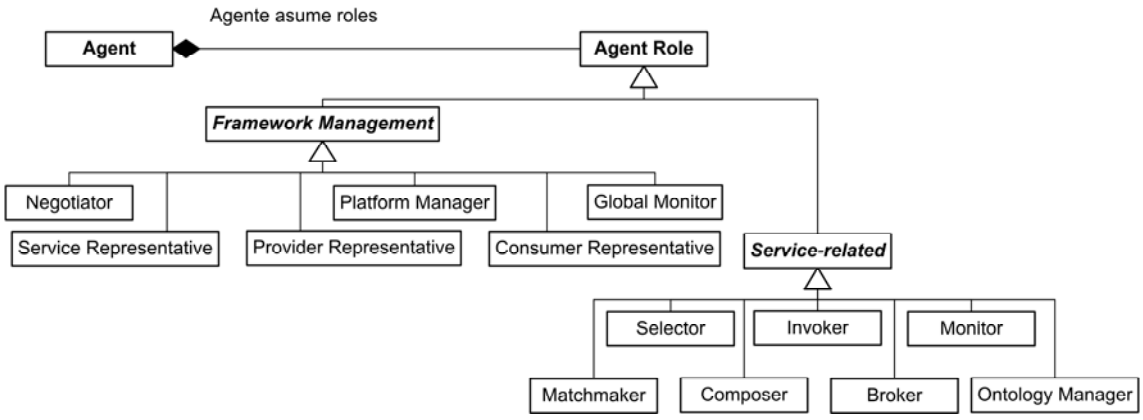


Fig. 77. Jerarquía de los roles que juegan los agentes

En la Fig. 77, se muestra una taxonomía con los roles que se han tenido en cuenta para la elaboración del marco de trabajo. Se han distinguido dos categorías principales de roles, aquellos que están relacionados con la gestión de los Servicios Web y las tareas relacionadas con el descubrimiento e invocación de los mismos, y aquellos que tienen que ver con la gestión integral de la plataforma y su funcionalidad a nivel de interacción con usuarios externos. En la Tabla 7, se muestra el listado de todos los roles junto con una breve descripción de los mismos.

Roles	Descripción
Roles para la gestión de servicios	
Broker	Representa la funcionalidad necesaria para resolver todo tipo de problemas de interoperabilidad y proporcionar mediación a distintos niveles (datos, procesos y funciones). Se ha incluido en esta categoría porque, en la mayoría de las ocasiones, los problemas de interoperabilidad surgen cuando la ontología de la que hace uso la plataforma y aquellas que se han utilizado para describir los servicios son distintas
Composer	Permite la composición de numerosos servicios en un flujo de trabajo integral con vistas a la consecución de un objetivo a partir de la ejecución de todos los servicios en el orden establecido. El proceso ideado en esta tesis doctoral se basa en la división del objetivo general en subobjetivos más específicos con la esperanza de que existan servicios que satisfagan por completo estos subobjetivos menos ambiciosos
Invoker	Se encarga de la invocación de un Servicio Web una vez se conocen tanto la operación a ser ejecutada como los parámetros esperados por la misma
Matchmaker	Este rol es el responsable de realizar el proceso de descubrimiento de aquellos servicios cuya descripción semántica se corresponda con los requisitos que imponga un objetivo especificado por el usuario final
Monitor	Este rol tiene como misión asegurar que los contratos establecidos en los procesos de negociación para la ejecución de los distintos servicios son satisfechos por ambas partes, proveedores de servicios y consumidores de servicios
Ontology Manager	Incluye la funcionalidad asociada con el acceso y manipulación de las ontologías. El propósito general es encapsular toda esta funcionalidad en un único componente y así evitar que otros elementos tengan que tratar con repositorios de ontologías específicos y mantener a la plataforma independiente de estos
Selector	Proporciona la funcionalidad necesaria para la ordenación de una lista de servicios de acuerdo con un valor de utilidad calculado en base a un conjunto de preferencias dado
Roles para la gestión de la plataforma	
Negotiator	Maneja las estrategias de negociación, dictando las propuestas a realizar en cada paso de la negociación y valorando las propuestas recibidas. A pesar de que, en la mayoría de las ocasiones, los procesos de negociación suceden entre los agentes que pretenden hacer uso de determinados servicios, se ha incluido en la categoría de gestión de la plataforma debido a que los restantes agentes también pueden hacer uso de este rol para establecer sus procesos de comunicación
Platform manager	Este rol se encarga de las tareas de control y balanceado de la carga de trabajo de todo el sistema
Provider Representative	Gestiona la interacción con el proveedor de servicios. A grandes rasgos, este rol se encarga de asegurarse que las condiciones establecidas por el proveedor como estratégicas en el discurrir de su negocio son satisfechas en todo momento
Service Representative	Gestiona las acciones que se toman en representación de los servicios, participando en los procesos de negociación que han de dar lugar a la ejecución de uno u otro servicio. Además, este rol es el encargado de mejorar la funcionalidad provista por el servicio cuando sea posible

Consumer Representative	Gestiona las acciones que se toman en representación de los usuarios consumidores de servicios. Entre sus objetivos se encuentran incorporar mecanismos que permitan al sistema determinar lo que el usuario desea en cada momento (esto es, transformación de los objetivos al formato interno esperado), así como las condiciones (esto es, preferencias) para su consecución. Además, es el encargado de procesar la salida para generar el formato que resulta más apropiado según el perfil del usuario final.
Global Monitor	Monitoriza los eventos que tienen lugar en la aplicación. Detecta los posibles problemas que surgen y determina las acciones a tomar para solventar los errores.

Tabla 7. Descripción de los roles jugados por agentes

En tiempo de ejecución, los agentes determinan qué roles deben de asumir y las conversaciones que han de establecer con otros agentes dependiendo de los objetivos que persigan en cada momento. Sin embargo, existen ciertos roles básicos que deben ser asumidos por cada uno de los agentes para que éstos no pierdan su identidad propia. En la Tabla 8, se muestra una asociación inicial entre roles y agentes que podrá variar en función de las necesidades de los agentes en cada momento y del estado del entorno.

La tarea de composición es un ejemplo claro que demuestra los beneficios de disociar agentes y comportamiento por medio de roles. Como se puede observar en la Tabla 8, es coherente incorporar en el mismo agente descubridor tanto el rol que realiza las búsquedas de los servicios, como aquel que es capaz de escindir un objetivo en diversos subobjetivos con el propósito de encontrar, para cada uno de esos subobjetivos, un servicio, de forma que la ejecución de todos estos servicios lleve a la consecución del objetivo original. Sin embargo, puede darse la circunstancia de que el usuario final disponga de una serie de preferencias por las cuales ese objetivo inicial deba de ser dividido, de modo que, en esta situación, el agente que representa al usuario el que deberá asumir este rol. Este cambio de roles sólo se podrá determinar en tiempo de ejecución.

Tipos de agentes	Roles asumidos
Customer Agent:	• Consumer Representative • Monitor • Broker • Ontology Manager
Discovery Agent:	• Matchmaker • Composer • Ontology Manager
Selection Agent:	• Selector • Negotiator • Ontology Manager
Provider Agent:	• Provider Representative • Monitor
Service Agent:	• Service Representative • Invoker • Negotiator • Monitor • Broker • Ontology Manager
Broker Agent:	• Broker • Ontology Manager
Framework Agent:	• Platform Manager • Global Monitor

Tabla 8. Asociación inicial entre agentes y los roles que estos asumen

III.4.1.3. Bases de Conocimiento

Las ontologías constituyen el elemento tecnológico clave de esta tesis doctoral. Se ha hecho uso de la tecnología ontológica como elemento mediador o intermediario entre los restantes componentes de la arquitectura, tal y como se representa en la Fig. 78. En particular, las ontologías operan como vocabularios universales, de forma que, tanto Servicios Web como agentes, comparten la misma interpretación de los términos contenidos del universo del discurso (esto es, dominio). Así, el sistema podrá beneficiarse de características tales como interoperabilidad y reusabilidad propias de la tecnología ontológica para alcanzar con éxito los objetivos marcados.

Fig. 78. Ontologías como núcleo de la arquitectura

Por tanto, para el desarrollo correcto y efectivo de las tareas que tienen lugar en el entorno de trabajo, es necesario acceder a cuatro fuentes de datos (ver Fig. 75) que contienen, en forma de ontologías, el conocimiento que será requerido durante las distintas fases de la ejecución del sistema:

1. ***Ontología de aplicación y del dominio***: la ontología de aplicación contiene el conocimiento esencial para poder modelar la aplicación particular que se desea proporcionar. El conocimiento incluido en la ontología de aplicación será, en general, no reutilizable, pero es crucial para asegurar que la asignación de tareas entre agentes se produce sin interpretaciones erróneas. Por su parte, la ontología del dominio constituye la conceptualización del dominio particular con el que trabaja la plataforma. En otras palabras, la ontología del dominio contiene los elementos de conocimiento (esto es, conceptos, relaciones, axiomas, etc.) referentes a los objetos que constituyen el universo del discurso. Gracias a esta ontología, los componentes del sistema podrán referirse a objetos del dominio sin incurrir en errores de interpretación.
2. ***Ontología de conocimiento local de agentes***: esta ontología contiene, para cada agente, el conocimiento que éste dispone acerca de su entorno, la tarea que tiene encomendada y los mecanismos y recursos de que dispone para satisfacer dicha tarea. Esta ontología depende, en gran medida, del tipo de agente y de los roles que éste asuma en cada instante. Así, por ejemplo, el '*Broker Agent*' debe tener acceso a conocimiento relacionado con las reglas de mapeo entre ontologías que le permitan resolver los problemas de interoperabilidad que pudieran surgir. Por su parte, los agentes '*Discovery Agent*' y '*Service Agent*' necesitan el

conocimiento concerniente a la aproximación de Servicios Web Semánticos que son capaces de manejar (dependiendo de cada caso: ontología OWL-S, ontología WSMO, ontología SWSO o etiquetas de extensión propuestas en WSDL-S). En general, la ontología de conocimiento local de cada agente estará construida en base a (o habrá sido extraída de) la ontología del dominio descrita anteriormente.

3. ***Ontología de negociación***: de forma similar a lo propuesto en (Tamma et al., 2005), esta ontología contiene los elementos que, tal y como se vió en la sección I.3.3, constituyen todo mecanismo de negociación: el protocolo de negociación y la estrategia. Con este método, se consigue independizar la plataforma de un mecanismo de negociación concreto, aportando flexibilidad para que éste sea elegido de forma dinámica en tiempo de ejecución dependiendo del problema a resolver. De este modo, cada vez que dos agentes vayan a entablar un proceso de negociación, éstos deberán ponerse de acuerdo en cuanto al protocolo y la estrategia a seguir, de forma que se maximice el resultado esperado. Esto lo harán teniendo en cuenta las particularidades del problema concreto al que se enfrentan.
4. ***Ontología de Servicios Web Semánticos***: en esta base de conocimiento se encuentran almacenadas las descripciones semánticas de las capacidades de los Servicios Web. Esta fuente de datos, al igual que las anteriores, puede estar situada localmente junto al entorno multi-agente, o estar dispuesta en un servidor remoto al que la plataforma tenga acceso. En el caso particular de las bases de conocimiento de Servicios Web Semánticos, es posible que se dé la situación de que existan varias fuentes de datos situadas remotamente y que, además, algunas de estas fuentes sean, en realidad, registros UDDI extendidos o mejorados para soportar la inclusión de descripciones semánticas. En todo caso, una vez los agentes tengan acceso a estas bases de conocimiento, éstos serán capaces de procesar las descripciones de los servicios y, así, automatizar las tareas de descubrimiento, composición, selección y ejecución de los mismos.

III.4.2. Integración con los Servicios

La misión última del entorno multi-agente descrito en la sección anterior es la de proporcionar acceso integrado y automático a los servicios de distinta índole que puedan estar dispersos por la red. Como ya se adelantó en la Fig. 48, y se puede observar con

mayor grado de detalle en la Fig. 79, se han identificado cuatro capas principales: capa de la lógica de negocio, capa de Servicios Web Semánticos, capa de Agentes Inteligentes y capa de aplicación. La plataforma propuesta en esta tesis doctoral abarca los dos niveles centrales, Agentes Inteligentes y Servicios Web Semánticos, permaneciendo, de este modo, independiente del dominio dónde ésta se quiera aplicar y de la aplicación que se quiera desarrollar sobre este dominio.

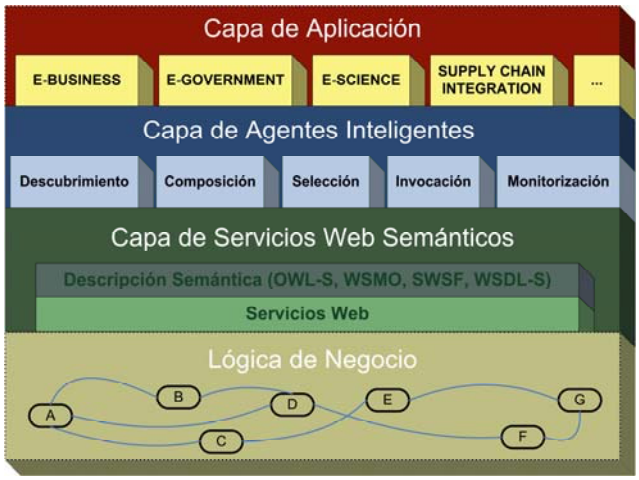


Fig. 79. Infraestructura multi-capa.

La capa inferior, denominada "capa de lógica de negocio", es la parte del sistema que se encarga de las tareas relacionadas con los procesos de negocio. Contiene los componentes que ofrecen la funcionalidad básica del sistema y las operaciones de procesamiento que serán utilizadas en capas superiores. Sobre esta capa de lógica de negocio se sitúan los Servicios Web. Éstos se encargan de exhibir públicamente la funcionalidad a la que se puede tener acceso en base a la lógica de negocio, permitiendo acceder a estas operaciones de forma global a través de Internet. En un nivel superior se emplaza la descripción semántica de los servicios, la cual consiste en añadir anotaciones semánticas a la descripción de las capacidades ofrecidas por los Servicios Web, de forma que éstas puedan ser procesadas de forma automática sin precisar intervención humana.

Los Servicios Web, junto con su descripción semántica, constituyen lo que se denomina "capa de Servicios Web Semánticos". La entidades encargadas de realizar el procesamiento automático de las descripciones semánticas de los Servicios Web son los

Agentes Inteligentes, que se sitúan en la capa denominada "capa de agentes inteligentes". Los agentes tienen la misión de hacer un uso apropiado de los servicios para proporcionar funcionalidad de alto nivel, tomando en consideración y siendo capaces de manejar el dinamismo propio de este tipo de entornos, donde nuevos servicios pueden aparecer mientras que otros pueden cambiar e, incluso, desaparecer. Entre las tareas que los agentes son capaces de realizar mediante el procesamiento de las anotaciones semánticas, se encuentran el descubrimiento, composición, selección, invocación y monitorización automática de los Servicios Web. Mediante este conjunto de tareas básicas los agentes son capaces de mostrar y propagar dinámicamente la funcionalidad cambiante que se encuentra disponible en las capas inferiores. Por último, la "capa de aplicación" es la responsable de organizar (esto es, orquestra y coordinar) los agentes, de tal forma que éstos pueden realizar tareas que sean de alguna utilidad para los usuarios finales. En este sentido, dependiendo de los agentes disponibles en el entorno, el modo en que éstos actúan conjuntamente y los servicios disponibles en cada momento en el sistema, se pueden obtener diferentes aplicaciones.

En esta sección, se detallan los mecanismos y técnicas incorporados en los agentes para cada una de las tareas básicas de gestión de servicios mencionadas anteriormente. Es conveniente destacar que no es propósito de esta tesis doctoral la elaboración de técnicas innovadoras para el desarrollo de estas tareas. En cambio, se ha analizado brevemente el estado del arte para cada una de estas operaciones y se ha incorporado una solución básica a la plataforma. Sin embargo, se han incluido las herramientas necesarias para permitir a los desarrolladores la incorporación de soluciones propias y nuevos algoritmos para cada una de estas tareas. El mecanismo, en la mayor parte de los casos, depende de la aproximación de Servicios Web Semánticos escogida, ya que, a pesar de que en algunos casos coinciden, sus diferencias pueden marcar la efectividad del método con el que se realizan dicho tipo de tareas.

III.4.2.1. Descubrimiento

Como se indicó en capítulos anteriores, uno de los mayores problemas asociados a la tecnología de los Servicios Web es que, conforme el número de servicios disponibles se incrementa, la dificultad para que los usuarios finales encuentren los servicios que le

interesan crece sustancialmente. Una de las ideas que motivan el concepto de los Servicios Web Semánticos es la automatización de este proceso. Por tanto, el descubrimiento en Servicios Web Semánticos consiste en encontrar los Servicios Web apropiados para resolver un objetivo procedente de la necesidad de un solicitante. Desde el origen de dicho concepto, varios investigadores han trabajado en el desarrollo de algoritmos que hagan esto posible (Paolucci et al., 2002; Keller et al., 2004; Grimm et al., 2006). En muchos casos, la investigación en este ámbito se ha basado en los resultados previos obtenidos en el campo de la Inteligencia Artificial y los sistemas multi-agente, donde los desarrolladores se tenían que enfrentar a un problema similar cuando se trataba de determinar el agente apropiado para la realización de una tarea concreta.

Se pueden distinguir varias técnicas de descubrimiento, dependiendo de variables como la computabilidad y la precisión. En (Keller et al., 2004) se destacan tres tipos distintos de descubrimiento, en función del nivel de utilización de la semántica (ver Fig. 80): descubrimiento basado en palabra clave, descubrimiento basado en semántica simple de descripciones de servicios, y descubrimiento basado en semántica rica de descripciones de servicios. Mediante una búsqueda basada en palabra clave, es posible realizar un filtrado rápido sobre una gran cantidad de servicios o hacer una clasificación de los mismos. El escenario típico para esta técnica consiste en hacer uso de un motor de consultas para el descubrimiento de servicios. Una consulta, en este caso constituida por un conjunto de palabras clave, se envía al motor de búsqueda y éste hace corresponder las palabras clave de la consulta con las palabras utilizadas para describir los servicios. Posibles mejoras de esta aproximación incluyen la utilización de diccionarios de sinónimos que permiten abrir el abanico de posibles servicios que satisfacen el objetivo buscado o el uso de diccionarios como WordNet, y técnicas de procesamiento de lenguaje natural que posibiliten la evaluación de la relevancia de los resultados de la búsqueda. En cualquier caso, es conveniente hacer uso de este mecanismo para reducir la lista de servicios sobre la que hacer búsquedas más precisas.

Las técnicas basadas en semántica simple y semántica rica precisan el modelado de objetivos y capacidades de servicios, en forma de ontologías. El descubrimiento basado en semántica simple de descripciones de servicios hace uso de estas ontologías para

realizar búsquedas basadas en palabra clave sobre un vocabulario controlado, el que proporcionan estas ontologías. Por su parte, el descubrimiento basado en semántica rica de descripciones de servicios explota todas las ventajas del modelado mediante ontologías, haciendo uso de mecanismos de razonamiento que permiten determinar con la mayor exactitud si un objetivo se corresponde con un servicio dado.

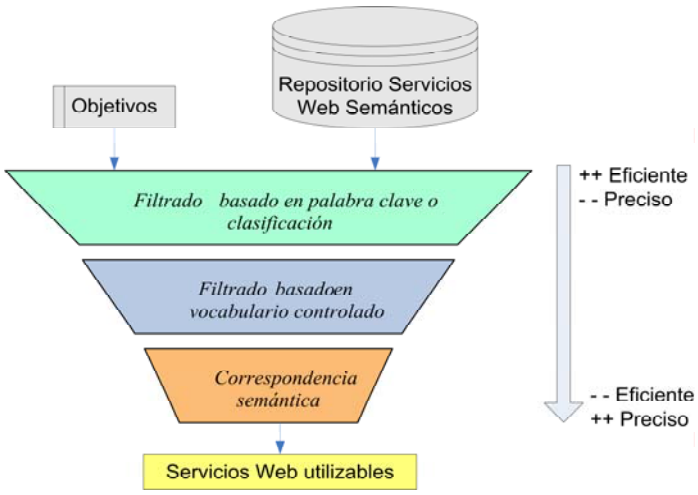


Fig. 80. Niveles de descubrimiento (adaptado de Stollberg & Haller, 2005)

Por analogía con la teoría de conjuntos, la correspondencia entre objetivos y Servicios Web puede suceder de múltiples formas (ver Fig. 81). El caso ideal es cuando existe correspondencia exacta entre lo que el servicio ofrece y lo que el objetivo requiere (Fig. 81-a). Otra opción que también puede ser de interés es cuando el servicio, además de resolver lo concerniente al objetivo, realiza otras tareas que nada tienen que ver con éste (Fig. 81-b). Si un servicio sólo es capaz de resolver una parte del objetivo, entonces será necesario buscar otros servicios que complementen la utilidad de éste. Esto puede ocurrir en dos situaciones: cuando los efectos del servicio son un subconjunto de todos aquellos esperados por parte del objetivo (Fig. 81-c), o que se produzca una intersección entre unos y otros (Fig. 81-d). Una última opción es la que contempla la posibilidad de que Servicio Web y objetivo no tengan nada que ver el uno con el otro (Fig. 81-e).

La alternativa que se ha tomado en esta tesis doctoral, sin esto repercutir en la posibilidad de que otras soluciones puedan incorporarse en el futuro, está alineada con el nivel de descubrimiento basado en semántica simple. Se ha diseñado una ontología muy sencilla para modelar internamente los objetivos a satisfacer por los componentes

del entorno en cada momento (ver Fig. 82). En base a los conceptos de esta ontología, y teniendo en cuenta el tipo de descripción semántica de servicios a la que accedemos (p.ej. OWL-S, WMSO, etc.), se realizan búsquedas de palabras clave controladas. Más concretamente, el descubrimiento se realiza haciendo corresponder las salidas esperadas por el objetivo con las salidas que produce el servicio. Si unas y otras coinciden, entonces se produce una correspondencia. En caso contrario, no.

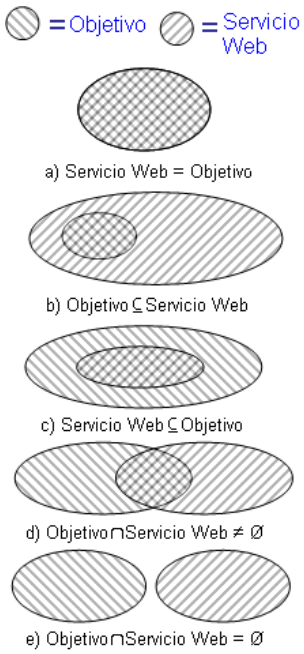


Fig. 81. Correspondencias posibles entre objetivos y Servicios Web (adaptado de Stollberg & Haller, 2005)

```
<rdf:RDF>
<owl:Ontology rdf:about=""/>
<owl:Class rdf:ID="Goal">
</owl:Class>
<owl:Class rdf:ID="Input">
</owl:Class>
<owl:Class rdf:ID="Output">
</owl:Class>
<owl:ObjectProperty rdf:ID="expect">
<rdfs:range rdf:resource="#Output"/>
<rdfs:domain rdf:resource="#Goal"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="recieves">
<rdfs:domain rdf:resource="#Goal"/>
<rdfs:range rdf:resource="#Input"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="inputName">
<rdfs:domain rdf:resource="#Input"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="inputValue">
```

```
<rdfs:domain rdf:resource="#Input"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="name">
<rdfs:domain rdf:resource="#Goal"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="outputValue">
<rdfs:domain rdf:resource="#Output"/>
</owl:DatatypeProperty>
```

Fig. 82. Modelo ontológico de los objetivos

III.4.2.2. Composición

La composición es otro de los temas recurrentes en la investigación sobre Servicios Web. Antes del surgimiento de la Web Semántica imperaban soluciones en las que el usuario era el encargado de establecer las relaciones entre los distintos servicios. En este campo, el lenguaje BPEL ('*Business Process Execution Language*') y, más concretamente, su especialización para el manejo de Servicios Web BPEL4WS ('*BPEL for Web Services*'), posteriormente denominado WS-BPEL[19] (estándar OASIS desde Abril de 2007), es el que ha gozado de mayor éxito en el entorno industrial, convirtiéndose en un estándar *de facto*. Mediante este lenguaje, y haciendo uso de su gran expresividad, es posible especificar, con un alto grado de detalle, flujos de trabajo a partir de Servicios Web que representan procesos de negocio.

Sin embargo, se ha trabajado durante muchos años en el desarrollo de mecanismos que permitan la automatización de este proceso de composición y, con especial énfasis, desde el origen de los Servicios Web Semánticos (McIlraith & Son, 2002; Abela et al., 2003; Sheshagiri et al., 2003; Sirin et al., 2004; Kuter et al., 2005). En la mayor parte de los casos, estas soluciones hacen uso de técnicas de planificación para resolver la composición. Además, es una práctica común en la comunidad investigadora en el área la integración de la composición con los procesos de búsqueda y selección de servicios, para así disponer de una solución integrada para el descubrimiento de Servicios Web Semánticos.

En esta tesis doctoral se ha seguido este modus operandi, generando un mecanismo integrado para el descubrimiento que consiste en realizar la búsqueda de servicios y su composición en un mismo flujo de ejecución. El algoritmo utilizado se encuentra

[19] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

descrito a un alto nivel de abstracción en la Fig. 83. El método recibe como entrada el objetivo que representa las necesidades del usuario. En primer lugar, se realiza la búsqueda de los servicios que se corresponden con dicho objetivo (*corresponder*) y, si se encuentra alguno, entonces el método termina. En caso de no encontrar ningún servicio apropiado para el objetivo inicial, éste se descompone en subobjetivos (*descomponer*), que serán nuevamente procesados para encontrar servicios que los satisfagan. El mecanismo de descomposición de objetivos se basa en técnicas de planificación. El método finaliza cuando se han encontrado servicios adecuados para cada uno de los subobjetivos.

```
descubrir(objetivo,listaServicios){
descubiertos = lista vacía
descubiertos = corresponder(objetivo)
si (vacio descubiertos){
   listaSubObjetivos = lista vacía
   listaSubObjetivos = descomponer(objetivo)
   para todo subobjetivo de listaSubObjetivos hacer{
      listaServicios.añadir(subobjetivo, descubrir(subobjetivo))
      }
   } si no{
   listaServicios.añadir(objetivo, descubiertos)
   }
devolver descubiertos
}
```

Fig. 83. Algoritmo para el descubrimiento y la composición

Nuevamente es importante destacar que, si bien el flujo de trabajo para los procesos de descubrimiento y composición están preestablecidos del modo en que se especifica en el algoritmo presentado en la Fig. 83, las operaciones para búsqueda (*corresponder*) y composición (*descomponer*) de servicios pueden estar implementadas por el desarrollador.

III.4.2.3. Selección

La selección en el entorno de los Servicios Web (semánticos) se refiere al proceso mediante el cual un servicio es seleccionado de entre una lista de servicios capaces de realizar una misma tarea después de que todos hayan sido evaluados de acuerdo con algún criterio de selección. Como ya se resaltó para las tareas de descubrimiento y composición, antes de la aparición de los Servicios Web Semánticos, eran usuarios humanos quienes se encargaban de determinar qué servicios ejecutar dirimiendo las

eventuales disyuntivas de acuerdo con algún criterio en base a la información disponible sobre los servicios.

Propiedades no funcionales	Descripción
Proveedor del servicio	Contiene información sobre la identificación del servicio, su nombre y el proveedor del servicio
Modelo temporal	Proporciona los conceptos necesarios para las descripciones de los servicios relacionadas con el tiempo: *Fecha Temporal*, *Hora*, *Intervalo Temporal* y *Duración Temporal*. Estos conceptos pueden ser, a su vez, refinados
Modelo locativo	Esta propiedad se utiliza para modelar la localización de un servicio. Entre los conceptos relacionados con este modelo se incluyen los siguientes: *Dirección*, *Región*, *Ruta*, *Punto*, *Referencia a la calle del directorio*, *Número de teléfono*, *URI* y *Dirección IP*
Disponibilidad del servicio	Combina aspectos locativos y temporales para describir cuándo y dónde se puede interactuar con el servicio
Obligaciones	Este modelo captura tanto las responsabilidades del solicitante del servicio como las del proveedor. Se distinguen tres tipos de obligaciones: *Obligaciones de precio*, *Obligaciones de pago* y *Obligaciones de relación*
Precio	Esta propiedad representa el coste asociado a la utilización del servicio pertinente
Pago	Captura el modo en que el solicitante del servicio puede llevar a cabo las obligaciones de pago
Descuentos	Representan descuentos que pueden percibir los solicitantes de servicios. Están categorizados en base al método de pago y la identidad del solicitante
Penalizaciones	Son utilizadas tanto por el proveedor de servicios como por el solicitante para especificar qué ocurre si la otra parte no cumple con una obligación
Derechos	Indica los permisos para realizar operaciones para proveedores y solicitantes
Lenguaje	Dividido en tres categorías: *lenguaje escrito* (p.ej. español), *lenguaje hablado* (p.ej. español) y *lenguaje estándar* (p.ej. WSDL)
Confianza	Depende directamente del modelo de confianza utilizado
Calidad	Se describe en relación a un estándar, pruebas industriales o esquema de calificación
Seguridad	Está estrechamente relacionado con el modelo locativo del servicio y se encuentra dividido en dos dimensiones: *identificación* y *confidencialidad*

Tabla 9. Listado de propiedades no funcionales (O'Sullivan et al., 2005; Toma & Foxvog, 2006)

Los Servicios Web Semánticos hacen posible la automatización de este proceso de selección. Para esto, la mayor parte de las aproximaciones han incluido, entre sus especificaciones, lo que se denominan "propiedades no funcionales" que, a diferencia del resto de elementos, ofrecen información que nada tiene que ver con el comportamiento o la funcionalidad que proporciona el Servicio Web, ni con el modo de

acceder al mismo en caso de desear invocarlo. En cambio, estas propiedades indican otras características propias del servicio, como el precio, la confianza, la calidad del servicio, etc. (ver Tabla 9).

Aunque estas propiedades no funcionales tienen también relevancia para el desarrollo del descubrimiento y la monitorización, se utilizan principalmente en todo lo concerniente al proceso de selección de servicios y, más concretamente, en la negociación.

El mecanismo de selección de servicios propuesto en esta tesis doctoral se basa en un proceso de negociación donde, además de las propiedades no funcionales de las descripciones semánticas de los servicios, se tienen en cuenta las preferencias establecidas por los usuarios finales (esto es, consumidores de servicios) y las que, opcionalmente, hayan podido incluir los proveedores. Cuando el descubrimiento y la composición han concluido, comienza la selección. El resultado de estas fases es una lista de conjuntos de servicios. Cada conjunto de la lista es una composición de servicios (o, si se da el caso, un servicio únicamente) cuya ejecución, en el orden establecido y asumiendo la ausencia de fallos en el transcurso de la operación, daría lugar a la consecución del objetivo que había establecido el usuario. Por tanto, el sistema sólo necesitará ejecutar uno de estos flujos de trabajo para llevar a cabo su propósito y tendrá que elegir el que proponga mejores condiciones para el usuario.

Así, en el entorno multi-agente desarrollado en esta tesis doctoral, el '*Selection Agent*' arranca un proceso de negociación con cada uno de los agentes '*Service Agent*' involucrados (esto es, aquellos que representan a alguno de los servicios incluidos en la lista). Los agentes '*Service Agent*' preparan sus propuestas de acuerdo con las propiedades no funcionales de los servicios a los que representan y a las condiciones establecidas por parte del proveedor (esta información se solicita a los agentes '*Provider Agent*' pertinentes). El '*Selection Agent*' valora todas las propuestas recibidas utilizando técnicas provenientes de la teoría de la utilidad. En particular, se ha elaborado una función de utilidad que relaciona las condiciones y preferencias introducidas por los usuarios finales con las propuestas de cada uno de los servicios para generar una ordenación total de la lista de servicios con respecto al valor de utilidad obtenido. Por último, el usuario final debe examinar la lista de servicios que ha sido previamente

ordenada por la aplicación y determinar qué servicio/s ejecutar. Nuevamente, el modo en que se valora cada propuesta con respecto a las preferencias del usuario puede ser establecido fácilmente por parte de los desarrolladores.

III.4.2.4. Invocación

La invocación es la tarea mediante la cual se ejecuta un método específico de un Servicio Web y se recogen los resultados (por parte del invocador). Para el desarrollo correcto de esta tarea, es necesario conocer los siguientes elementos: URL del Servicio Web, nombre del método, parámetros del método y valores de los parámetros para la invocación. Esta información está disponible en la descripción semántica del servicio y en el objetivo del usuario. Por un lado, los datos relativos a la dirección para invocación del servicio, nombre de método y la signatura del mismo (esto es, parámetros de entrada y salida) se pueden obtener de la sección '*Grounding*' de las descripciones semánticas de las capacidades de los servicios. Por otra parte, los valores a asociar a cada parámetro de entrada del método a invocar estarán, en algunos casos, incluidos en la descripción del objetivo del usuario. En caso contrario, para poder ejecutar de forma satisfactoria el servicio, será necesario solicitar esta información al usuario final.

El agente encargado de la tarea de invocación en el sistema diseñado como parte de esta tesis doctoral es el '*Service Agent*'. En particular, el proceso de ejecución del servicio se lleva a cabo a través del rol '*Invoker*'. Este rol recibe como entrada una descripción pormenorizada del servicio a invocar. La descripción del servicio se obtiene durante la fase de descubrimiento, momento en que se crea un modelo para cada servicio a analizar (esto es, aquellos que satisfacen el objetivo requerido) que incluye la dirección del servicio, nombre del método y parámetros necesarios para su invocación. Adicionalmente, el rol '*Invoker*' toma como entrada el objetivo para el cual se ejecutará el servicio. El objetivo es procesado y los valores de los parámetros para la invocación del servicio extraídos. Si alguno de los valores de los parámetros necesarios fuera desconocido después de esta etapa, ocurrirá una de las siguientes situaciones:

- La información requerida para la invocación del servicio (o parte de la misma) se debe de producir como resultado de la invocación de otro servicio. En tal caso, el '*Service Agent*' pide al '*Customer Agent*' los datos necesarios y éste, una vez

recibidos los resultados de la invocación correspondiente, envía al '*Service Agent*' lo que éste necesita.

- La información requerida para la invocación del servicio (o parte de la misma) puede ser extraída de la lista de preferencias que el usuario estableció para esa consulta o las preferencias generales. En tal caso, el '*Service Agent*' solicita los datos para la invocación al '*Customer Agent*', que recoge tal información y se la envía de forma inmediata al '*Service Agent*'.
- La información requerida para la invocación del servicio (o parte de la misma) no se encuentra disponible en el sistema. En tal caso, el '*Service Agent*' envía un mensaje con los datos que le son imprescindibles para la invocación al '*Customer Agent*', que se encarga de solicitar tal información al usuario final. Una vez recogidos los datos pertinentes, el '*Customer Agent*' los envía de vuelta al '*Service Agent*', quien podrá continuar con la ejecución del servicio.

Cuando se ejecuta el servicio , los resultados se le devuelven al '*Customer Agent*', que se encarga de recolectar todos los resultados parciales y generar un resultado integrado último que le será entregado al usuario final de acuerdo con sus preferencias en la presentación de resultados.

III.4.2.5. Monitorización

La tarea de monitorización en el entorno de los Servicios Web cumple dos propósitos fundamentales. En primer lugar, controla que no se produzca ningún tipo de error durante la ejecución del servicio (p.ej. fallo en la red, error en el servicio, desconexión del servicio, etc.) y, en caso de ocasionarse, notifica los datos referentes al mismo a la entidad encargada de tomar medidas al respecto. La segunda meta de esta tarea es la de asegurar que ambas partes, proveedores y consumidores, cumplen las condiciones del contrato de ejecución del servicio. Como se mencionó anteriormente, durante la fase de selección de servicios se establece, en muchos casos, un proceso de negociación en el que las preferencias, tanto de consumidores como de proveedores, son evaluadas y, en base a éstas, se selecciona un servicio por encima del resto. En otros casos, sin entablar un proceso de negociación como tal, se evalúan las propiedades no funcionales de las

descripciones semánticas de los servicios y, en base a esta información, se determina cuál es el servicio más apropiado. En ambos casos, el resultado de la fase de selección es el servicio a invocar y las condiciones por las cuales esa invocación debe de tener lugar (esto es, el contrato que se debe satisfacer). Ambas partes deben de cumplir este contrato y, en caso contrario, podrán ser requeridas acciones de compensación por la otra parte.

En el marco de trabajo elaborado como parte de esta tesis doctoral, esta tarea ha sido diseñada de forma distribuida entre distintos componentes. Como se destacó con anterioridad, el '*Selection Agent*' negocia con los agentes '*Service Agent*' que representan a los servicios que se están examinando. De este proceso de negociación, resultan seleccionados el servicio o los servicios a ejecutar junto con las condiciones por las que estos servicios han accedido a proporcionar su funcionalidad y han sido elegidos. Esta información le llega al '*Customer Agent*', representante del usuario que inició la interacción, quien debe ponerse en contacto con los agentes '*Service Agent*' de los servicios seleccionados para comenzar la ejecución de los mismos. En este punto, tanto el agente representante del usuario, como los representantes de los servicios, comienzan el proceso de monitorización por el que cada parte se asegura que la otra parte implicada está cumpliendo lo establecido en el contrato. Asimismo, los agentes '*Provider Agent*', representantes de los proveedores cuyos servicios estén en ejecución, se encargan de controlar que las condiciones preestablecidas con arreglo a los objetivos estratégicos de la organización también sean satisfechas en este proceso.

III.4.2.6. Negociación

La tarea de negociación no está generalmente incluida entre aquellas básicas para el manejo de los Servicios Web. En realidad, esta tarea puede verse como parte de otros procesos más básicos como lo es la selección. Sin embargo, los entornos de ejecución de Servicios Web Semánticos desarrollados hasta la fecha no incorporan esta funcionalidad o, si lo hacen, proporcionan un mecanismo poco flexible incapaz de adaptarse a los requerimientos y condiciones del entorno en cada instante.

El uso de agentes inteligentes para el control y gestión de Servicios Web propuesto en esta tesis doctoral conlleva, entre sus múltiples ventajas, tener disponible de forma

automática los elementos necesarios para llevar a cabo procesos de negociación. En el área de los Sistemas Multi-Agente, en el que múltiples entidades autónomas compiten por el uso de recursos o cooperan para llegar a metas comunes, la negociación es un mecanismo básico e imprescindible que permite a estas entidades coordinarse. Por consiguiente, existen entre las especificaciones FIPA y, en concreto, entre aquellas dedicadas a la descripción del lenguaje de comunicación entre agentes ('*Agent Communication Language*' o ACL), unas dedicadas a los protocolos de interacción[20] donde se enumeran los protocolos de los que se puede hacer uso en un entorno multi-agente para entablar procesos de negociación. Por lo tanto, todas las plataformas multi-agente construidas conforme al estándar FIPA, incorporan las herramientas necesarias para que los agentes puedan coordinarse a través de mecanismos de negociación.

En esta tesis doctoral, se propone la utilización de un mecanismo de negociación que, utilizando como base conceptual los elementos de la especificación de FIPA, añade los elementos necesarios para dotar a la solución de la flexibilidad y dinamismo que son requisitos indispensables en todo entorno de Servicios Web (semánticos). Por tanto, aplicando la idea propuesta por los autores en (Tamma et al., 2005), se ha desarrollado una ontología de negociación que incluye diversos protocolos y estrategias de negociación. Es en tiempo de ejecución, cuando los agentes, dependiendo de las características de la aplicación y las condiciones del entorno en cada momento, adoptan un acuerdo sobre el uso de uno u otro mecanismo de negociación (esto es, protocolo y estrategia de negociación) de modo que se obtenga la solución óptima en cada interacción.

III.4.3. Mediación

Se puede dar la situación, especialmente en entornos abiertos y distribuidos, como Internet, de que sistemas o componentes, no diseñados para operar conjuntamente, precisen iniciar un proceso de comunicación para, generalmente, compartir la funcionalidad ofrecida por cada elemento. En estas situaciones, la heterogeneidad inherente a estos entornos puede suponer un grave problema. En particular, es necesario

[20] http://www.fipa.org/repository/ips.php3

tratar diversos niveles de incompatibilidad: estructural, semántica y conceptual. Con el objetivo de superar esta dificultad, surgió el concepto de *mediación* (Wiederhold, 1994). La mediación se puede definir como el mecanismo mediante el cual los distintos problemas de incompatibilidad son solventados.

En esta tesis doctoral, se consideran cuatro tipos de mediación: (i) mediación de datos, (ii) mediación de protocolos, (iii) mediación de procesos y (iv) mediación de funcionalidad. A nivel de datos, se trabaja en la mediación de fuentes de datos heterogéneas. Por su parte, la mediación entre patrones de comunicación heterogéneos se produce a nivel de protocolos. A nivel de procesos, en cambio, se realiza la mediación de procesos de negocio. Por último, a nivel de funcionalidad se median componentes distribuidos heterogéneos.

En esta sección, se da cuenta de las soluciones desarrolladas para tratar el problema de la heterogeneidad en cada uno de los niveles mencionados. También se destaca la participación tanto de roles como de agentes en estos procesos.

III.4.3.1. Mediación de datos

Actualmente, existen innumerables fuentes de datos accesibles a través de Internet, cada una de las cuales hace uso de mecanismos propios para la representación de dichos datos que dificultan la interoperabilidad con otros sistemas. Además, la semántica que se le confiere a cada elemento de datos representado puede variar de un sistema a otro. Esta heterogeneidad en entornos abiertos y distribuidos como Internet complica la comunicación entre sistemas que no hayan sido inicialmente diseñados para cooperar.

La Web Semántica, esto es, la adición de información semántica sobre los datos publicados en la Web, es una de las soluciones adoptadas para enfrentarse a este problema de heterogeneidad. El uso de ontologías para la representación del conocimiento permite tratar el problema de la incompatibilidad en la representación de los datos, aportando un lenguaje estándar para este propósito. Sin embargo, también existen problemas relacionados con la semántica. En particular, se puede dar el caso de que dos entidades modelen un mismo elemento del mundo real de diferente manera (ver Fig. 84). La pregunta a responder es cómo se pueden integrar aplicaciones que utilizan diferentes conceptualizaciones del mismo dominio.

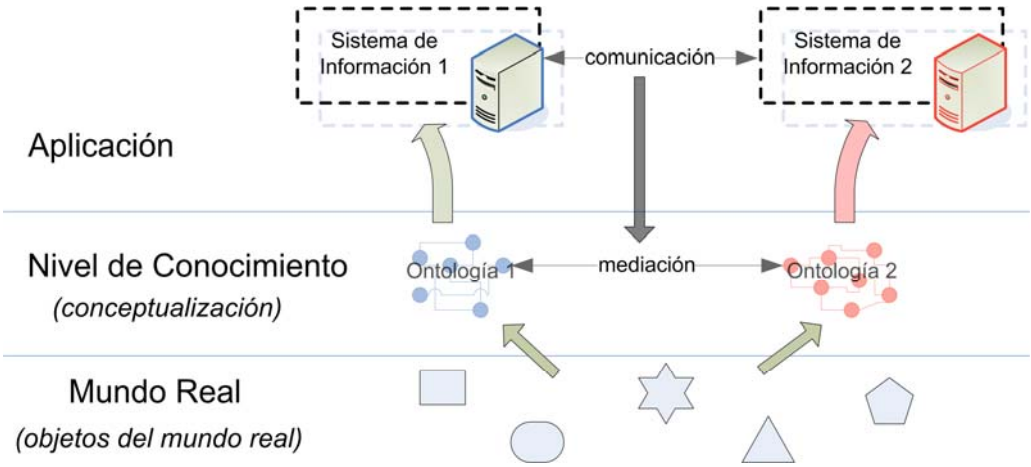


Fig. 84. Heterogeneidad de datos en la Web Semántica

En la Fig. 84 se puede observar cómo dos aplicaciones que hacen uso de distintas ontologías para representar los mismos objetos del mundo real, precisan de un proceso de mediación si han de establecer comunicación entre sí. La *mediación de datos* es, por tanto, el proceso por el cual los elementos (conceptos, relaciones, instancias) de una ontología fuente son transformados en elementos de una ontología destino. Las soluciones más comunes para este tipo de problemas pasan por la definición de reglas de mapeo entre dos ontologías (‘*ontology mapping*’) (Kalfoglou & Schorlemmer, 2003) y la combinación de ontologías fuente para generar una nueva ontología donde una combinación del conocimiento disponible en las fuentes se encuentra representado (‘*ontology merging*’) (Noy & Musen, 2000). Las técnicas para encontrar conceptos que se puedan mezclar (‘*merge*’) en diferentes ontologías y para encontrar correspondencias (‘*mappings*’) entre los conceptos de diferentes ontologías, son parecidas ya que ambas se basan en una medida de similitud de conceptos. De hecho, las reglas de mapeo entre dos ontologías se pueden utilizar como base para generar una ontología combinada.

El marco de trabajo diseñado en esta tesis doctoral asume la existencia de una base de conocimiento donde se encuentran almacenadas las reglas de mapeo que, en todo momento, pueda necesitar la aplicación para su correcto funcionamiento. Concretamente, tal y como se señaló en la sección III.4.1.3, este tipo de conocimiento (esto es, el expresado en forma de reglas) forma parte del conocimiento local del ‘*Broker Agent*’. Este agente es el responsable último de realizar la mediación de datos ante problemas de interoperabilidad que puedan surgir entre los restantes agentes.

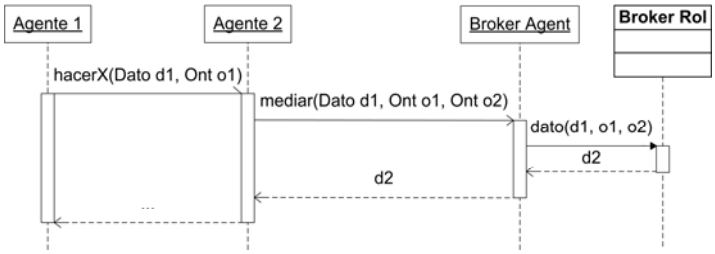


Fig. 85. Mecanismo para la mediación de datos

El mecanismo utilizado en el marco de trabajo para la mediación de datos y las entidades involucradas se presenta en la Fig. 85. El '*Broker Agent*', a través del comportamiento que persigue la mediación de datos, y el rol '*Broker*' son los responsables de llevar a cabo este proceso. Cuando un agente cualquiera de la plataforma recibe un dato que es incapaz de interpretar conforme al conocimiento de que éste dispone, le envía la petición de mediación al '*Broker Agent*'. Éste delega esta tarea en el rol '*Broker*', que se encarga de recoger el elemento de la ontología destino que se corresponde con el solicitado de la ontología de origen y, todo esto, conforme a las reglas de mapeo disponibles en la base de conocimiento. Como se comentó anteriormente, mecanismos más elaborados para la realización de la mediación de datos pueden ser incorporados por los desarrolladores extendiendo e implementando nuevos tipos del rol '*Broker*'.

III.4.3.2. Mediación de protocolos

La mediación de protocolos es necesaria cuando se trata con patrones de comunicación heterogéneos (Williams et al., 2005). En la Fig. 86, se presenta un ejemplo donde los procesos de dos entidades se comunican para realizar algún proceso conjunto. Cada uno de los servicios utiliza protocolos diferentes (p.ej. EDIFACT y RosettaNet), por lo que se precisa de un componente capaz de mediar entre ambos protocolos.

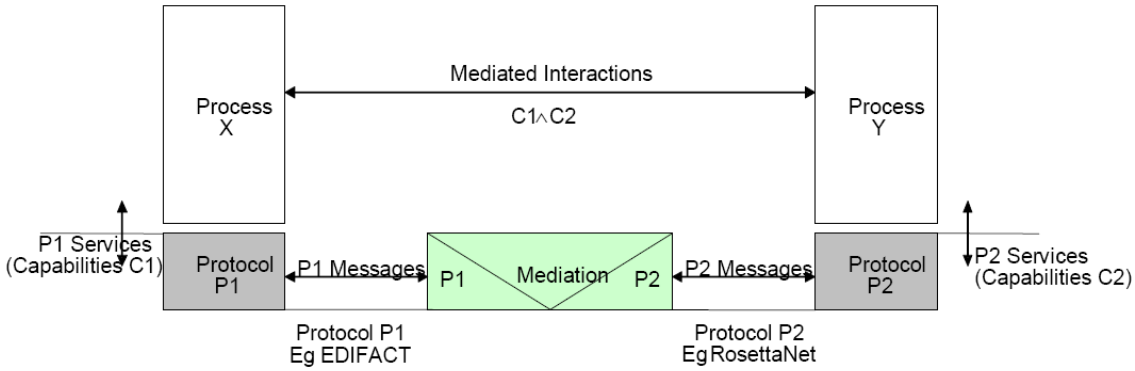


Fig. 86. Modelo conceptual de la mediación de protocolos (Williams et al., 2005)

En el marco de trabajo elaborado en esta tesis doctoral, pueden tener lugar dos tipos de interacciones, a saber, interacciones agente-agente (ver Fig. 87) e interacciones agente-servicio (el marco se abstrae de las posibles invocaciones servicio-servicio que se puedan producir externamente). La comunicación al nivel más básico entre agentes no supone ningún problema, ya que tiene lugar en un entorno controlado, como es un entorno multi-agente, que proporciona los mecanismos para el transporte de mensaje mediante protocolos (p.ej. IIOP, ORBacus, HTTP) que satisfacen las especificaciones FIPA. Lo mismo sucede en la comunicación a nivel de performativas. Esta interacción sucede de acuerdo con un protocolo de negociación de entre los especificados en el estándar FIPA. Dado que los agentes habrán concertado la utilización de un mecanismo de negociación particular antes de entablar la comunicación, en ningún momento será necesaria la mediación de protocolos.

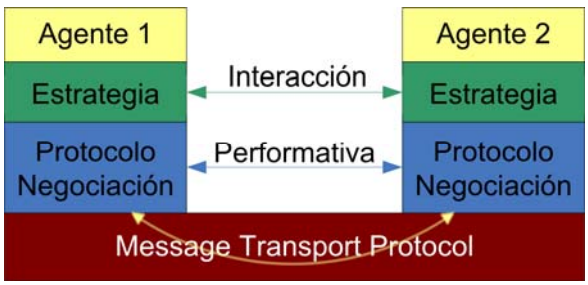


Fig. 87. Comunicación a nivel de protocolos en Sistemas Multi-Agente

En cuanto a la interacción agente-servicio, el marco de trabajo aquí descrito tampoco ha considerado la utilización de uno u otro protocolo. En cambio, en la plataforma nos abstraemos de este nivel de detalle, de modo que los servicios se conciben como

componentes sobre los que se pueden ejecutar métodos que proporcionan una funcionalidad determinada.

III.4.3.3. Mediación de procesos

La mediación de procesos tiene lugar cuando es necesario mediar entre procesos de negocio heterogéneos (Cimpian & Mocan, 2005). En concreto, la mediación de procesos se ocupa de determinar cómo pueden acoplarse dos procesos públicos para proporcionar una determinada funcionalidad. En otras palabras, se trata de hacer que varios procesos de negocio sean interoperables. El problema viene representado gráficamente en la siguiente figura (ver Fig. 88):

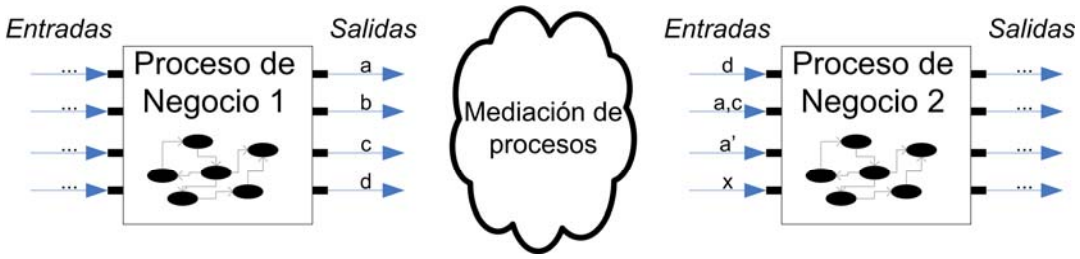


Fig. 88. Problema que requiere mediación de procesos

Como es apreciable en la Fig. 88, se pueden encontrar distintos tipos de problemas en la interacción entre dos procesos. Cimpian y Mocan (2005) destacan los siguientes:

- Mensaje no esperado: esto sucede cuando uno de los participantes envía un mensaje que el otro no espera recibir. La solución radica en que el mediador detenga dicho mensaje.
- Orden de mensajes inverso: situación en la que uno de los participantes envía los mensajes en un orden diferente al esperado por el otro participante. El mediador debe, en tal caso, reordenar los mensajes que se envían entre los participantes.
- División de mensaje: escenario en que uno de los participantes envía en un único mensaje con múltiple información integrada, mientras que el otro espera recibir dicha información pero en diferentes mensajes. En esta situación, el mediador debe dividir el mensaje original en tantos mensajes como los esperados por el participante destino de la interacción, incluyendo en cada mensaje la información apropiada.

- Combinación de mensajes: caso opuesto al anterior. Un participante envía información en múltiples mensajes cuando el otro participante espera esta información en un único mensaje. La solución, en este caso, pasa por integrar los mensajes de origen en un único mensaje.
- Falta de reconocimiento: pueden surgir situaciones de bloqueo cuando un participante espera un mensaje de reconocimiento (*ack*) pero el otro no lo envía. El mediador puede generar de forma automática el mensaje de reconocimiento y enviarlo de vuelta al iniciador de la interacción.
- Mensaje no existente: es la situación más difícil de manejar. Ocurre cuando un participante espera un mensaje con una información que el otro participante no ha enviado (ni tiene la intención de hacerlo). El mediador, al no disponer de la información necesaria para enviar el mensaje al participante receptor, es incapaz de solucionar este problema.

Una clasificación más abstracta es la propuesta en (Fensel & Bussler, 2002), donde se distinguen tres posibles casos de incompatibilidades en el intercambio de mensajes:

- Correspondencia exacta: dos participantes comparten el mismo patrón en la realización de su proceso de negocios, de forma que cada uno envía exactamente los mensajes que el otro espera en cada momento.
- Incompatibilidad resoluble: en este caso, los dos participantes utilizan diferentes patrones de intercambio y, para poder llevar a cabo la interacción, se deben de realizar numerosas transformaciones.
- Incompatibilidad no resoluble: uno de los participantes espera un mensaje que el otro no pretende enviar. Si el mediador es incapaz de generar este mensaje, la comunicación llega a una situación de punto muerto.

El marco de trabajo propuesto en esta tesis doctoral, basado en la integración de la tecnología de agentes y Servicios Web descritos semánticamente, es capaz de superar esta deficiencia restringiendo las interacciones que pueden tener lugar y las entidades que pueden participar en las mismas. En primer lugar, y como ya se destacó en el apartado anterior, en la comunicación agente-agente no se pueden dar

incompatibilidades en el proceso. Esto es debido a la utilización de protocolos estándar, que indican la secuencia de mensajes permisible (ver Fig. 89), previamente acordados entre las partes. Por otro lado, la interacción agente-servicio está restringida al '*Service Agent*', único agente que dispone de los medios apropiados para hacer uso de los servicios (ver Fig. 90). Para esto, el agente recupera, de la base de conocimiento de Servicios Web Semánticos, la información relativa a las entradas esperadas por el método del servicio a ejecutar. Adicionalmente, este agente recibe los datos necesarios para satisfacer la demanda de entradas por parte del servicio. En el caso en que el agente no disponga de todos los datos necesarios, se solicitará al usuario final –a través de su correspondiente '*Cutomer Agent*'– tales datos. Una vez el agente dispone de esta información, adapta el rol '*Invoker*' para realizar la llamada sobre el servicio.

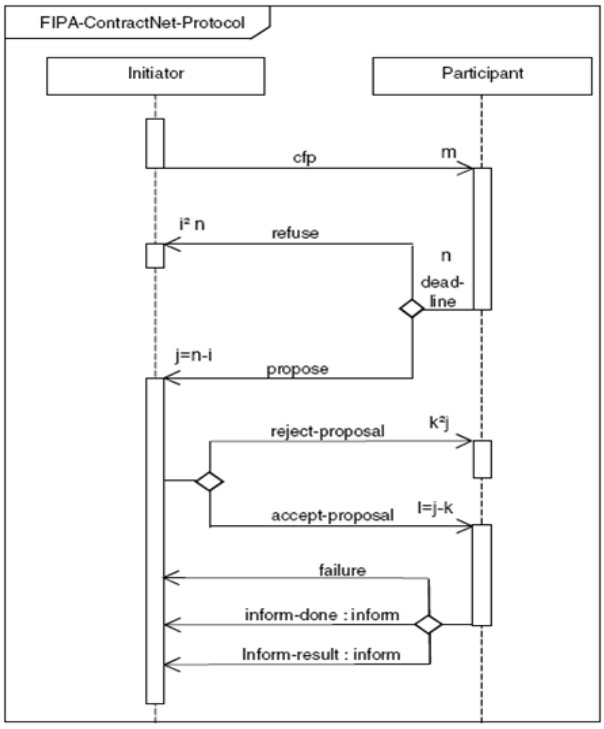


Fig. 89. Protocolo de interacción *Contract Net*[21]

[21] Extraído de http://www.fipa.org/specs/fipa00029/SC00029H.html

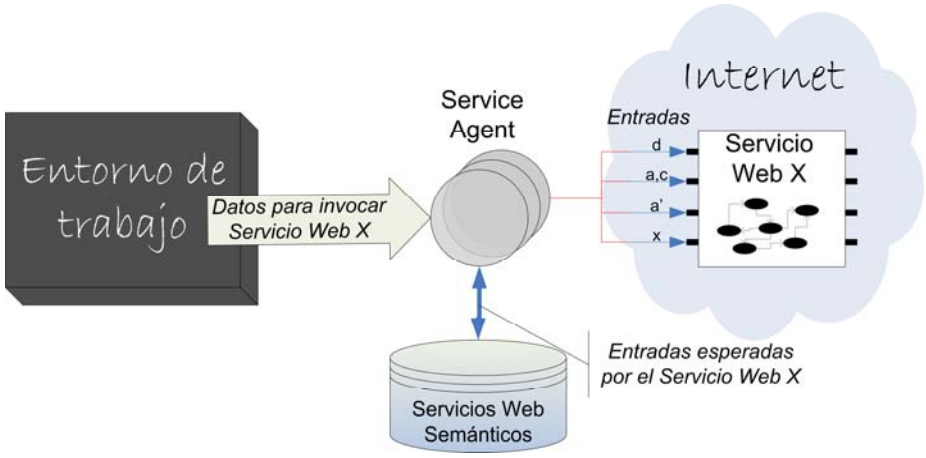


Fig. 90. Mediación de procesos en el entorno de trabajo

III.4.3.4. Mediación de funcionalidad

La mediación de funcionalidad surgió con el propósito de contrarrestar los problemas relacionados con la heterogeneidad a nivel funcional (Stollberg et al., 2006). Los problemas de interoperabilidad funcional surgen cuando la funcionalidad proporcionada por un servicio no se corresponde por completo con la solicitada por el cliente. En la Fig. 81(pag. 174) se representan los cinco tipos posibles de relaciones entre lo que un cliente requiere (esto es, su objetivo) y lo que un servicio ofrece (esto es, su capacidad). En las secciones III.4.2.1 y III.4.2.2, se describe en detalle el mecanismo empleado por el marco de trabajo para la mediación de funcionalidad. Éste se basa en la complementación de las tareas de descubrimiento y composición de servicios web.

III.4.4. Usuarios Externos

El último aspecto a destacar de la arquitectura del marco de trabajo presentado (ver Fig. 75, pag. 155), es la relación que ésta mantiene con las entidades externas a la plataforma. Se contemplan tres interfaces diferentes para tres tipos de usuarios externos:

1. *Usuarios consumidores de servicios*: los consumidores de servicios son aquellos usuarios individuales, colectivos (esto es, en representación de compañías) u otras entidades software que deseen llevar a cabo una determinada tarea por medio de la ejecución de servicios a través de la plataforma.
2. *Usuarios proveedores de servicios*: son aquellas entidades cuyo proceso de negocio les permite resolver determinadas tareas. Los proveedores de servicios

crean componentes software que ofrecen esta funcionalidad (esto es, servicios web) y los hacen accesibles globalmente.

3. *Desarrolladores*: en el ámbito del entorno de trabajo descrito, los desarrolladores son los usuarios que crean aplicaciones específicas sobre dominios particulares para resolver unos problemas determinados en esos dominios. Los desarrolladores se encargan, por tanto, de instanciar las entidades del entorno que consideran necesarias para satisfacer sus necesidades e introducen los datos concretos relativos a los elementos que parametrizan el entorno de trabajo. De esta forma, consiguen particularizar el entorno sobre un problema concreto.

Los perfiles de los usuarios de cada tipo son claramente diferentes. Atendiendo a estos perfiles, se han diseñado interfaces personalizadas que permiten a los usuarios externos interactuar con la plataforma. A continuación, se describen las características principales de cada una de estas interfaces

III.4.4.1. Desarrolladores

Los desarrolladores son los responsables de elaborar aplicaciones a partir del entorno de trabajo proporcionado. Para la aplicación del entorno de trabajo sobre un dominio concreto, el desarrollador dispone de una interfaz mediante la cual es posible realizar las siguientes funciones:

- *Añadir/Eliminar repositorios de ontologías*: el desarrollador se debe de encargar de indicar la localización de las bases de conocimiento a las que debe de acceder la plataforma para su correcto funcionamiento. En particular, es imprescindible que el desarrollador establezca la referencia al repositorio o repositorios donde se encuentran almacenadas las descripciones semánticas de los servicios, que proporcionan, en último término, la funcionalidad para la que la plataforma ha sido desarrollada. Será a estos repositorios donde la plataforma acudirá cuando se encuentre en la fase de descubrimiento de servicios. Además, las descripciones semánticas de los servicios, que los proveedores de servicios podrán registrar desde la misma plataforma, serán almacenados en estos repositorios.

- *Añadir/Eliminar ontologías*: uno de los elementos principales que permiten al desarrollador restringir el entorno de trabajo a una aplicación y un dominio concreto es por medio de la inserción de las ontologías específicas que particularicen dicha aplicación y dominio. Entre las ontologías que se espera que el desarrollador incluya en la plataforma, se encuentran las siguientes:
    - Ontología del dominio: conceptualización del dominio donde se pretende aplicar la plataforma. Contiene las entidades de conocimiento (conceptos, relaciones, propiedades, axiomas, etc.) que representan todo aquello que se considera relevante del universo del discurso.
    - Ontología de la aplicación: contiene el conocimiento necesario para modelar las características concretas de la aplicación que se desea ofrecer a partir del entorno de trabajo. En esta ontología, se incluye el conocimiento de las tareas que se deben de llevar a cabo para generar la aplicación prevista. Mediante esta ontología, los agentes que forman parte de la plataforma se pueden comunicar sin que haya interpretaciones erróneas.
    - Ontología de negociación: como ya se detalló previamente, la ontología de negociación incluye información relativa a los distintos mecanismos de negociación que pueden ser empleados por los agentes de la plataforma para realizar sus procesos de comunicación. Esta ontología incluye conocimiento relativo tanto a estrategias de negociación como a protocolos de negociación. El desarrollador es el encargado de incluir aquellas estrategias y protocolos que considere más apropiadas para la aplicación a desarrollar y el dominio.
    - Ontología de reglas de mapeo: para poder llevar a cabo las tareas de mediación que resuelvan los problemas de interoperabilidad, es imprescindible la existencia de reglas de mapeo que relacionen los elementos de conocimiento de las distintas ontologías involucradas en el sistema. Actualmente, aunque existen diversos trabajos de investigación en este sentido, la generación automática de estas reglas de mapeo no es posible. Así, es necesario que el desarrollador incluya una ontología con

este conocimiento, de forma que el sistema pueda manejar la heterogeneidad del entorno.

- *Implementación de roles*: el uso de roles es lo que provee al marco de trabajo de mayor flexibilidad y adaptabilidad. Tal y como se describió con anterioridad, el comportamiento de los agentes de la plataforma viene determinado por los roles que estos asuman. De este modo, se enumeraron los diversos roles clasificados en dos grupos, a saber, los que participan en las tareas de gestión de la plataforma, y los que proporcionan funcionalidad relacionada con el manejo de los servicios web (semánticos). Un rol identifica un comportamiento o funcionalidad que, en última instancia, debe ser implementado de acuerdo con un algoritmo concreto, y utilizando unas determinadas herramientas. Sin embargo, el marco de trabajo se ha diseñado de forma que se pueda disponer de diversas implementaciones para cada uno de los roles. Es tarea del desarrollador implementar, de un modo u otro, la funcionalidad prevista en cada uno de los roles. En realidad, se proporciona una implementación básica inicial para cada rol, y el desarrollador debe determinar si quiere incluir una implementación más sofisticada o no. De este modo, se pueden incorporar a la plataforma los algoritmos más eficientes para, entre otras, la ejecución de las tareas de descubrimiento, selección y composición de Servicios Web. Además, este mecanismo también resuelve el problema asociado a la existencia de múltiples especificaciones para la descripción semántica de servicios. En particular, pueden coexistir en el mismo entorno varias implementaciones de un mismo tipo de rol, teniendo, de este modo, la posibilidad de abarcar diversas aproximaciones al mismo tiempo. Para que esto sea una realidad, también es imprescindible la aportación de la función de la interfaz de desarrolladores para la instanciación de agentes que se describe a continuación.
- *Instanciación de agentes*: si bien los agentes de "usuario" (esto es, aquellos que representan a entidades externas, a saber, '*Customer Agent*', '*Provider Agent*' y '*Service Agent*') se crean conforme éstos se van necesitando (p.ej. cuando un nuevo cliente ha accedido al sistema), los agentes que constituyen el núcleo central de la plataforma (esto es, '*Broker Agent*', '*Discovery Agent*', '*Framework Agent*' y '*Selection Agent*'), deben ser instanciados al inicio. Es tarea del

desarrollador determinar cuántos agentes de estos tipos debe instanciar y qué roles deben asumir. Esto es lo que, en definitiva, establece la funcionalidad de la que es capaz la plataforma. En particular, y con respecto a lo que se mencionaba anteriormente en lo concerniente a la existencia de múltiples especificaciones para la descripción semántica de las capacidades de los servicios, es posible instanciar numerosos agentes de los tipos '*Discovery Agent*' y '*Selection Agent*' que, dando por supuesto que hacen uso de distintas implementaciones de los mismos roles, sean capaces de procesar descripciones semánticas de servicios que cumplan una u otra especificación.

En resumen, los pasos que todo desarrollador debe seguir para ajustar el marco de trabajo a las condiciones de una aplicación particular sobre un dominio concreto, son los siguientes: (1) incluir la localización de los repositorios de ontologías y, en particular, aquel o aquellos donde se encuentren las descripciones semánticas de las capacidades de los servicios web; (2) añadir las ontologías que particularizan tanto el dominio como la aplicación donde se va a emplear la plataforma, a saber, ontología del dominio, ontología de la aplicación, ontología de negociación y ontología de reglas de mapeo; (3) implementar, en caso necesario, los roles de acuerdo con las características propias del entorno donde se precise ejecutar la plataforma; e (4) instanciar los agentes que conforman el núcleo de la plataforma (esto es, '*Broker Agent*', '*Discovery Agent*', '*Framework Agent*' y '*Selection Agent*') indicando, para cada instancia de agente, los roles que ésta ha de asumir en la plataforma.

III.4.4.2. Proveedores de servicios

En el marco de trabajo propuesto en esta tesis doctoral, los Servicios Web son los componentes encargados de proporcionar los niveles más básicos de funcionalidad. Los agentes, por su parte, explotan esta funcionalidad aportando un valor añadido a lo ofrecido por los servicios. Las entidades responsables de poner en disposición del resto de usuarios los servicios web son los proveedores de servicios. Éstos no tienen la obligación de registrarse en la plataforma y hacer uso de las opciones que ésta les ofrece para que sus servicios puedan estar disponibles para los clientes de la plataforma. Sin

embargo, registrarse en la plataforma podría suponer ventajas para los proveedores de servicios, en tanto en cuanto éstos pueden, entre otras cosas, añadir preferencias respecto a los servicios que proporcionan. A continuación, se enumeran las utilidades que el proveedor de servicios tiene disponibles a través de la interfaz una vez se registra en la plataforma:

- *Añadir/Modificar/Eliminar servicios*: la plataforma, haciendo una búsqueda en profundidad sobre todos los repositorios de servicios web semánticos a los que tiene acceso, muestra al proveedor los servicios que, en el momento actual, éste ha puesto a disposición de la plataforma (esto es, aquellos que se pueden ejecutar porque su descripción semántica se encuentra almacenada en alguno de los repositorios a los que tiene acceso la plataforma). El proveedor, entonces, puede modificar o eliminar los servicios a su conveniencia. La modificación consiste en variar la descripción semántica del servicio de forma que futuros accesos por parte de la plataforma recuperen la versión actualizada de la descripción. La eliminación, por el contrario, supone el borrado total de la descripción semántica del repositorio en que ésta se encuentre. Al quitar esta descripción de los repositorios a los que tiene acceso la plataforma, el servicio queda, automáticamente, fuera del ámbito del sistema. Por otro lado, añadir un nuevo servicio implica agregar la descripción semántica de un nuevo servicio a las ya disponibles. A tales efectos, se dispone de dos mecanismos para añadir servicios: (i) mediante un fichero que contenga la descripción semántica; o (ii) indicando la localización de un nuevo repositorio donde encontrar la descripción del servicio. En el primer caso, la información contenida en el fichero se almacena en el repositorio más apropiado de acuerdo con el lenguaje con el que el servicio haya sido anotado (esto es, OWL-S, WSMO, WSDL-S, SWSF, SAWSDL) –dato que será consecuentemente proporcionado por el proveedor. En el caso en que se indique la localización de un nuevo repositorio, la plataforma procederá del mismo modo cuando es el desarrollador el que ha indicado tal información, registrando dicho repositorio, de forma que todos los servicios cuyas descripciones se encuentren almacenadas en este repositorio queden disponibles para los clientes de la plataforma.

- *Indicar preferencias generales*: una de las ventajas más importantes que proporciona la plataforma a aquellos proveedores de servicios que se registren en la misma, es la de tener en cuenta las preferencias de éstos en relación a la ejecución de los servicios que proporcionan. En particular, se le permite al proveedor la inclusión de dos tipos de condiciones o preferencias: preferencias generales y preferencias específicas del servicio (descritas en el siguiente ítem de la lista). Las preferencias generales hacen referencia a aquellas condiciones o reglas que vienen marcadas por los objetivos estratégicos de la organización. Estas restricciones se tienen en cuenta para todos los servicios proporcionados por el correspondiente proveedor y se aplican en el momento en que se negocian las condiciones por las cuales uno de estos servicios vaya a ser ejecutado.
- *Indicar preferencias sobre los servicios*: además de las preferencias generales aplicables a todos los servicios, los proveedores tienen la posibilidad, a través de la interfaz, de establecer una serie de condiciones particulares o específicas para cada uno de los servicios que proporcionan. Al igual que las preferencias generales, las específicas sobre los servicios se utilizan en el instante en que tiene lugar la negociación para determinar si un servicio se ejecuta o no. El '*Service Agent*' es el encargado de componer una propuesta –en respuesta a la solicitud del '*Selection Agent*'– que indique las condiciones *sine qua non* que se deben satisfacer para que se lleve a cabo la ejecución del servicio. Para ellos, el '*Service Agent*' accede a la información relativa tanto a las preferencias generales como a las específicas. Estas descripciones de preferencias complementa aquellas que, como se describió en apartados anteriores, están representadas como propiedades no funcionales en la descripción semántica de los servicios y se aplica de forma conjunta a éstas en el instante de la negociación.

III.4.4.3. Consumidores de servicios

Los usuarios consumidores de servicios pueden acceder a toda la funcionalidad ofrecida por los servicios disponibles a través de la plataforma por medio de su interfaz. Para esto, se pone a disposición de los usuarios una serie de opciones que le ayudan en este proceso. A continuación, se enumeran las más importantes:

- *Registro y establecimiento de preferencias generales*: al igual que para los usuarios proveedores de servicios, los consumidores de servicios deben registrarse en la plataforma para sacar el máximo provecho a la funcionalidad ofrecida por la herramienta. Durante el proceso de registro en la plataforma, el usuario incluye, además de los datos generales –que le sirven al sistema para generar un perfil inicial del usuario–, un conjunto de preferencias generales. Posteriormente, estas preferencias son utilizadas por el agente '*Customer Agent*', que actúa en representación del usuario final en la plataforma, en los procesos de decisión que tienen lugar en las distintas fases de que consta el procesado de una consulta, principalmente en las etapas de descubrimiento y selección, y la posterior presentación de los resultados. Las preferencias generales son de utilidad para todas y cada una de las consultas que realice el usuario, habiendo iniciado sesión en la plataforma. Entre las preferencias generales, se puede incluir información relativa al método de pago preferido por parte del usuario, coste máximo asumible en la ejecución de un servicio, tiempo de ejecución máximo, modo de presentación de resultados preferido, etc. Pero el registro en el sistema supone aún más beneficios para los usuarios. Las operaciones que realiza un usuario registrado son almacenadas en una bitácora de modo que, posteriormente, pueden ser utilizadas por la plataforma con distintos propósitos. En concreto, accediendo a la información relativa a las operaciones llevadas a cabo por parte de un usuario entre distintas sesiones, es posible generar sistemas de recomendación (esto es, presentar al usuario la información que a éste le puede interesar), sistemas de ayuda a la decisión (esto es, proponer al usuario acciones a tomar de acuerdo con la información histórica) y sistemas de personalización (esto es, amoldar la información que se presenta a un usuario a sus preferencias).
- *Consulta y establecimiento de preferencias particulares*: la misión principal del marco de trabajo que aquí se presenta es la de llevar a cabo las tareas que el usuario requiere. Con este propósito, los usuarios deben de introducir, mediante una consulta, el objetivo que desean obtener. El sistema, entonces, procesa dicha consulta interpretando los deseos del usuario y genera una representación interna del objetivo. A partir de ese momento, comienza todo el proceso que debe dar

lugar a la ejecución de los servicios apropiados para llevar a cabo la tarea requerida. Para la elaboración de la consulta por parte del usuario, éste debe elegir de entre las diversas alternativas que se han diseñado: (i) consulta a partir de texto en lenguaje natural; (ii) adjuntar fichero con la ontología que representa el objetivo según el modelo diseñado (ver Fig. 82); (iii) indicar URL donde se puede encontrar la ontología del objetivo; y (iv) utilizar una herramienta de generación de textos a partir de ontologías (Bernstein & Kaufmann, 2006 ;Ontopath System[22]). De forma adicional, el usuario tiene la posibilidad de incluir, acompañando a la consulta, un conjunto de condiciones o restricciones sobre esa consulta (esto es, preferencias particulares de la consulta). Estas preferencias, que sólo son de utilidad para la consulta actual, son aplicadas en distintas fases del proceso interno de la plataforma. En particular, estas indicaciones auxiliares de los usuarios tienen especial relevancia durante las etapas de descubrimiento y selección de servicios, en las que los servicios encontrados se filtran con respecto a las condiciones suficientes y necesarias. Del mismo modo, en la etapa de invocación de servicios, se pueden necesitar datos, para la ejecución de un método concreto, no incluidos en el objetivo extraído de la consulta del usuario. En tal caso, el '*Customer Agent*' intenta recuperar dichos datos de las preferencias particulares de la consulta (y de las generales) y, en última instancia, solicita los datos al usuario final.

- *Selección y ejecución de servicios*: una vez introducida la consulta, se inicia un proceso de búsqueda y descubrimiento de servicios que satisfagan dicha consulta. Posteriormente, sobre los servicios encontrados, se realiza un proceso de selección que concluye con una ordenación parcial de los servicios, de acuerdo con un valor de utilidad calculado en base a las preferencias de los usuarios y las condiciones propuestas por los agentes '*Service Agent*' que representan a dichos servicios. Entonces, al usuario se le muestra la lista parcialmente ordenada de servicios que contiene los subobjetivos en que se ha descompuesto el objetivo general y, para cada sub-objetivo, la lista ordenada de servicios que satisfacen dicho sub-objetivo. Con esta información, el usuario elige, de entre los servicios asociados a cada sub-

[22] http://oogl.snu.ac.kr/desc/index.html

objetivo, el más apropiado (si la ordenación que propone el sistema de ayuda a la decisión es adecuada, el usuario elegirá el primer servicio de cada lista). Una vez seleccionados los servicios, la plataforma comienza el proceso que lleva a la ejecución de los mismos y a la posterior presentación de resultados al usuario. Si más de un servicio tuvo que ser ejecutado, los resultados se combinan y se muestran de forma integrada al usuario final.

También se ha provisto al sistema con una envoltura de servicio web que permite a sistemas software externos hacer uso de la funcionalidad ofrecida por la plataforma del mismo modo en que el usuario humano, consumidor de servicios, puede hacerlo (ver Fig. 91). Para esto, una aplicación se comunica con la interfaz de consumidores de servicios del marco de trabajo a través de mensajes SOAP, teniendo acceso a las mismas opciones que se le ofrecen a los usuarios humanos.

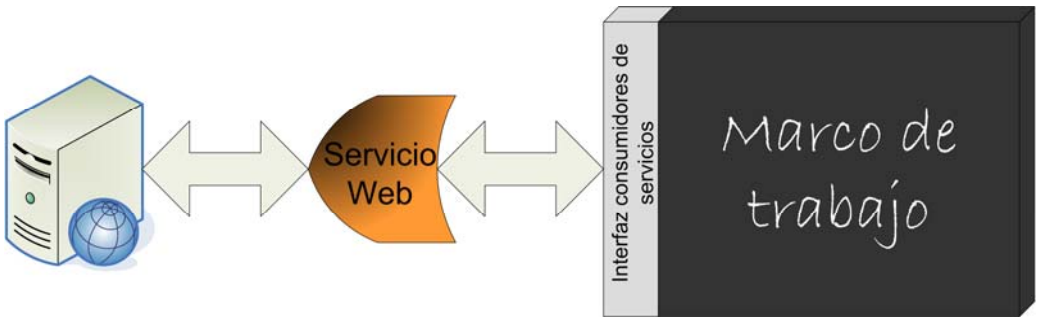


Fig. 91. Envoltura de Servicio Web que permite a sistemas informáticos actuar como consumidores de servicios

III.5. Resumen

En este capítulo, se ha presentado de forma detallada la arquitectura completa del marco de trabajo propósito de esta tesis doctoral. En primer lugar, se muestra un análisis exhaustivo de la información más relevante recogida en los capítulo I (estudio del estado del arte) y II (trabajo relacionado). Del capítulo I, se puede concluir que la próxima evolución de la Web pasa por hacer un uso combinado de las tecnologías de agentes y de Servicios Web (Semánticos). Esto viene justificado, en primer lugar, por el potencial que estas tecnologías suponen sobre la Web, en segundo lugar, por los

problemas asociados a las mismas cuando se aplican por separado y, finalmente, por la complementariedad de ambas, que conduce a una relación simbiótica donde se obtiene el máximo provecho de su potencial, y se restringen los problemas. En el capítulo II, se describen numerosas metodologías donde agentes inteligentes y Servicios Web (semánticos) se combinan en una misma plataforma. Estas soluciones poseen limitaciones que desaconsejan su utilización y, por ende, justifican la necesidad de desarrollar un sistema como el diseñado en esta tesis doctoral.

Una vez la finalidad de esta tesis doctoral ha sido debidamente justificada, se han especificado los objetivos que se persiguen, siendo el objetivo principal el desarrollo de un entorno de Servicios Web Semánticos basado en tecnología de agentes mediante la aplicación de técnicas de ingeniería del conocimiento. Como objetivos secundarios, se propone mantener al entorno de trabajo independiente del dominio y la aplicación, tener en cuenta propiedades como la usabilidad, la flexibilidad y la eficacia, y permitir ciertos parámetros de configuración para que el usuario pueda personalizar la herramienta.

A continuación, en base a estos objetivos, se presentan, de forma resumida, detalles relativos al diseño y la especificación del sistema de acuerdo con las fases y etapas que establece la metodología INGENIAS, dedicada al desarrollo de SMA. En la siguiente sección, se muestran casos de uso, modelos de organización, de agentes, de tareas y objetivos, de interacción, y de entorno, de forma que se puede apreciar una visión global del sistema elaborado.

Teniendo en cuenta los modelos y diagramas desarrollados en las fases de análisis y diseño, se muestra la arquitectura general del entorno de trabajo. De esta arquitectura, se describen los componentes principales que la constituyen, a saber, agentes –y los roles en que éstos se basan–, repositorios de datos y bases de conocimiento utilizadas para el desarrollo de tareas intensivas en conocimiento, y las interfaces externas que permiten la interacción con la plataforma por parte de desarrolladores, proveedores de servicios y consumidores de servicios. Además, se detallan las técnicas utilizadas para la gestión (descubrimiento, composición, selección, invocación y monitorización) de servicios web y la resolución de los problemas de interoperabilidad causados por la heterogeneidad inherente a Internet a distintos niveles (datos, protocolos, procesos y funcionalidad).

La diferencia principal entre la solución aportada a través de esta tesis doctoral y otras aplicaciones desarrolladas previamente, radica, fundamentalmente, en la asunción de que, habiendo sido concebidas con propósitos bien diferenciados, las tecnologías de agentes y de Servicios Web (semánticos) deben permanecer en dos niveles de abstracción. Partiendo de esta hipótesis, se ha diseñado un entorno de trabajo que, a diferencia de los desarrollados hasta el momento, es capaz de explotar al máximo el potencial de sus tecnologías constituyentes (esto es, tecnología de agentes y de Servicios Web). Para esto, se propone una solución no intrusiva basada en el uso intensivo de ontologías como mecanismo de representación de conocimiento, que permite una comunicación fluida y eficaz entre las distintas entidades que componen el entorno. De esta forma, se consigue una complementariedad que, por un lado, conduce a la acotación de los problemas y dificultades de las tecnologías mencionadas, y, por otro lado, permite la explotación de los aspectos positivos y beneficiosos de cada una.

CAPÍTULO IV. CASO DE USO – EJEMPLOS DEL SISTEMA

IV.1. Introducción

En este capítulo, se muestra el funcionamiento del marco de trabajo, descrito en el capítulo anterior, a través de una serie de ejemplos, que cubren un amplio rango de las posibilidades ofrecidas por el sistema. Todas las tareas de ejemplo, que se presentan a continuación, son representativas del dominio de comercio electrónico, en particular de la compra-venta de componentes de ordenador. El caso de uso supone que numerosos proveedores de componentes informáticos ponen a disposición de sus clientes los productos que ofertan a través de Internet, por medio de servicios web. Los clientes que desean comprar los productos informáticos de estos proveedores, tienen que interaccionar con estos servicios web y realizar las transacciones oportunas.

Se plantean un total de tres casos de ejemplo. En primer lugar, se examina en profundidad el comportamiento del marco de trabajo en las tareas de descubrimiento, selección e invocación de servicios, actividades básicas en todo entorno de Servicios Web. A continuación, se aplica el sistema a la resolución de un problema de composición de servicios, que surge como consecuencia de que un objetivo no puede ser satisfecho por un único servicio y son necesarios varios para cumplir con las necesidades del usuario. Por último, se muestra el funcionamiento del marco de trabajo en un entorno en el que es preciso tratar con fuentes de datos heterogéneas y proporcionar un acceso integrado a las mismas.

IV.2. Tarea de Ejemplo 1 – descubrimiento, selección e invocación de servicios

El propósito de esta sección es mostrar el comportamiento del marco de trabajo en relación a las tareas de descubrimiento, selección e invocación de servicios web. Para esto, se propone un escenario de ejemplo donde se precisa una herramienta capaz de

llevar a cabo con éxito estas tareas. Posteriormente, se describe cómo el marco de trabajo da respuesta a las dificultades que surgen en el escenario propuesto. Finalmente, se compara la utilidad y el proceder del marco de trabajo con respecto a la de otros entornos de ejecución de Servicios Web Semánticos.

IV.2.1. Escenario

En el escenario que se plantea para esta tarea de ejemplo (ver Fig. 92), existen dos actores principales, el usuario que pretende adquirir un componente informático y las compañías que venden componentes de ordenador. Tal y como se puede observar en la Fig. 92, se consideran tres proveedores de productos informáticos: "*Proveedor A*", "*Proveedor B*" y "*Proveedor C*". Existe, para cada proveedor, un servicio web que permite acceder a las ofertas y productos de dicho proveedor: "*Servicio A*", "*Servicio B*" y "*Servicio C*". Además, las descripciones semánticas de estos servicios, a saber, "*Descripción A*", "*Descripción B*" y "*Descripción C*", se encuentran almacenadas en un repositorio, "*Repositorio X*", que se encuentra disponible a través de Internet.

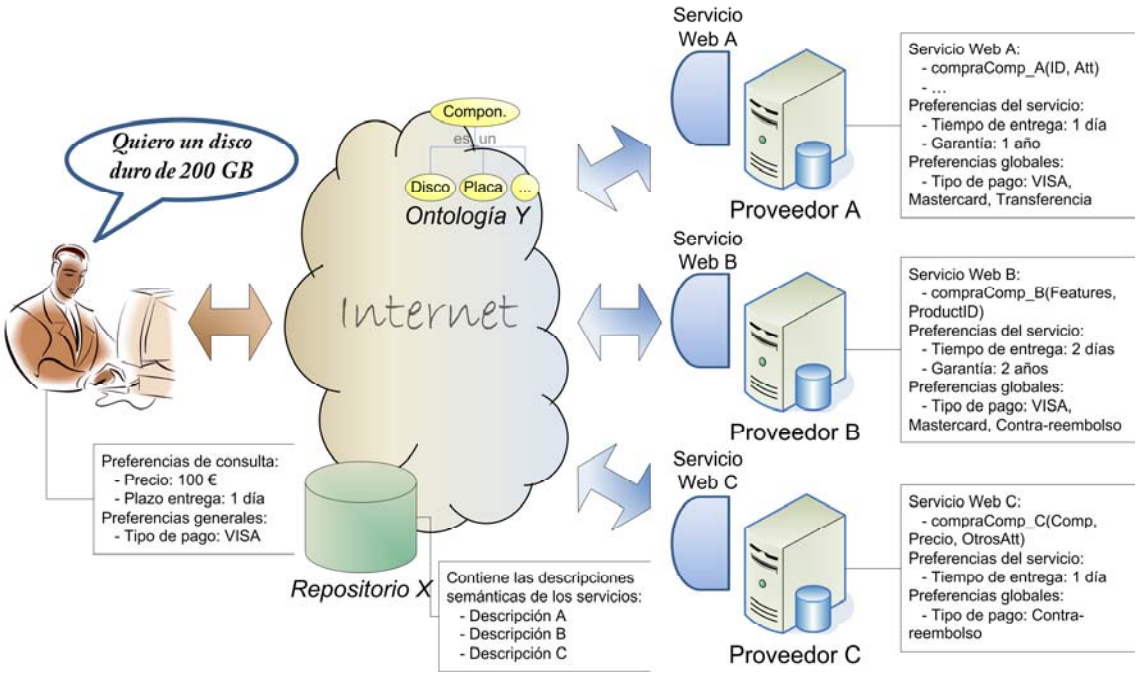


Fig. 92. Escenario para la tarea de ejemplo 1

El usuario final, que en este escenario juega el rol de consumidor de servicios, tiene la necesidad de adquirir un disco duro con unas determinadas características: la capacidad

debe de ser de 200 Gigabytes, el precio debe rondar los 100€ y el plazo de entrega deseado es de 1 día. Además, en todas sus compras, el usuario quiere hacer uso de su tarjeta de crédito VISA. Opcionalmente, es posible que la entidad que juega el papel de consumidor de servicios sea otro componente software de cualquier índole (agente software, Servicio Web, etc.).

Por su parte, cada uno de los proveedores dispone también de una serie de preferencias que es necesario tener en cuenta para poder ejecutar los servicios que proporcionan. El "*Proveedor A*" indica que, en las transacciones correspondientes al "*Servicio Web A*", puede ofrecer una garantía de 1 año, y asegura un tiempo máximo de entrega de 1 día. Además y ya no sólo para el "*Servicio Web A*" sino para todos los servicios ofertados, el "*Proveedor A*" acepta, como métodos de pago, transferencias bancarias y tarjetas de crédito VISA y Mastercard. De forma similar, el "*Proveedor B*" establece, para el "*Servicio Web B*", que el plazo de entrega es de 2 días y la garantía de 2 años, y, para todos los servicios que proporciona, que se acepta el pago a través de tarjetas VISA y Mastercard, y contra-reembolso. Por último, las condiciones propuestas por el "*Proveedor C*" determinan que las tareas asociadas al "*Servicio Web C*" tienen un plazo de entrega de 1 día, y el único tipo de pago aceptado, por todos los servicios que proporciona este proveedor, es contra-reembolso.

Finalmente, con respecto a los servicios y sus descripciones semánticas, se ha optado por una visión bastante simplista del escenario. Así, todos los servicios tienen un método similar que permite la compra de componentes informáticos indicando el tipo y las características del mismo ("*compraComp_A(…)*", "*compraComp_B(…)*" y "*compraComp_C(…)*"). Además, se asume que la descripción semántica de las capacidades de los servicios se ha realizado utilizando una única ontología ("*Ontología Y*"), disponible a través de Internet, de forma que se eluden los problemas asociados a la heterogeneidad.

IV.2.2. Aplicación del marco de trabajo

Los requisitos funcionales d el marco de trabajo para satisfacer las necesidades del escenario presentado en la Fig. 92, son los siguientes:

1. La plataforma tiene que proporcionar una interfaz de usuario para la interacción con los clientes, tanto para usuarios humanos como para entidades software.
2. La plataforma tiene que descubrir los Servicios Web apropiados para cada petición del usuario.
3. La plataforma tiene que seleccionar el Servicio Web más conveniente y ordenar la lista de servicios de acuerdo con las preferencias tanto de consumidores como de proveedores.
4. La plataforma tiene que invocar Servicios Web y proporcionar un entorno de ejecución completo de Servicios Web con funciones de control, manejo de excepciones, y soporte a interacciones opcionales con los usuarios.
5. La plataforma tiene que proporcionar interfaces para la interacción con las entidades proveedoras de servicios.

El marco de trabajo descrito en el capítulo III cumple todos y cada uno de estos requisitos funcionales. En primer lugar, el '*Customer Agent*', en combinación con una aplicación Web, se encarga de proporcionar la interfaz más conveniente según el perfil del usuario. Este agente, además, es capaz de interpretar la consulta de los usuarios para, a partir de ésta, determinar el objetivo que el sistema tiene que perseguir. En cuanto a la interacción con otros componentes software, el marco de trabajo incluye, en la interfaz con los consumidores de servicios, un Servicio Web (ver Fig. 91) que permite a otras entidades computacionales interaccionar con la plataforma, de igual forma en que un usuario humano lo hace. Así, suponiendo que el marco de trabajo hace uso de la "*Ontología Y*" de componentes de ordenador, el objetivo que representa las necesidades del usuario se muestra en la Fig. 93 (conforme a la estructura de los objetivos en el marco de trabajo presentada en la Fig. 82).

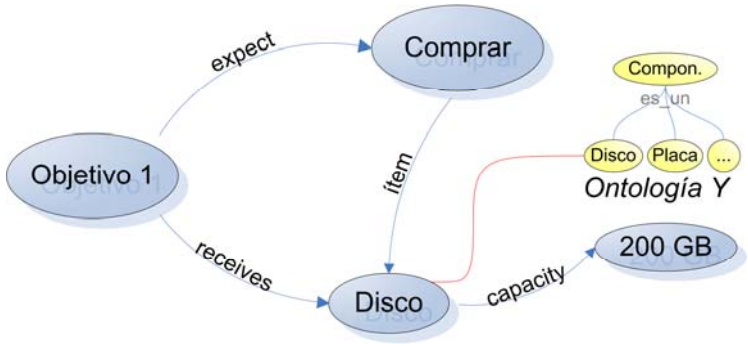


Fig. 93. Ontología del objetivo para tarea de ejemplo 1

Una vez que el '*Customer Agent*' conoce la necesidad del usuario, el flujo de trabajo se transfiere al '*Discovery Agent*', responsable de descubrir los servicios disponibles a través de Internet que puedan satisfacer la petición del usuario. Con este propósito, el agente en cuestión realiza una consulta sobre el "*Repositorio X*", que culmina con la devolución de todos los servicios que satisfacen las condiciones impuestas. En este caso de ejemplo, la respuesta del "*Repositorio X*" ante la consulta del '*Discovery Agent*' es la siguiente lista de descripciones de servicios: {"*Descripción A*", "*Descripción B*", "*Descripción C*"}, que se corresponden con los servicios "*Servicio Web A*" ('S1'), "*Servicio Web B*" ('S2') y "*Servicio Web C*" ('S3'), respectivamente, satisfaciendo, así, el segundo requisito funcional que se ha establecido. El '*Discovery Agent*', entonces, envía esta lista de descripciones de servicios al '*Customer Agent*', que comienza la siguiente fase en el procesamiento de la consulta del usuario, a saber, la ordenación de los servicios de acuerdo con su valor de utilidad.

La etapa de ordenación y selección comienza cuando el '*Customer Agent*' envía la lista de descripciones de servicios al '*Selection Agent*'. Las tareas que lleva a cabo este agente son las siguientes: (1) negociar con los agentes '*Service Agent*' de los servicios descubiertos; y (2) ordenar los servicios de acuerdo con un valor de utilidad calculado en base a las preferencias del usuario y a las condiciones para la ejecución de los servicios. Para la primera tarea, siguiendo el ejemplo, el '*Selection Agent*' tiene que entablar un proceso de negociación con tres agentes '*Service Agent*': 'sea1'

(representante de 'S1'), 'sea2' (representante de 'S2') y 'sea3' (representante de 'S3')[23]. Cada uno de estos agentes, a su vez, solicita información relativa al proveedor de los servicios, a sus respectivos agentes representantes: 'p1' (que representa al "*Proveedor A*", 'p2' (que representa al "*Proveedor B*") y 'p3' (que representa al "*Proveedor C*")[23]. De este modo, los agentes '*Service Agent*' en cuestión generan, teniendo en cuenta tanto las preferencias particulares del servicio como las generales de los proveedores, una propuesta que es, posteriormente, enviada al '*Selection Agent*'.

Cuando el '*Selection Agent*' recibe todas las propuestas por parte de los agentes '*Service Agent*' pertinentes, procede a la evaluación de cada una de éstas[24]. Para esto, se comparan las preferencias del usuario con las condiciones convenidas por los representantes de los servicios, y se calcula un valor de utilidad para cada propuesta. El marco de trabajo propuesto no fija ninguna solución específica para el cálculo del valor de utilidad, de forma que es tarea del desarrollador indicar la que crea conveniente. Para la tarea del ejemplo en cuestión, se plantea un valor de utilidad que mide el grado de satisfacción de las preferencias del usuario, valorando las que se refieren a la consulta actual más que a las generales[25]. De este modo, la ordenación de los servicios o, para ser precisos, de las descripciones semánticas de los servicios para el escenario planteado en la Fig. 92 queda como sigue: {"*Descripción A*", "*Descripción C*", "*Descripción B*"}. Para esta ordenación, se ha valorado que 'S1' satisface tanto las preferencias particulares como las generales, 'S3' sólo satisface las particulares (ya que no se acepta el pago con la tarjeta VISA) y, finalmente, 'S2' sólo cumple las preferencias generales (dado que el plazo de entrega es superior al indicado por el usuario). También hubiera sido posible aplicar el criterio de descartar todos los servicios que incumplan algunas de las condiciones establecidas por el usuario, ya sean particulares de la consulta actual, o generales, pero esto cae en el ámbito de decisión del desarrollador.

[23] Se omite el proceso de instanciación de tales agentes, asumiendo que éstos han sido creados, por parte del '*Framework Agent*' en interacciones anteriores.

[24] Por simplicidad, se asume que el mecanismo de negociación utilizado no permite el envío de contra-ofertas.

[25] Por simplicidad, se omite la valoración de las propiedades no funcionales de las descripciones semánticas de los servicios.

La lista ordenada de servicios se le envía, a continuación, al '*Customer Agent*', que es el encargado de mostrar la lista al cliente para que éste proceda a elegir el servicio que más le convenga. Y con esto se completa el tercer requisito funcional. En la Fig. 94, se muestra un diagrama de colaboración que representa todos los pasos desde la consulta del usuario a la presentación de los servicios ordenados.

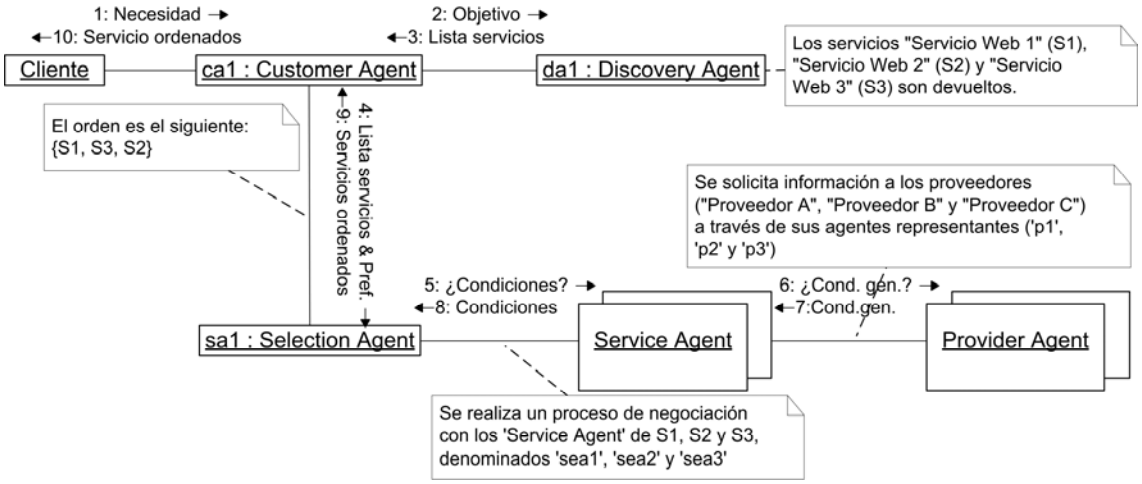


Fig. 94. Diagrama de colaboración para la tarea de ejemplo 1 – fases de descubrimiento y selección

El usuario final o cliente debe, entonces, evaluar cada uno de los servicios, pudiendo seleccionar el servicio sugerido por el '*Selection Agent*' o cualquier otro de la lista. Una vez hecho esto, comienza la etapa de invocación del servicio seleccionado. Supongamos que el cliente está conforme con la ordenación propuesta por el sistema y selecciona, para su ejecución, el servicio 'S1'. Esta elección le llega al '*Customer Agent*', que delega en el agente 'sea1' (la instancia de '*Service Agent*' que representa a 'S1'), la tarea de invocar el método apropiado del servicio. Accediendo a la información relativa del servicio seleccionado, el objetivo de partida, y las preferencias del cliente, el agente 'sea1' compone la información necesaria para realizar la ejecución del método correspondiente. En caso de que alguno de los parámetros necesarios para ejecutar el servicio no se encuentre entre la información recogida al inicio (cosa que no ocurre en el escenario de ejemplo), el cliente es informado y tiene la posibilidad de adjuntar los datos faltantes para que se pueda llevar a cabo, con éxito, la ejecución del servicio. Una vez que el agente 'sea1' ha ejecutado el método y recuperado los resultados, éstos le son

remitidos al cliente a través del '*Customer Agent*'. Todo este proceso se detalla en el diagrama de colaboración que se muestra en la Fig. 95.

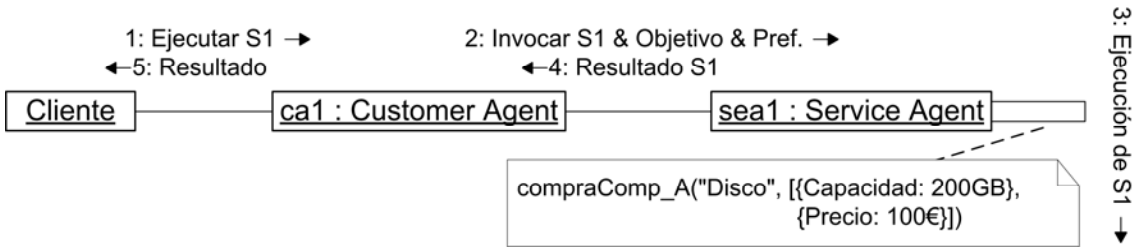


Fig. 95. Diagrama de colaboración para la tarea de ejemplo 1 – fase de invocación

Pero, para disponer de un entorno de ejecución de Servicios Web Semánticos completo de acuerdo con los requisitos funcionales, es necesario un mecanismo para el control de errores y el manejo de excepciones. Son cuatro los agentes del marco de trabajo involucrados en estas tareas, a través de los roles que éstos asumen a lo largo de su ciclo de vida, a saber, '*Framework Agent*', '*Provider Agent*', '*Service Agent*' y '*Customer Agent*'. El 'Framework Agent' es el encargado de monitorizar todo lo que tiene lugar en la plataforma, evitar bloqueos, y tomar las acciones necesarias en caso de producirse un error al que no se puedan enfrentar las otras entidades. '*Provider Agent*' y '*Service Agent*' controlan el proceso de ejecución desde el punto de vista del proveedor. Mientras el representante del proveedor se asegura de que las preferencias globales de éste son respetadas, el representante del servicio cumple una doble misión: responder ante los errores que se puedan producir durante la ejecución del servicio y garantizar que las preferencias de los proveedores, con respecto a este servicio en particular, son satisfechas. Así, se complementa lo resumido en la Fig. 95 y se cumple íntegramente lo dispuesto por el cuarto requisito funcional.

Los agentes '*Provider Agent*' y '*Service Agent*' se encargan de satisfacer el quinto y último requisito funcional de la lista presente en el comienzo de esta sección. Éstos, a través de la interfaz Web, proporcionada por el marco de trabajo, gestionan el proceso de interacción con las entidades proveedoras de servicios. La interfaz con los proveedores, descrita en la sección III.4.4.2, le permite a éstos realizar diversas operaciones sobre el marco de trabajo. En particular, se destacan tres operaciones

básicas: (1) Añadir/Modificar/Eliminar servicios; (2) Establecer las preferencias generales; (3) Establecer preferencias particulares para cada uno de estos servicios.

IV.2.3. Diferencias con otras plataformas

Un usuario humano que descartase utilizar un entorno de ejecución de Servicios Web Semánticos no podría aprovechar los numerosos beneficios que conlleva la anotación semántica de las capacidades de los Servicios Web. En ese hipotético caso, el usuario tendría que realizar, manualmente, la búsqueda de los servicios disponibles a través de Internet, estudiar la funcionalidad de cada uno de ellos y valorar si satisfacen o no sus necesidades. Entonces, también de forma manual, el usuario tendría que seleccionar, de entre los servicios encontrados que cumplen sus deseos, aquel que mejor se ajusta a sus pretensiones y, posteriormente, invocarlo dando valor a los parámetros de entrada después de interpretar su significado. Finalmente, el usuario recuperaría los resultados de la invocación y, nuevamente, tendría que interpretarlos para conocer su contenido.

Del mismo modo, la utilización del marco de trabajo presentado en esta tesis doctoral supone numerosas ventajas en cuanto a la experiencia del usuario frente al uso de alguno de los entornos de ejecución de Servicios Web Semánticos desarrollados hasta la fecha y que fueron descritos en el capítulo II. Se pueden destacar las siguientes diferencias:

- El marco de trabajo elaborado permanece desvinculado de la tecnología de Servicios Web Semánticos que utilicen para anotar semánticamente los servicios, lo cual supone un soporte tanto para las especificaciones actuales como WSMO, OWL-S, SWSF, WSDL-S y SAWSDL, como para las que puedan surgir en el futuro, que pueden ser dinámicamente incorporados al marco de trabajo.
- El marco de trabajo descrito permite realizar procesos de negociación que tengan en cuenta tanto las necesidades y exigencias de los proveedores como las condiciones y preferencias de los consumidores.
- El marco de trabajo propuesto en esta tesis doctoral hace uso de ontologías durante todo el ciclo de vida del procesamiento de consultas, desde el análisis de los objetivos del consumidor de servicios hasta la invocación y recuperación

de resultados, y posterior presentación de los mismos al usuario final. Este enfoque cumple una doble misión: por un lado, la comunicación entre las distintas entidades que conforman el sistema se realiza sin peligro de que se produzcan interpretaciones erróneas y, por otro, con esto se consigue mantener la indispensable independencia entre las tecnologías de agentes y de Servicios Web, mientras se produce una comunicación fluida y eficaz entre agentes y servicios.

IV.3. Tarea de Ejemplo 2 – composición de servicios

El propósito de esta sección es mostrar el comportamiento del marco de trabajo en relación a la tarea de composición de servicios web y su relación con el descubrimiento. Para esto, se propone un escenario de ejemplo donde se precisa una herramienta capaz de llevar a cabo con éxito esta tarea. Posteriormente, se describe cómo el marco de trabajo da respuesta a las dificultades que surgen en el escenario propuesto. Finalmente, se compara la utilidad y el proceder del marco de trabajo con respecto a la de otros entornos de ejecución de Servicios Web Semánticos.

IV.3.1. Escenario

Los actores que participan en el escenario para esta tarea de ejemplo son los mismos que para el ejemplo anterior, a saber, un usuario que desea realizar una compra de componentes informáticos y que actúa como consumidor de servicios, y diversas compañías minoristas o mayoristas de componentes de ordenador, que juegan el rol proveedores de servicios. Al igual que para el ejemplo anterior, el escenario, presentado en la Fig. 96, está compuesto por tres proveedores de servicios ("*Proveedor A*", "*Proveedor B*" y "*Proveedor C*"), que ofrecen y publicitan sus productos en Internet por medio de Servicios Web ("*Servicio A*", "*Servicio B*" y "*Servicio C*"). La descripción semántica de estos servicios ("*Descripción A*", "*Descripción B*" y "*Descripción C*") se encuentra almacenada en una base de conocimiento ("*Repositorio X*"), a la que se puede acceder a través de Internet. Además, para la elaboración de estas descripciones semánticas, los tres proveedores han utilizado una misma ontología ("*Ontología Y*"),

disponible en Internet. De esta forma, se evitan los inconvenientes relacionados con la interoperabilidad de los sistemas y la heterogeneidad de las fuentes de datos.

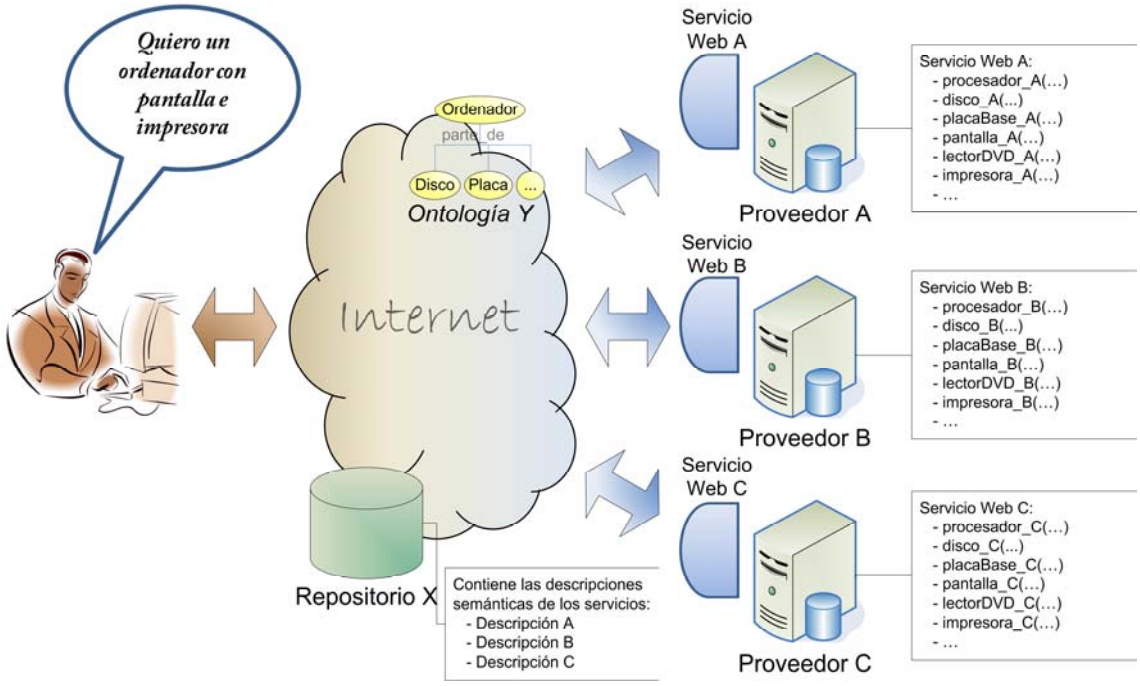


Fig. 96. Escenario para la tarea de ejemplo 2

El usuario final presente en el escenario mostrado, tiene la necesidad de hacer la compra de un ordenador personal que incluya pantalla e impresora, y así lo indica a través de su consulta. Con el propósito de simplificar el ejemplo, no se considera la introducción de ninguna condición especial ni preferencia por parte del usuario en su búsqueda de estos elementos informáticos. Opcionalmente, es posible que la entidad que juega el papel de consumidor de servicios sea otro componente software de cualquier índole (agente software, Servicio Web, etc.).

Como se destacó anteriormente, el escenario presenta tres proveedores de componentes de ordenadores diferentes. Cada uno de estos proveedores, tiene a su disposición un Servicio Web con distintos métodos. Así, el "*Servicio A*" del "*Proveedor A*" está compuesto de los métodos "*procesador_A(…)*", "*disco_A(…)*", "*placaBase_A(…)*", etc. De forma similar, los métodos del "*Servicio B*" del "*Proveedor B*" son "*procesador_B(…)*", "*disco_B(…)*", "*placaBase_B(…)*", etc., y los del "*Servicio C*" del "*Proveedor C*" son "*procesador_B(…)*", "*disco_B(…)*", "*placaBase_B(…)*", etc.

Por tanto, en este escenario simplificado los métodos de cada servicio son idénticos en cuanto a funcionalidad y existe un método por cada componente que constituye un ordenador. Además, se omite la inclusión, por parte del proveedor, de preferencias y condiciones para la ejecución de los servicios que éste proporciona.

Por último, con el objetivo de simplificar el problema, se ha hecho uso de una misma ontología ("*Ontología Y*") para realizar las descripciones semánticas de los distintos servicios. Esta ontología incluye, entre otras cosas, una partonomía que describe los componentes que constituyen un ordenador. De este modo, la salida de cada uno de los métodos de los Servicios Web se anota con uno de los conceptos de la partonomía, que denota su clasificación como parte constituyente de un ordenador. Esto facilita, a la postre, el descubrimiento de los métodos que es necesario invocar y componer para obtener todos los componentes que conforman un ordenador.

IV.3.2. Aplicación del marco de trabajo

Los requisitos funcionales del marco de trabajo para satisfacer las necesidades del escenario presentado en la Fig. 96 son los siguientes:

1. La plataforma tiene que proporcionar una interfaz de usuario para la interacción con los clientes, tanto para usuarios humanos como para entidades software.
2. La plataforma tiene que descubrir los Servicios Web apropiados para una petición de usuario, realizando la composición, si es necesario, de diversos servicios para alcanzar uno objetivo común.
3. La plataforma tiene que seleccionar los Servicios Web más apropiados.
4. La plataforma tiene que invocar Servicios Web de forma que una posible composición de éstos retorne el resultado esperado. Como complemento a esto, debe proporcionar un completo entorno de ejecución de Servicios Web con funciones de control, manejo de excepciones, y soporte a interacciones opcionales con los usuarios.
5. La plataforma tiene que proporcionar interfaces para la interacción con las entidades proveedoras de servicios.

Si se comparan los requisitos funcionales para este escenario de ejemplo con los enumerados en el ejemplo anterior (sección IV.3.1), se pueden destacar dos diferencias principales. En primer lugar, se constata que, durante la fase de descubrimiento, se puede precisar la composición de varios servicios para alcanzar el objetivo de inicio. Por otro lado, también se destacan las implicaciones que, de cumplirse lo anterior, habrían de considerarse durante las etapas de selección e invocación de los servicios. Por tanto, en esta sección se presta especial atención a demostrar cómo el uso del marco de trabajo propuesto permite hacer frente a estos nuevos requisitos, obviando algunos detalles relativos a los requisitos funcionales ya justificados para el escenario del ejemplo previo.

El proceso que lleva a cabo el sistema para procesar la consulta del usuario es idéntico a lo expuesto con anterioridad. El '*Customer Agent*' recibe, a través de la interfaz Web, la consulta del usuario, la analiza, y genera una ontología que representa, en el formato utilizado internamente, el objetivo que el usuario ha introducido (ver Fig. 97). Para esto, el agente en cuestión hace uso de la ontología del dominio, en el ejemplo "Ontología Y", y la ontología de la aplicación, también disponible por parte del sistema y que, en el caso de ejemplo, está constituida por el vocabulario concerniente a las tareas de compra/venta en entornos B2C ("*Business to Consumer*").

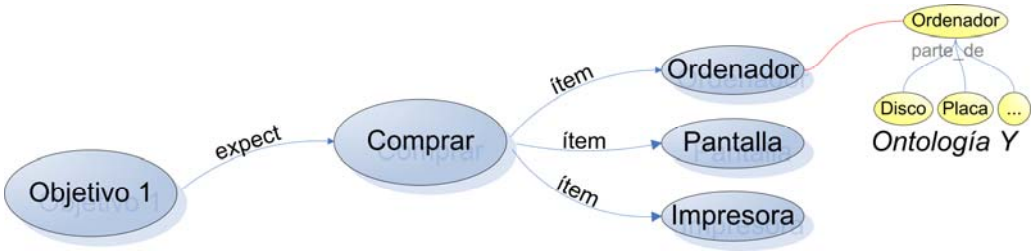


Fig. 97. Ontología del objetivo para tarea de ejemplo 2

El '*Discovery Agent*', entonces, recibe esta ontología con el objetivo y comienza la búsqueda de los servicios capaces de alcanzar dicho objetivo. Para esto, se realiza una consulta sobre el "*Repositorio X*", donde se encuentran almacenadas las descripciones semánticas de los servicios disponibles por parte del sistema. Dado que en el escenario de ejemplo ninguno de los servicios presentados incluye un método que permita la compra conjunta de un ordenador completo junto con la pantalla y la impresora, la

búsqueda concluye sin producir ningún resultado. Llegado a este punto, el '*Discovery Agent*' debe hacer uso del '*Composer Rol*', que lleva a cabo la división del objetivo original en numerosos subobjetivos conforme a algún mecanismo de descomposición. En la Fig. 98, se representa una posible descomposición del objetivo. En este caso, la necesidad de comprar un ordenador se ha dividido en relación a los componentes que lo constituyen, a saber, placa base, disco duro, procesador, etc., y a esto se le añade la necesidad de comprar una pantalla y la necesidad de comprar una impresora. El '*Discovery Agent*' realiza, a continuación, una búsqueda sobre el "*Repositorio X*" para cada subobjetivo, y genera una estructura de datos donde los subobjetivos y los servicios capaces de resolverlos quedan ligados. Para ser exactos, la información devuelta tras la consulta en el repositorio es relativa a alguno de los métodos que forman parte de los servicios disponibles. Así, y a modo de ejemplo, los métodos "*disco_A(…)*", "*disco_B(…)*" y "*disco_C(…)*" pertenecientes a los servicios "*Servicio Web A*", "*Servicio Web B*" y "*Servicio Web C*", respectivamente, serían el resultado de consultar el subobjetivo "*Comprar disco*"

Esta estructura llega al '*Customer Agent*', que delega en el '*Selection Agent*' la misión de valorar cada una de las posibilidades. A diferencia de lo que sucede en el ejemplo anterior, en este caso es necesario realizar la ordenación, por separado, de las listas de servicios asociadas a cada subobjetivo. De este modo, el '*Selection Agent*' evalúa, de forma individual, cada subobjetivo y realiza la ordenación parcial de cada lista de servicios. Finalmente, se presenta al usuario final la estructura de datos donde las listas de servicios han sido ordenadas en función de su valor de utilidad. En la Fig. 99, se muestra un diagrama de colaboración que representa todos los pasos desde la consulta del usuario hasta la presentación de los servicios para cada subobjetivo ordenados.

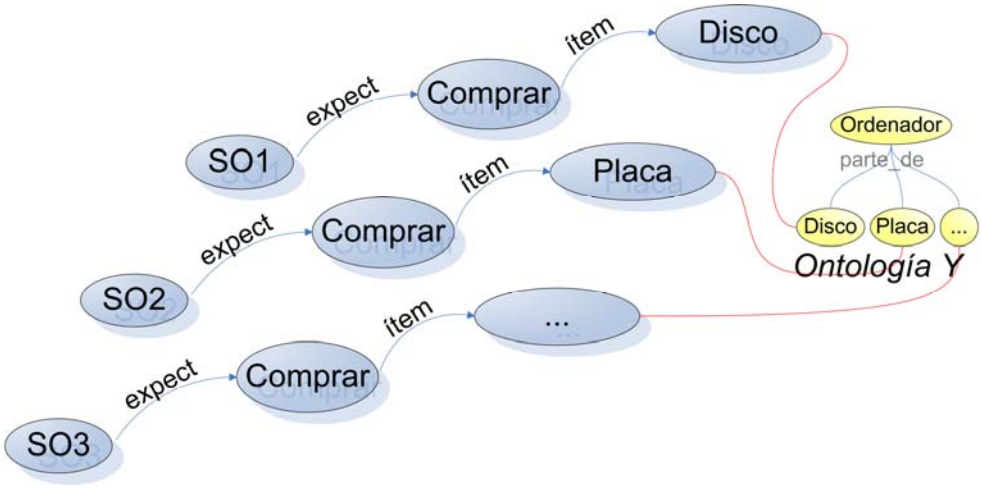


Fig. 98. Lista de subobjetivos para tarea de ejemplo 2

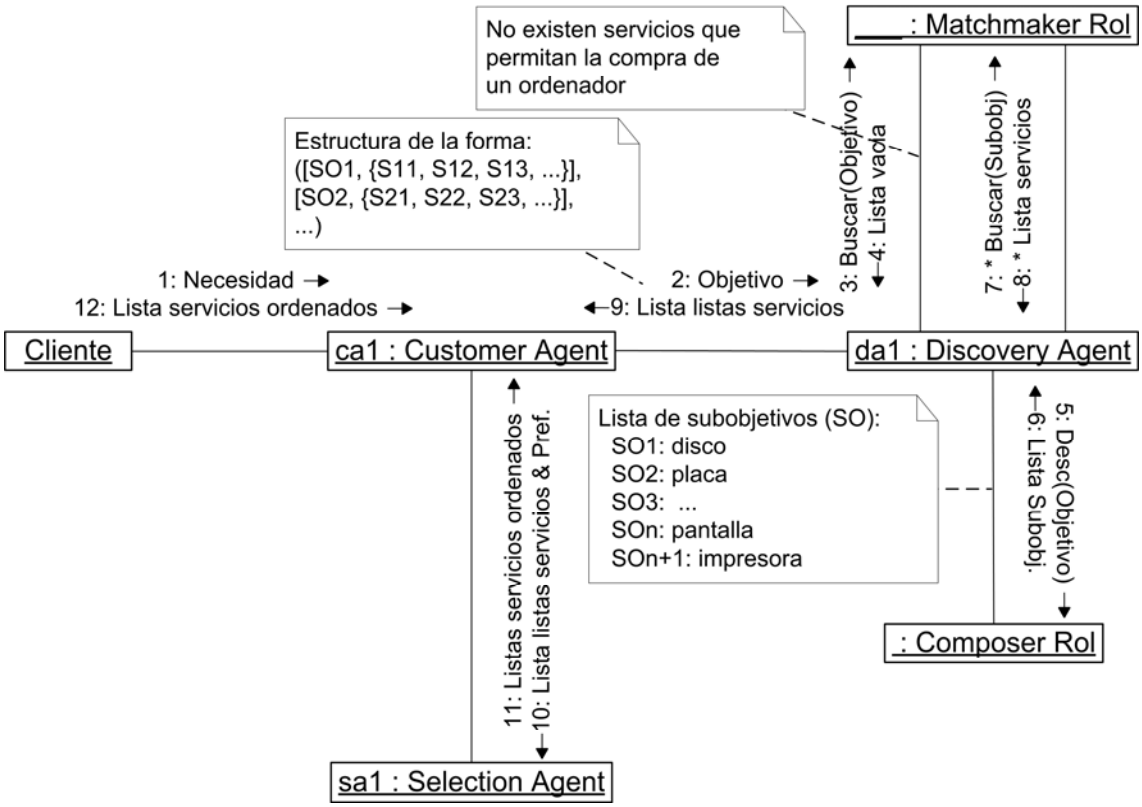


Fig. 99. Diagrama de colaboración para la tarea de ejemplo 2 – fase de composición

El usuario, por tanto, tiene que decidir qué servicio ejecutar para cada subobjetivo. Asumiendo que el usuario decide comprar, por ejemplo, el disco del "*Proveedor A*", la placa base del "*Proveedor B*" y la pantalla del "*Proveedor C*", la plataforma hace

posible la interactuación entre los tres Servicios Web presentes ("*Servicio Web A*", "*Servicio Web B*" y "*Servicio Web C*"). La selección realizada por el usuario llega al '*Customer Agent*' con el formato que se presenta a continuación (para n subobjetivos):

{[Subobjetivo$_1$, Método$_{X1}$-Servicio$_{Y1}$],
[Subobjetivo$_2$, Método$_{X2}$-Servicio$_{Y2}$],
[…, …],
[Subobjetivo$_n$, Método$_{Xn}$-Servicio$_{Yn}$]}

Haciendo uso de esta información y subobjetivo tras subobjetivo, el '*Customer Agent*' se pone en comunicación con los agentes '*Service Agent*' pertinentes, en este caso, 'sea1', sea2' y 'sea3'. Aquí, la invocación de los servicios se puede hacer en paralelo, ya que no existe ninguna relación de dependencia entre ellos. En general, para la composición no se tienen en cuenta constructores complejos que den lugar a la generación de flujos de trabajo completos, como los que se utilizan en los lenguajes para especificación de flujos de trabajo (p.ej. BPEL, YAWL, XPDL). En su lugar, se asume que el flujo de ejecución puede ser en paralelo o secuencial, dependiendo del caso concreto. La necesidad de ejecutar dos servicios de forma secuencial es patente para el sistema cuando éste no dispone de la información necesaria para ejecutar uno de los servicios, y esta información se obtiene como resultado de la ejecución de otro servicio. También es posible, como se comentó para el ejemplo anterior, que los datos para sustituir alguno de los parámetros en la invocación de un servicio no se encuentren disponibles ni en la consulta del usuario, ni en las preferencias establecidas por éste, ni se obtengan como respuesta de la invocación de algún otro servicio. En este caso, el cliente es informado y se le permite incluir los datos restantes para llevar a cabo con éxito la ejecución del servicio.

Cuando el '*Customer Agent*' recibe el resultado de cada una de las invocaciones por parte de los agentes '*Service Agent*', se debe realizar un proceso de integración. Para esto, el '*Customer Agent*' hace uso de la ontología del dominio, "*Ontología Y*", que le sirvió para generar el objetivo del usuario. Los resultados, una vez integrados, son devueltos al usuario final o cliente como respuesta a su consulta inicial. Los pasos que llevan a la invocación de los servicios y la posterior integración de los resultados se muestran en la Fig. 100.

Se comprueba, por tanto, cómo el marco de trabajo presentado en el capítulo anterior puede hacer frente a las nuevas condiciones (presentes entre los requisitos funcionales) relativas a la necesidad, en determinadas situaciones, de componer varios servicios para la consecución de los objetivos. Así, en la fase de descubrimiento, se incluye un componente dedicado a la descomposición de un objetivo en subobjetivos con mayor granularidad, para los cuales, se espera, sea más fácil encontrar servicios que los satisfagan. De igual modo, en la fase de invocación se tiene presente esta nueva necesidad, y se pueden invocar varios servicios en base a los subobjetivos presentes.

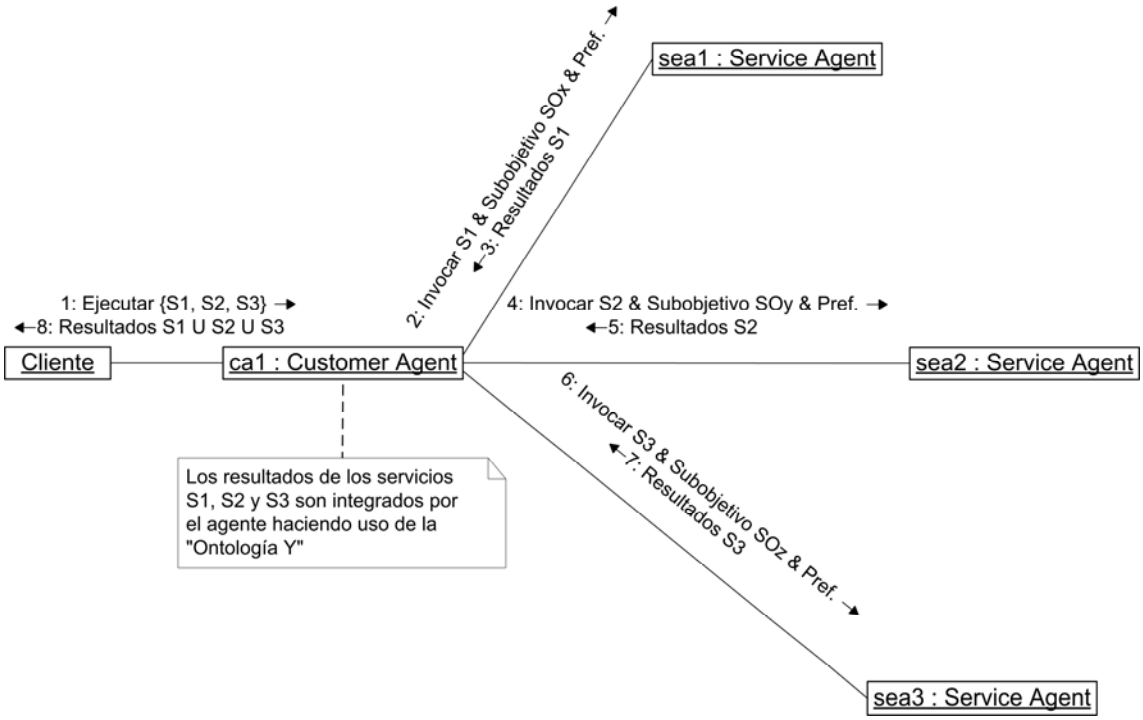


Fig. 100. Diagrama de colaboración para la tarea de ejemplo 2 – fases de invocación e integración

IV.3.3. Diferencias con otras plataformas

Si se compara el proceso que lleva a cabo el usuario para resolver el problema planteado mediante el uso del marco de trabajo, con las tareas que debería llevar a cabo en caso de enfrentarse a este mismo problema sin emplear una herramienta de Servicios Web Semánticos, las diferencias son patentes. Si bien, al igual que se comentó anteriormente, el usuario tiene que buscar "manualmente" los servicios que pueden satisfacer su necesidad y seleccionar aquellos más apropiados, el escenario que se describe en esta

sección presenta un problema adicional para el usuario, esto es, la necesidad de componer servicios. Ya no es un solo servicio el que se tiene que seleccionar y, posteriormente, ejecutar, sino varios que, además, pueden incorporar dependencias entre sí. Así, el usuario en la invocación tiene que tener en cuenta las dependencias, y, por último, interpretar los resultados devueltos tras la ejecución de cada uno de los servicios. Todo este proceso es automático si se utiliza un entorno de ejecución de Servicios Web Semánticos.

En cuanto a la relación entre el marco de trabajo y otras aplicaciones similares, no se pueden destacar diferencias significativas en relación al mecanismo de composición, más allá de las ya enumeradas en la sección anterior. En este marco de trabajo, se ha optado por una solución basada en el uso de técnicas de planificación y descomposición de objetivos. Para el futuro, queda pendiente la elaboración de mecanismos más avanzados basados en la definición de flujos de trabajo a partir de alguno de los lenguajes de especificación de flujos de trabajo como BPEL, YAWL, etc., o a través de Redes de Petri.

IV.4. Tarea de Ejemplo 3 – acceso integrado a fuentes de datos heterogéneas

El propósito de esta sección es mostrar el comportamiento del marco de trabajo en un entorno donde es necesaria la integración de varias fuentes de datos heterogéneas. Para esto, se propone un escenario a modo de ejemplo donde se precisa una herramienta capaz de llevar a cabo con éxito esta tarea. Posteriormente, se describe cómo el marco de trabajo da respuesta a las dificultades que surgen en el escenario propuesto. Finalmente, se compara la utilidad y el proceder del marco de trabajo con respecto a la de otros entornos de ejecución de Servicios Web Semánticos.

IV.4.1. Escenario

El escenario en este caso de ejemplo es un poco más complicado que los anteriores, aunque se repiten los actores involucrados: un usuario consumidor de servicios y varias compañías que venden productos informáticos y que actúan como proveedores de servicios. En la Fig. 101 se presenta el escenario completo. Éste lo componen, como en

los casos anteriores, tres proveedores de servicios: "*Proveedor A*", "*Proveedor B*" y "*Proveedor C*". Para cada proveedor, hay un servicio cuyas operaciones repercuten en el proceso de negocio de sus respectivos proveedores: "*Servicio A*", "*Servicio B*" y "*Servicio C*". Las descripciones semánticas de estos servicios ("*Descripción A*", "*Descripción B*" y "*Descripción C*"), se encuentran almacenadas en un repositorio ("*Repositorio X*") que se encuentra disponible a través de Internet. A diferencia de lo ocurrido en los escenarios anteriores, las anotaciones semánticas de las capacidades de los servicios se han elaborado utilizando distintas ontologías sobre el mismo dominio. Así, cada proveedor hace uso de su propia ontología, a saber, "*Ontología A*", "*Ontología B*" y "*Ontología C*", mientras que existe otra ontología de este mismo dominio disponible en Internet, la "*Ontología Y*". De esta forma, se recalca la heterogeneidad de las fuentes de datos a las que es necesario acceder.

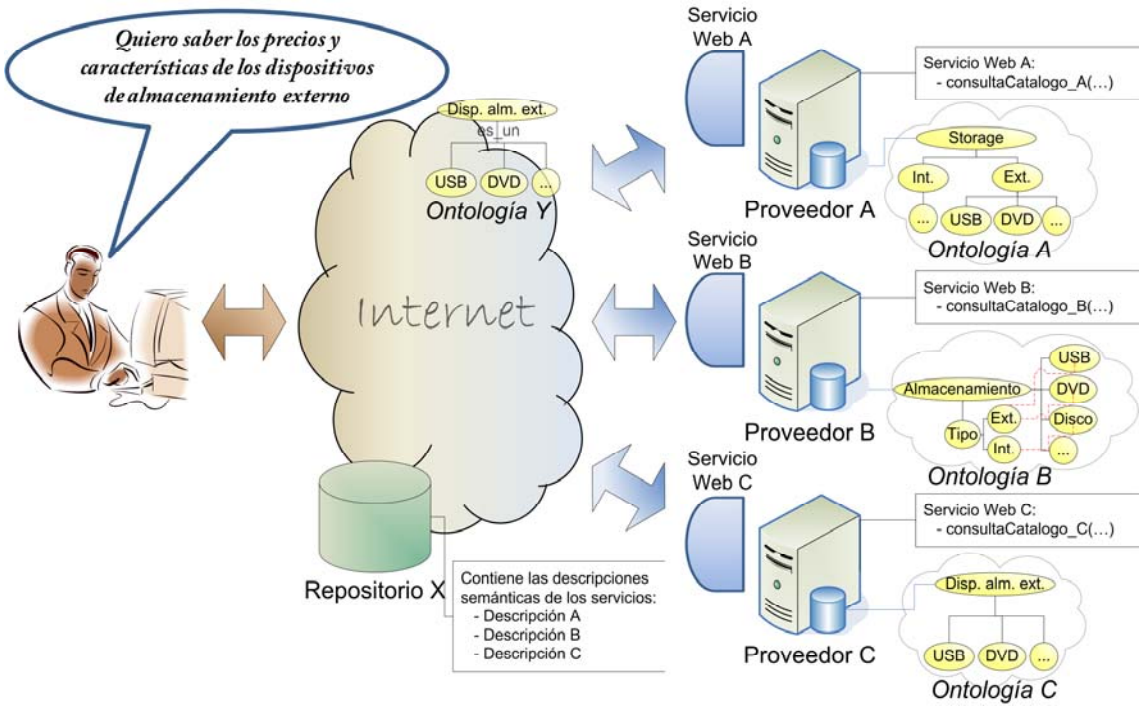


Fig. 101. Escenario para la tarea de ejemplo 3

En este caso de ejemplo, el usuario final, que juega el rol de consumidor de servicios, establece el objetivo de conocer la variedad existente en cuanto a dispositivos de almacenamiento externo, el precio de cada producto y sus características principales. Al igual que en el escenario anterior, por simplicidad, se omite la inclusión de

preferencias/condiciones por parte del usuario final. Opcionalmente, es posible que la entidad que juega el papel de consumidor de servicios sea otro componente software de cualquier índole (agente software, Servicio Web, etc.).

Por su parte, los servicios de los proveedores están compuestos por un único método "*consultaCatálogo(…)*" que, en todos los casos, devuelve la descripción de los productos que pertenecen a la categoría indicada en la llamada al método. A diferencia de lo ocurrido en los ejemplos anteriores, cada proveedor anota sus respectivos servicios haciendo uso de una ontología que difiere de la del resto. Por lo tanto, el "*Proveedor A*" utiliza la "*Ontología A*" para describir semánticamente el "*Servicio A*" (resultando en la "*Descripción A*"), el "*Proveedor B*" anota el "*Servicio B*" mediante la "*Ontología B*" (resultando en la "*Descripción B*"), y el "*Proveedor C*" hace uso de la "*Ontología C*" para indicar la funcionalidad del "*Servicio C*" semánticamente (resultando en la "*Descripción C*").

De forma resumida, este escenario supone el acceso a varias fuentes de información heterogéneas para dar respuesta a una consulta del usuario. A pesar de su relación con los procesos de descubrimiento y composición, en este ejemplo se pone especial énfasis en el problema de la interoperabilidad de datos, consecuencia de la heterogeneidad de los entornos abiertos y distribuidos como Internet.

IV.4.2. Aplicación del marco de trabajo

Los requisitos funcionales del marco de trabajo para satisfacer las necesidades del escenario presentado en la Fig. 101 son los siguientes:

1. La plataforma tiene que proporcionar una interfaz de usuario para la interacción con los clientes, tanto para usuarios humanos como para entidades software.
2. La plataforma tiene que descubrir los Servicios Web apropiados para una petición de usuario.
3. La plataforma tiene que seleccionar los Servicios Web más apropiados u ordenar la lista de servicios de acuerdo con algún criterio.
4. La plataforma tiene que invocar Servicios Web y proporcionar un completo entorno de ejecución de Servicios Web con funciones de control, manejo de excepciones, y soporte a interacciones opcionales con los usuarios.

5. La plataforma tiene que procesar los resultados de la invocación de Servicios Web para combinarlos en una única estructura entendible por el usuario, independientemente de los modelos de datos que manejen los distintos servicios, que pueden ser iguales o no.
6. La plataforma tiene que proporcionar interfaces para la interacción con las entidades proveedoras de servicios.

Este escenario de ejemplo incorpora un nuevo requisito funcional al que debe hacer frente el marco de trabajo, a la vez que afronta los ya enumerados en escenarios anteriores. Este nuevo requisito funcional es consecuencia de la utilización, por parte de los proveedores de servicios, de distintas ontologías para representar la conceptualización de un mismo dominio. Este hecho conlleva la aparición de problemas de interoperabilidad derivados de la heterogeneidad de los datos manejados en el entorno. En esta sección, se explica cómo el marco de trabajo propuesto en esta tesis doctoral acomete este nuevo reto.

El proceso comienza con el análisis de la consulta del usuario para generar un objetivo de acuerdo con la representación interna manejada por la plataforma. El responsable de esta tarea es el '*Customer Agent*', al que llega la consulta del usuario a través de la interfaz. Una vez se dispone de la representación ontológica del objetivo del usuario, que, para el escenario de ejemplo, se presenta en la Fig. 102, se le envía al '*Discovery Agent*'. Accediendo al "*Repositorio X*", el '*Discovery Agent*' descubre que, tanto el "*Servicio Web A*" como el "*Servicio Web B*" y el "*Servicio Web C*" disponen de los medios apropiados para satisfacer la necesidad del usuario (a través de los métodos "*consultaCatálogo_A(…)*", "*consultaCatálogo_B(…)*" y "*consultaCatálogo_C*"). Para esto, en la consulta realizada por el '*Discovery Agent*' sobre el "*Repositorio X*", se tienen que abordar los problemas de interoperabilidad asociados a la utilización de ontologías distintas para la definición del objetivo ("*Ontología Y*") y para la descripción semántica de los servicios ("*Ontología A*" para la "*Descripción A*", "*Ontología B*" para la "*Descripción B*", y "*Ontología C*" para la "*Descripción C*").

Fig. 102. Ontología del objetivo para tarea de ejemplo 3

Cuando se dispone de la lista de servicios (descripciones semánticas de Servicios Web) que cumplen el objetivo pretendido, ésta se le envía de vuelta al '*Customer Agent*', el cual delega en el '*Selection Agent*' la tarea de ordenar la lista conforme al valor de utilidad que se calcula teniendo en cuenta las preferencias de los usuarios y las condiciones que proponen los proveedores de servicios. La lista de servicios ordenada de mayor a menor utilidad se le envía al '*Customer Agent*', que la presenta al usuario final para que éste seleccione el servicio o servicios a ejecutar. En la Fig. 103, se muestra el diagrama de colaboración simplificado donde están presentes todas las interacciones que tienen lugar entre los distintos agentes de la plataforma desde el instante en que el usuario envía la consulta, hasta el instante en que se le enumeran los servicios capaces de complacer tal necesidad.

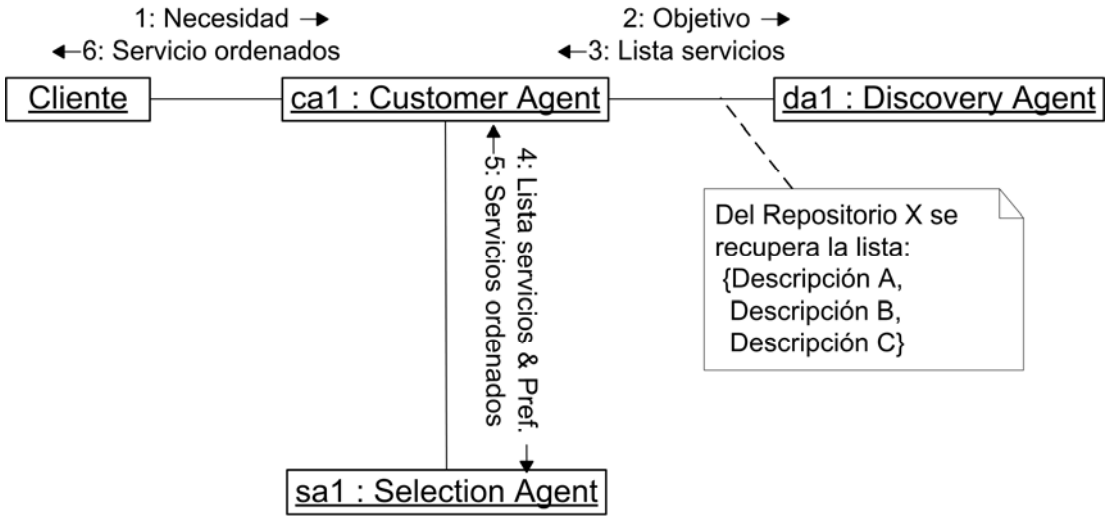


Fig. 103. Diagrama de colaboración para la tarea de ejemplo 3 – fases de descubrimiento y selección

En ese momento, el usuario final o cliente debe evaluar las distintas posibilidades y determinar qué servicio o servicios ejecutar. Para el escenario de ejemplo, y teniendo en cuenta que el usuario desea conocer el catálogo de todos los proveedores en lo que respecta a dispositivos de almacenamiento externo, los tres servicios disponibles son seleccionados para su ejecución, aunque todos ellos persigan un fin similar. Por tanto, y del mismo modo en que se trata una composición de servicios, el '*Customer Agent*' realiza la comunicación con los tres agentes '*Service Agent*' pertinentes, y comienza el proceso de ejecución de los servicios. En esta etapa, los agentes 'sea1', 'sea2' y 'sea3' (representantes de los servicios "*Servicio Web A*", "*Servicio Web B*" y "*Servicio Web C*" respectivamente) han de enfrentarse al problema de la heterogeneidad para ser capaces de realizar correctamente la invocación de cada uno de los servicios. Dado que, por un lado, las ontologías "*Ontología A*", "*Ontología B*" y "*Ontología C*" son las utilizadas por parte de los proveedores para anotar semánticamente las descripciones de sus servicios y, por otro lado, el sistema hace uso internamente de la "*Ontología Y*", deben existir reglas de correspondencia entre unas y otras para que el sistema sea capaz de realizar la tarea de invocación (del mismo modo que las necesitó para hacer el descubrimiento).

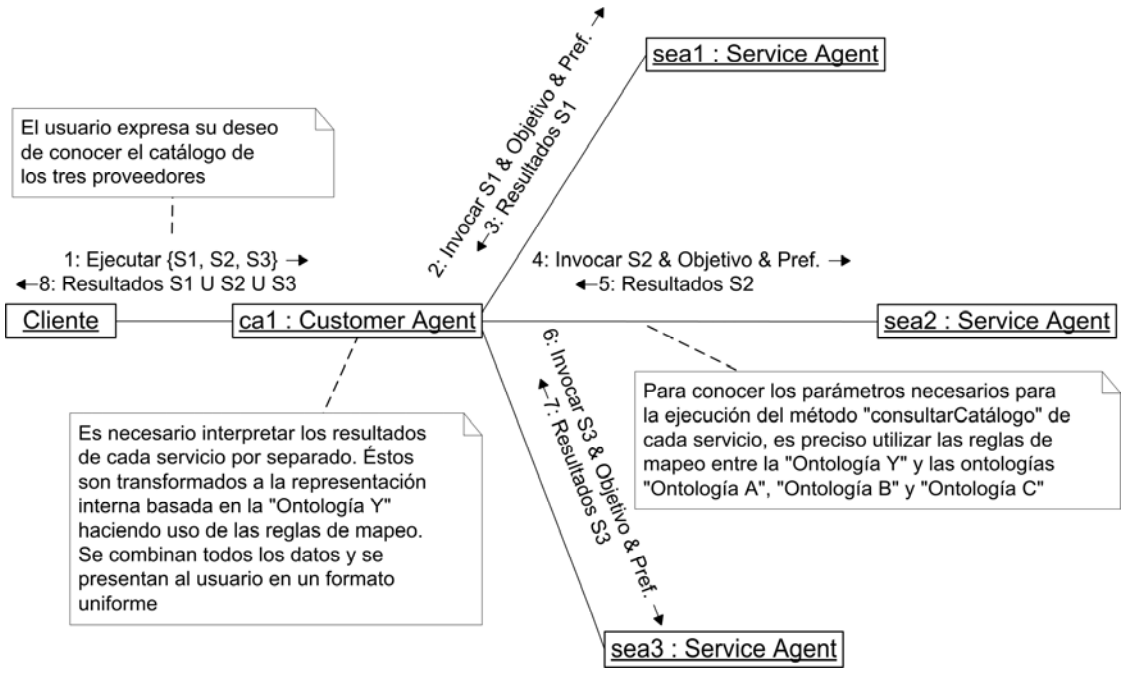


Fig. 104. Diagrama de colaboración para la tarea de ejemplo 3 – fases de invocación e integración

Una vez superadas las dificultades asociadas a la interoperabilidad, cada '*Service Agent*' establece correctamente los parámetros del método y realiza la ejecución. Nuevamente, ya que los resultados de la invocación de los servicios se recuperan conforme al modelo de datos provisto por el proveedor, el sistema debe reconciliar estos datos con la "*Ontología Y*". El '*Customer Agent*' es el responsable de consumar esta tarea. El agente en cuestión recibe, por separado, los resultados de cada invocación y, mediante la aplicación de las reglas de mapeo mencionadas anteriormente, combina todos los datos en una estructura de datos única y uniforme conforme a la "*Ontología Y*". Posteriormente, estos datos son presentados de forma combinada al usuario final a través de la interfaz de consumidor. Este proceso, se puede ver representado gráficamente en la Fig. 104.

Mediante este ejemplo, se ha demostrado la utilidad del marco de trabajo para resolver las dificultades relacionadas con el acceso integrado a fuentes de datos heterogéneas. Sin embargo, no es el propósito del marco de trabajo (ni era la intención del ejemplo que aquí se presenta) hacer frente al problema, mucho más amplio, de **integración de datos**. Para tratar las dificultades concernientes a este problema (heterogeneidad sintáctica, estructural, semántica o de sistema), es necesario modificar la plataforma en diversos aspectos y la inclusión de nuevos componentes, lo cual queda pendiente como trabajo futuro.

IV.4.3. Diferencias con otras plataformas

El principal problema al que se debe de enfrentar un usuario humano que no desee utilizar una herramienta de ejecución de Servicios Web Semánticos en este escenario de ejemplo con respecto a los anteriores será la interpretación de los datos, tanto de entrada como de salida de cada uno de los servicios. De este modo, además de realizar de forma manual las tareas de descubrimiento y selección de servicios, el usuario o cliente debe ser capaz, en el momento de la invocación, de determinar los datos a incluir como parámetros en las llamadas y el modo de interpretar los resultados, cuando cada uno de los servicio hace uso de un modelo completamente diferente. El marco de trabajo que se propone en esta tesis doctoral automatiza la mayor parte de este trabajo, liberando, así, a los usuarios de la realización de estas tareas.

El acceso integrado a fuentes de datos heterogéneas es un elemento de funcionalidad recurrente en todos los entornos de ejecución de Servicios Web Semánticos que han sido desarrollados hasta la fecha. En general, todas estas herramientas incorporan mecanismos de mediación que permiten hacer frente a este tipo de problemas. La solución adoptada en esta tesis doctoral, basada en la utilización de reglas de mapeo, es una más de las presentes en la literatura. En todo caso, las técnicas que se aplican en cada sistema dependen, fundamentalmente, de las facilidades que, para este propósito, proporciona la aproximación de Servicios Web Semánticos con la que el sistema trate. Así, si bien este es uno de los puntos fuertes en WSMO, donde se incorpora el concepto de mediador como elemento de primer nivel, en OWL-S es uno de los aspectos más criticados.

IV.5. Resumen

En este capítulo, se demuestra la utilidad del marco de trabajo propuesto en esta tesis doctoral por medio de su aplicación en varias tareas de ejemplo. Cada una de estas tareas plantea una serie de retos que la plataforma es capaz de afrontar. En particular, se describen tres ejemplos dónde el marco de trabajo debe demostrar sus habilidades. Para cada tarea de ejemplo se describe un escenario imaginario en el que un cliente desea realizar una tarea específica y tiene a su disposición numerosos servicios suministrados por diversos proveedores de servicios. Posteriormente, se examina el comportamiento del marco de trabajo al enfrentarse a cada una de las situaciones planteadas y, finalmente, se compara ésta situación con aquella en la que el usuario deba realizar la misma tarea pero sin emplear un entorno de ejecución de Servicios Web Semánticos, o utilice alguno de los desarrollados hasta el momento.

En la primera tarea de ejemplo, se analiza el modo de proceder del marco de trabajo en relación a las tareas básicas para la gestión y explotación de los Servicios Web Semánticos, como son el descubrimiento, la selección y la invocación. En este ejemplo, se constata como, mediante la aplicación de los distintos agentes que constituyen la plataforma, el sistema es capaz de hacer frente de todos los requisitos funcionales que plantea el escenario propuesto. En concreto, el '*Discovery Agent*', mediante consultas a

los repositorios donde se encuentran almacenadas las descripciones semánticas de los servicios, realiza con éxito el descubrimiento de los servicios. Por su parte, el '*Selection Agent*' es el encargado de realizar una ordenación de la lista de posibles servicios a ejecutar a partir de un valor de utilidad que se calcula en base a las preferencias de los usuarios y las condiciones propuestas por los proveedores para la ejecución de los servicios que estos suministran. Los agentes '*Service Agent*' son los responsables de interactuar con los servicios a los que representan, realizar las llamadas a los métodos y recoger los resultados. Por último, el '*Customer Agent*' controla todo lo relativo a la interacción con el usuario, procesamiento de la consulta y presentación de los resultados.

En el segundo escenario de ejemplo, el marco de trabajo debe hacer frente a las dificultades asociadas a la composición de servicios. Numerosos agentes se ven afectados por los nuevos requisitos relacionados con esta tarea en particular. En primer lugar, el '*Discovery Agent*', asumiendo el rol '*Composer Rol*', es capaz de descomponer el objetivo inicial en varios subobjetivos que, en su conjunto, abarcan todo lo establecido en el objetivo de origen, pero de un modo más granular. Entonces, para cada subobjetivo, el '*Discovery Agent*' descubre los servicios capaces de cumplirlo. Por su parte, el '*Selection Agent*' tiene en cuenta la descomposición de los objetivos y realiza una ordenación parcial de cada lista de servicios para cada subobjetivo. Por último, el '*Customer Agent*' es el responsable de enviar la petición de invocación a todos los agentes '*Service Agent*' que estén estipulados, y hacer frente a los problemas que puedan surgir debidos a las dependencias entre los servicios.

En la última tarea de ejemplo, se estudia la capacidad del sistema de proporcionar acceso a fuentes de datos heterogéneas. El escenario de ejemplo plantea una situación en que cada proveedor de servicios hace uso de ontologías diferentes para describir los servicios. De nuevo, diversos agentes se ven afectados por esta situación. El '*Discovery Agent*', dado que las descripciones semánticas de cada servicio se han desarrollado empleando distintas ontologías, tendrá que tener en cuenta este hecho para determinar los servicios que cumplen las condiciones del objetivo del usuario. Los agentes '*Service Agent*' que representen a los servicios, por su parte, interpretan los parámetros de entrada de los métodos a ejecutar con respecto a la ontología que maneja el sistema

haciendo uso de las reglas de mapeo. Finalmente, el '*Customer Agent*' recibe todas las respuestas y, al igual que los agentes '*Service Agent*', procesa los resultados para adecuarlos al modelo de datos que se utiliza internamente, para lo cual hace uso de las reglas de mapeo definidas entre las ontologías de los proveedores y la ontología del sistema.

CAPÍTULO V. UNA APLICACIÓN SOFTWARE PARA LA EJECUCIÓN DINÁMICA DE SWS

V.1. Introducción

En este capítulo, se describe la aplicación software que ha sido implementada utilizando como fundamento la base teórica del marco de trabajo presentado en el capítulo III. El objetivo general de la aplicación que aquí se define es la de servir como entorno integral para la ejecución dinámica de Servicios Web Semánticos.

En primer lugar, se identifican las herramientas y librerías que han sido fundamentales para el desarrollo de la herramienta. Seguidamente, se muestra la arquitectura general del sistema implementado, destacando las características generales de la aplicación, así como las limitaciones introducidas con respecto al marco de trabajo que constituye su base teórica. Por último, se describe la aplicación a partir de las interfaces con los tres usuarios externos identificados: desarrolladores de aplicaciones, proveedores de servicios y consumidores de servicios.

V.2. Fundamentos tecnológicos

Para el desarrollo de la aplicación software, se han empleado diversas herramientas así como numerosas librerías que han servido para facilitar la implementación de la funcionalidad del sistema en varios sentidos. En esta sección, se describen estos elementos, que han sido fundamentales para la construcción de la aplicación en cuestión.

V.2.1. Herramientas

Entre las herramientas que han sido utilizadas para el desarrollo de la aplicación mencionada arriba, se encuentran JADE, Tomcat, Sesame, MySQL, Java y Eclipse. A continuación, se realiza una breve descripción de cada una de ellas.

V.2.1.1. JADE

JADE[26] es la plataforma multi-agente que ha sido utilizada para la implementación del entorno de agentes. Entre sus características más importantes, se pueden destacar las siguientes: cumple el estándar FIPA que permite la interoperabilidad entre SMA, está completamente desarrollado en Java, es de código abierto y es el más utilizado por los investigadores en todo el mundo. En la sección I.3.5.2, se ofrece una descripción más detallada de la plataforma.

V.2.1.2. Tomcat

Tomcat[27] es un contenedor aplicaciones que implementa las especificaciones de tecnologías Java como "*Java Servlet*" y "*JavaServer Pages*" (JSP). Tomcat está escrito en Java, por lo que es posible instalarlo sobre cualquier sistema operativo que disponga de la máquina virtual Java, y se trata de un software de código abierto.

La última versión de Tomcat, la 6.x, implementa las especificaciones 2.5 de Servlet y 2.1 de JSP, además de soportar la versión 2.1 del lenguaje de expresión unificado ("*Unified Expression Language*") y ejecutarse sobre entornos Java SE 5.0 y posteriores. En versiones anteriores, se recomendaba la utilización del servidor web Apache en combinación con Tomcat dado que Apache era mucho más rápido sirviendo páginas Web. Sin embargo, en las últimas versiones de Tomcat, se ha mejorado bastante la eficiencia en este sentido y ya no es necesaria la utilización de Apache para la realización de dicha tarea (esto es, como servidor de paginas Web).

V.2.1.3. Sesame

Sesame[28] es un marco de trabajo de código abierto que permite almacenar, inferir y consultar datos en formato RDF. Fue desarrollado como parte del proyecto "On-To-Knowledge" entre 1999 y 2002 y, en la actualidad, está siendo mantenido por una compañía de software holandesa denominada '*Aduna*'.

[26] http://jade.tilab.com/

[27] http://tomcat.apache.org/

[28] http://www.openrdf.org/

Para el desarrollo de la aplicación software, se ha empleado la versión 2 de esta herramienta, que se encuentra, en el momento de escribir esta tesis, en fase beta. Esta nueva versión es incompatible con versiones anteriores e incorpora las siguientes características principales: precisa de la versión 5 de Java o posteriores, da soporte a información contextual que permite seguir el rastro de unidades de datos RDF, permite la gestión de transacciones y '*rollbacks*', incorpora un protocolo HTTP '*RESTful*' (transferencia de estado representacional) que da soporte al protocolo SPARQL y al formato XML de resultados de consultas SPARQL, y, finalmente, Sesame 2 soporta el lenguaje de consultas SPARQL, hecho por el cual este marco de trabajo fue elegido.

En esta tesis, Sesame se ha utilizado como mero repositorio de datos en formato RDF, mientras que todo el soporte para la gestión de ontologías se realiza a través de la librería Jena, que se describe en otro apartado de este capítulo.

V.2.1.4. MySQL

MySQL[29] es un sistema de gestión de base de datos relacional multihilo y multiusuario. MySQL es una herramienta de software libre con licencia de uso GNU GPL, de forma que empresas que lo incorporen a sus productos comerciales deberán comprar una licencia para tal efecto. MySQL proporciona un servidor de consultas SQL rápido y robusto.

La última versión, MySQL 5.0, incluye, entre otras, las siguientes características: soporte de un amplio subconjunto de la especificación ANSI SQL 99 y algunas extensiones, permite los procedimientos almacenados, se puede instalar en gran cantidad de plataformas y sistemas, proporciona diversas opciones de almacenamiento (cada opción proporciona una relación velocidad/número de operaciones diferente), da soporte a las transacciones y las claves foráneas, permite una conectividad segura (soporte para SSL), y permite la replicación. Además, existen APIs que permiten, a aplicaciones escritas en diversos lenguajes de programación (Java entre ellos), acceder a las bases de datos MySQL.

[29] http://www.mysql.com/

V.2.1.5. Java y Eclipse

El lenguaje de programación empleado para la implementación de la aplicación software ha sido Java[30]. Esta elección se debe a las numerosas propiedades y ventajas que tiene Java con respecto a otros lenguajes de programación (Flanagan, 1999): orientado a objetos (otorga una gran flexibilidad a la hora de presentar jerarquías complejas), distribuido (aporta mecanismos para interactuar con otras máquinas mediante TCP/IP), robusto (tipado estáticamente, generación de excepciones, etc.), seguro (comprobaciones de privilegios), multiplataforma (ejecución sobre la máquina virtual Java), soporte Web (numerosas APIs para el desarrollo de aplicaciones Web), en expansión (lenguaje muy popular que no deja de evolucionar). Una razón adicional por la que se escogió este lenguaje de programación es que JADE, la plataforma multi-agente que se ha empleado en el desarrollo de esta tesis doctoral, está implementado en Java.

Por otro lado, Eclipse[31] ha sido el entorno de desarrollo utilizado. Eclipse es una plataforma de software de código abierto independiente de la plataforma, que incorpora un entorno integrado de desarrollo (IDE) para la elaboración de aplicaciones Java. Entre las características principales de Eclipse se encuentran las siguientes: editor de texto, resaltado de sintaxis, compilación en tipo real, pruebas unitarias con JUnit, control de versiones con CVS, integración con Ant, distintos asistentes para creación de proyectos, clases, etc., y refactorización.

V.2.2. Librerías

Además de estas herramientas, se han empleado numerosas librerías o bibliotecas que contienen funcionalidad de interés para la aplicación. Entre éstas, se pueden destacar JadeGateway (para la interacción Aplicación Web – SMA), Jena (para la gestión de las ontologías), Axis2 (para la comunicación con los Servicios Web), y KAText (para el reconocimiento del lenguaje natural). En los siguientes apartados, se ofrece una breve descripción de cada una de ellas.

[30] http://java.sun.com/

[31] http://www.eclipse.org/

V.2.2.1. JadeGateway

JadeGateway no se trata, en realidad, de una librería completa, sino que es una clase particular contenida en el paquete '*jade.wrapper.gateway*' del API de Jade, y que se utiliza para la comunicación con la aplicación Web. Este paquete, además de la clase '*JadeGateway*', contiene la clase '*GatewayAgent*', que también resulta fundamental para definir el mecanismo de interacción con el servidor de aplicaciones.

Como se ha mencionado anteriormente, las clases del paquete en cuestión son los pilares básicos que sustentan la comunicación del SMA con componentes externos, en este caso, la aplicación Web que se ejecuta en el servidor Tomcat. La solución aportada a este problema se basa en el trabajo de Kelemen (2006) y se encuentra ilustrado gráficamente en la Fig. 105.

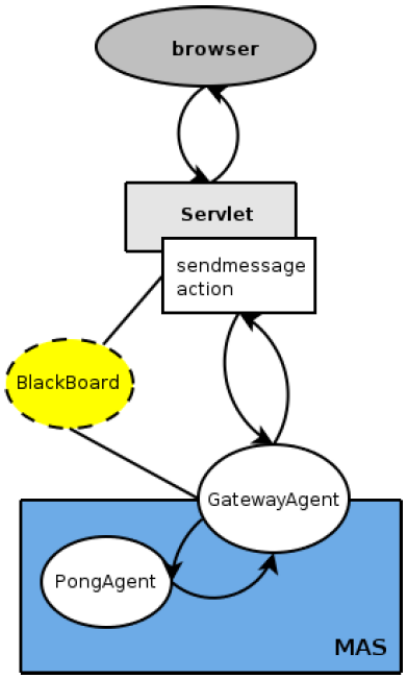


Fig. 105. Mecanismo de interacción SMA-Aplicación Web (Kelemen, 2006)

Esta técnica se basa en la definición de una agente, el '*Gateway Agent*', que sirva de puerta de comunicación entre el SMA y la aplicación. Cronológicamente, el proceso es el siguiente: (i) el usuario provoca un evento en la interfaz Web del navegador que supone la generación de un mensaje que le llega al Servlet; (ii) el Servlet, después de procesar el mensaje, invoca la acción '*sendmessage*'; (iii) esta acción crea un objeto

'*BlackBoard*' donde se incluye el agente al que está destinado el mensaje y el contenido del mismo; (iv) el agente '*Gateway Agent*', que también tiene acceso al objeto '*BlackBoard*', analiza el contenido del mismo y genera el mensaje apropiado, ahora sí dirigido al agente para el que estaba destinado; (v) el agente receptor realiza la tarea encomendada y devuelve el resultado nuevamente al '*Gateway Agent*'; (vi) este agente genera el objeto '*BlackBoard*' correspondiente, y lo envía al Servlet; (vii) finalmente, el Servlet envía la respuesta del sistema al navegador.

V.2.2.2. Jena

Jena[32] es un entorno de desarrollo de código abierto para la Web Semántica en Java. Proporciona un API que permite extraer datos desde y escribir en grafos RDF. Los grafos se representan como un "modelo abstracto". Un modelo puede provenir de datos a partir de ficheros, bases de datos, URLs, o una combinación de éstos. Se puede consultar sobre un modelo por medio de SPARQL.

El entorno de desarrollo Jena consta de un API para RDF, mecanismos de lectura y escritura de tripletas RDF en distintos formatos (p. ej., RDF/XML, N3 y N-Triples), un API para OWL, almacenamiento en memoria y almacenamiento persistente (da soporte a diversos gestores de bases de datos relacionales como MySQL, PostgreSQL, Oracle y Microsoft SQL Server), y un motor de consultas SPARQL.

Jena es similar, por tanto, a Sesame, que fue descrito con anterioridad. Proporciona soporte para el lenguaje OWL, e incorpora diversos razonadores básicos, permitiendo asociar nuevos razonadores al marco de trabajo. Entre los razonadores básicos incluidos en el núcleo de Jena2 se encuentran los siguientes: RDFSRuleReasoner (un razonador sobre RDFS), una implementación basada en reglas del subconjunto OWL-Lite, y un motor de reglas de propósito general, utilizado para implementar los razonadores para RDFS y para OWL, pero que también está disponible para uso general. Además, esta versión de Jena mantiene un mínimo soporte para el lenguaje DAML. Para poder llevar a cabo un razonamiento completo sobre la versión DL de OWL, se recomienda la utilización de un razonador externo como Pellet, Racer o FaCT.

[32] http://jena.sourceforge.net/

Entre las características principales de Jena, se puede destacar que permite gestionar todo tipo de ontologías, almacenarlas y realizar consultas sobre aquellas (como, por ejemplo, añadir hechos, borrar hechos y editar hechos). Proporciona soporte para los lenguajes DAML, RDF y OWL, y es independiente del lenguaje.

V.2.2.3. Axis2

Axis2[33] es una implementación del protocolo SOAP desarrollada por "*The Apache Software Foundation*". Apache Axis2 proporciona un marco de trabajo completo para el desarrollo de Servicios Web basado en XML y de código abierto. Esta nueva versión supone una clara evolución con respecto a la anterior, Apache Axis. La nueva arquitectura en que Axis2 se basa, es más flexible, eficiente y configurable que la arquitectura previa en que estaba basado Axis1.x.

Apache Axis2, además de dar soporte para las especificaciones 1.1 y 1.2 de SOAP, también incorpora soporte para los Servicios Web en estilo REST (también denominados *Servicios Web RESTful*), que se basan en emular el protocolo HTTP, y otros protocolos similares, empleando operaciones estándar (p.ej., GET, PUT y DELETE) para interaccionar con los recursos. Una de las características principales de Axis2 con respecto a versiones anteriores, aparte de su mejora en eficiencia, es la configurabilidad. Axis2 está diseñada para permitir la adición de módulos que extiendan su funcionalidad en aspectos tales como la seguridad y la fiabilidad. Entre los módulos disponibles actualmente o en proceso de desarrollo, se pueden destacar los que implementan las siguientes especificaciones de Servicios Web: '*WS-ReliableMessaging*', '*WS-Coordination*', '*WS-Security*', y '*WS-Addressing*'.

Por tanto, como se mencionó anteriormente, Axis2 añade nuevas propiedades, mejoras e implementaciones de especificaciones industriales. Entre los atributos clave a resaltar, se encuentran los siguientes: rapidez (es más eficiente que sus predecesores), bajo consumo de memoria, Servicios Web asíncronos, flexibilidad (el desarrollador puede añadir extensiones), estabilidad, soporte a protocolos de transporte (p.ej., HTTP, SMTP, JMS, TCP), y soporte a WSDL (especificaciones 1.1 y 2.0). Además, existen

[33] http://ws.apache.org/axis2/

dos implementaciones principales de este motor de Servicios Web, el "*Apache Axis2/Java*" y el "*Apache Axis2/C*", para los lenguajes de programación Java y C++, respectivamente.

V.2.2.4. KAText

Entre los mecanismos que se le ofrecen al usuario consumidor de servicios para interaccionar con la aplicación software, se encuentra la consulta a partir de texto en lenguaje natural. La aplicación, para "entender" la necesidad de un usuario, debe ser capaz de procesar esa entrada en lenguaje natural y generar un objetivo de acuerdo con un modelo de datos propio. Con este propósito, el SMA implementado hace uso internamente de una herramienta para el procesamiento del lenguaje natural. En este sentido, la tarea del procesador de lenguaje natural es la de filtrar y procesar la entrada del usuario a partir de texto en lenguaje natural, y determinar las entidades ontológicas (esto es, conceptos, atributos y relaciones) que están incluidas en el texto.

El procesador de lenguaje natural (también conocido como "reconocedor de lenguaje natural") utilizado para el desarrollo de esta aplicación está basado en KAText, la metodología propuesta en (Valencia-García et al., 2004; Valencia-García, 2005) para extraer conocimiento desde texto. Esta metodología, que emplea ontologías y una técnica para al adquisición incremental de conocimiento denominada MCRDR (Kang, 1996), tiene su fundamento en la idea de que las relaciones entre conceptos están, generalmente, asociadas a los verbos en el lenguaje natural. Esta metodología hace uso de la técnica mencionada y de la categoría gramatical de las palabras en la frase actual para inferir otras entidades de conocimiento, de forma que se genere una ontología a partir de un fragmento de texto.

El proceso de adquisición de conocimiento de KAText está dividido en tres fases secuenciales, a saber, "*POS-Tagging*", "*Concept search*" e "*Inference*". El objetivo principal de la fase de "*POS-Tagging*" es obtener la categoría gramatical de cada palabra en la oración actual. Para este propósito, se utiliza el etiquetador sintáctico (*POS-Tagger*) descrito en (Ruiz-Sánchez et al., 2003). Por otro lado, durante la fase de "*Concept Search*", se identifican las expresiones lingüísticas que representan conceptos. Las asociaciones entre las expresiones lingüísticas y los conceptos deben haber sido

previamente almacenadas en una base de conocimiento de conceptos durante la fase de entrenamiento del sistema. Como resultado de la fase "*Concept Search*", se obtienen todas las expresiones del fragmento que se encontraban previamente en la base de conceptos. Finalmente, en la fase "*Inference*", se utiliza la teoría, comúnmente aceptada, de que las relaciones entre los conceptos están asociadas a verbos en lenguaje natural. El componente MCRDR, que se utiliza para obtener las relaciones entre conceptos, está formado por una base de conocimiento que contiene expresiones lingüísticas que representan relaciones conceptuales genéricas y por un subsistema que infiere los participantes en dichas relaciones.

El *modus operandi* completo es el siguiente. En primer lugar, se identifica el verbo de la frase actual. Entonces, el sistema busca el tipo de relación semántica que está asociado a dicho verbo. Una vez que se ha encontrado esta relación, el subsistema MCRDR se aplica para extraer conocimiento por medio de la categoría gramatical de las palabras, su posición en la frase y el tipo de relación que se ha asociado al verbo.

Para el correcto funcionamiento de la herramienta, es necesaria una fase inicial de entrenamiento en la que un experto establezca los patrones que se utilizarán por parte del sistema para obtener los conceptos y las relaciones de la frase introducida por el usuario. Esta fase de entrenamiento se basa en el estudio de un conjunto de sentencias de un texto en el dominio de aplicación en el que se quiera entrenar la herramienta. En este proceso, los expertos indican los conceptos, atributos de esos conceptos, valores de esos atributos y relaciones entre los conceptos relevantes en cada una de las oraciones del texto, creándose de este modo patrones que serán los que aplique la herramienta de forma automática una vez haya sido correctamente entrenada. De este modo, cuando el usuario real introduce una oración, el sistema debe ser capaz de obtener de manera automática los conceptos, atributos, valores y relaciones asociados a la oración en cuestión.

A pesar de que KAText es, en conjunto, una aplicación de escritorio, también proporciona una interfaz muy sencilla para su utilización a modo de librería en terceros sistemas. En esta interfaz, empleada para el desarrollo de la aplicación software que forma parte de esta tesis doctoral, se encuentra disponible un único método que

devuelve los conceptos, relaciones, atributos, y valores de los atributos de la frase en lenguaje natural enviada como entrada.

V.2.2.5. Planificador

STRIPS (Fikes & Nilsson, 1971) es un planificador lineal que intenta encontrar una secuencia de operadores en un espacio de modelos del mundo ('*world models*') para transformar un modelo inicial dado a un modelo final donde se pueda probar que las fórmulas de que está formado son verdaderas. Un modelo del mundo se representa como un conjunto de fórmulas de predicados de primer orden. Para cada modelo del mundo, se definen un conjunto de operadores aplicables que transforman un modelo en otro. Un demostrador de teoremas puede encontrar una composición de operadores para transformar un modelo del mundo inicial a otro que satisfaga una condición de fin. Cada operador en STRIPS se compone por:

- **Precondiciones**: condiciones que se deben de cumplir para que la acción relacionada con un operador pueda ser ejecutada.
- **Borrados**: conjunto de fórmulas que dejarán de ser verdad una vez se haya aplicado este operador; luego, el planificador tiene que borrarlas del modelo del mundo actual.
- **Añadidos**: conjunto de fórmulas que serán verdad después de que el operador haya sido aplicado; luego, el planificador debe añadirlas al modelo del mundo actual.

En esta tesis doctoral, la planificación ha sido empleada para la tarea de composición de Servicios Web. Según el modelo teórico planteado, la composición de servicios consiste en la división de un objetivo en subobjetivos para los que se buscarán los servicios correspondientes. En este sentido, en el momento de realizar la planificación, se parte de un estado inicial vacio y un objetivo que coincide con el objetivo que el usuario desea alcanzar. Los operadores a aplicar para llegar al mundo final en el que el objetivo haya sido satisfecho son las distintas acciones que, ejecutadas de forma secuencial, dan lugar a la consecución del objetivo, esto es, los subobjetivos. Por tanto, los elementos que constituyen todo lo necesario para definir un problema de planificación (posibles estados del sistema, acciones que se pueden llevar a cabo, estado

inicial y objetivo), dependen del dominio de aplicación y, consecuentemente, deben ser definidos para cada aplicación que se desee desarrollar.

La librería de la que se ha hecho uso para implementar el planificador ofrece la flexibilidad suficiente para satisfacer la condición resaltada anteriormente, esto es, la definición de un conjunto de operadores concretos para cada aplicación y dominio. Esta librería[34], desarrollada en la Universidad de Sheffield, se basa en el acceso a un conjunto de ficheros donde se definen los operadores de acuerdo con sus partes constituyentes (precondiciones, borrados y añadidos). De este modo, con sólo modificar los ficheros a los que tiene acceso la plataforma, se consigue definir un conjunto nuevo de operadores para el planificador.

V.3. Framework Multi-Agente para la Ejecución Dinámica de SWS

Con arreglo al marco de trabajo teórico presentado en el capítulo III, la aplicación software implementada está compuesta por un entorno multi-agente que asume toda la funcionalidad del sistema. Para facilitar a los usuarios el acceso a la funcionalidad aportada por el SMA, se ha desarrollado una aplicación Web que constituye la interfaz con los usuarios externos identificados: desarrolladores, que ajustan la plataforma a las condiciones de un problema concreto, proveedores de servicios, y consumidores de servicios. En la Fig. 106, se representa gráficamente la arquitectura global del sistema.

En esta sección, se examinan algunos aspectos relativos a la implementación del SMA y los distintos mecanismos de gestión de los Servicios Web. Además, se incluyen detalles de la estructura de la aplicación Web, la interacción con el SMA, y el acceso a la base de datos.

[34] http://www.dcs.shef.ac.uk/~pdg/com1080/java/strips/

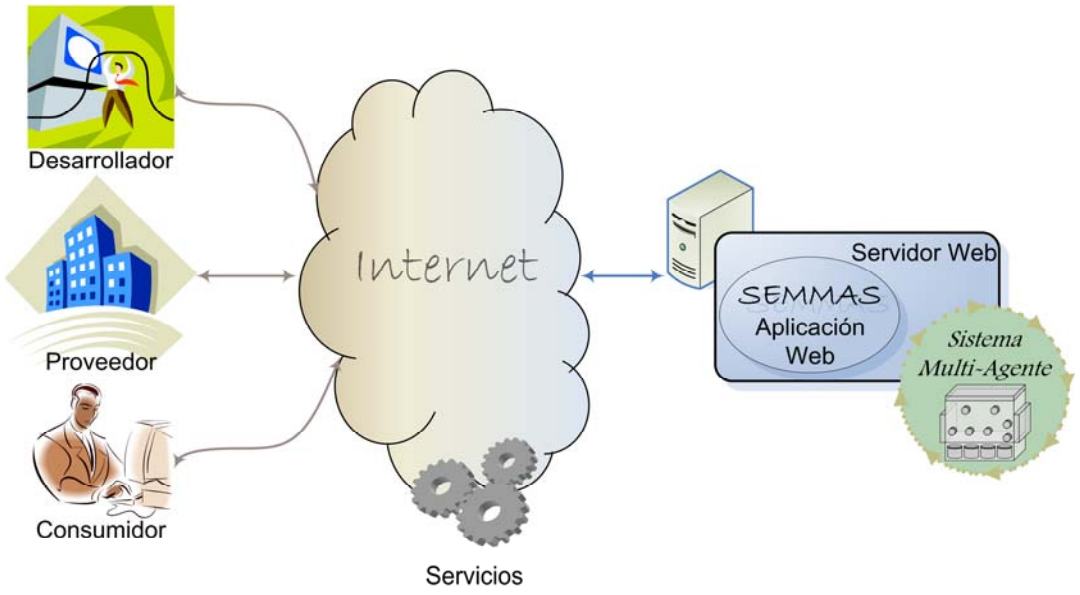


Fig. 106. SEMMAS – Aplicación Web para ejecución de Servicios Web Semánticos

V.3.1. Agentes, comportamientos y roles

Siguiendo las pautas establecidas en la fase de diseño, el SMA implementado consta de siete agentes principales, a saber, '*Broker Agent*', '*Customer Agent*', '*Discovery Agent*', '*Framework Agent*', '*Provider Agent*', '*Selection Agent*' y '*Service Agent*'. Adicionalmente, se ha incorporado al entorno un nuevo agente, el '*MyGatewayAgent*', que es el responsable de mediar en la comunicación entre la aplicación Web y los agentes de la plataforma.

Para definir la funcionalidad de cada agente, Jade, la plataforma multi-agente utilizada para desarrollar el SMA, hace uso del concepto '*Comportamiento*'. Así, el modo de proceder de un agente viene definido por los comportamientos que asume. Asimismo, con el fin de seguir fielmente la especificación teórica del marco de trabajo, se ha empleado el concepto '*Rol*'. En particular, a cada comportamiento se le asocia uno o varios roles que constituyen la verdadera funcionalidad proporcionada por dicho comportamiento. A nivel de comportamientos, se controla únicamente la comunicación entre agentes. Un mensaje enviado por un comportamiento concreto de un agente particular es recibido por el comportamiento, de otro agente, encargado de tratar dicho mensaje. Es durante el análisis y procesado de dicho mensaje cuando puede ser preciso invocar una determinada funcionalidad de alguno de los roles asociados al

comportamiento en cuestión. En la Tabla 10, se muestra la relación entre los distintos comportamientos que han sido identificados y los roles que se definieron en el capítulo III. Esta asociación es completamente estática, y ha sido definida en tiempo de implementación, de forma que no se puede modificar dinámicamente.

Comportamiento	Roles necesarios
Broker Behaviour:	• Broker • Ontology Manager
Consumer Representative Behaviour:	• Consumer Representative • Ontology Manager
Global Monitor Behaviour:	• Global Monitor
Invoker Behaviour:	• Invoker • Ontology Manager
Matchmaker Behaviour:	• Matchmaker • Composer • Ontology Manager
Monitor Behaviour:	• Monitor
Negotiation Responder Behaviour:	• Negotiator • Ontology Manager
Platform Manager Behaviour:	• Platform Manager
Provider Representative Behaviour:	• Provider Representative
Selector Behaviour:	• Selector • Negotiator • Ontology Manager
Service Representative Behaviour:	• Service Representative

Tabla 10. Asociación en tiempo de diseño entre Comportamientos y Roles

A diferencia de lo que ocurre en la relación entre comportamientos y roles, la asociación entre agentes y comportamientos, que viene especificada en la Tabla 11, puede evolucionar en tiempo de ejecución, dependiendo de los roles que juegue cada uno de los agentes. Como se puede comprobar en la tabla referida, cada agente está estrechamente relacionado con uno o varios comportamientos que debe asumir forzosamente. En consecuencia, los roles conectados con estos comportamientos deben ser obligatoriamente jugados por el agente en cuestión. Existe, además, para cada

agente, una lista de comportamientos que ese agente, de acuerdo con los objetivos que persigue (ver Tabla 5, pag. 159), puede asumir opcionalmente. En general, la decisión de llevar a cabo un comportamiento determinado u otro se toma en función de los roles que el agente acepte.

Tipos de agente	Comportamientos asociados
Broker Agent	Obligatorios: o Broker Behaviour
Customer Agent	Obligatorios: o Customer Representative Behaviour Posibles: o Monitor Behaviour o Broker Behaviour o Selector Behaviour
Discovery Agent	Obligatorios: o Matchmaker Behaviour Posibles: o Selector Behaviour
Framework Agent	Obligatorios: o Platform Manager Behaviour o Global Monitor Behaviour
Provider Agent	Obligatorios: o Provider Representative Behaviour Posibles: o Monitor Behaviour
Selection Agent	Obligatorios: o Selector Behaviour
Service Agent	Obligatorios: o Service Representative Behaviour o Invoker Behaviour o Negotiation Responder Behaviour Posibles: o Broker Behaviour o Monitor Behaviour

Tabla 11. Relación de comportamientos que muestran los agentes, necesarios y permitidos

Por tanto, como se destacó anteriormente, los roles son los responsables, en última instancia, de proporcionar la funcionalidad que exhibe el SMA. En consonancia con lo establecido por el marco teórico, y con la finalidad de dotar a la aplicación software con la flexibilidad y modularidad pretendida, se ha diseñado, para cada rol, una interfaz básica que, extendida por los desarrolladores, permite añadir nuevos algoritmos y

técnicas para las distintas tareas de que consta el sistema. En la Tabla 12, se muestran los distintos métodos de que consta cada interfaz.

Roles	Nomenclatura de los métodos
Broker	• Model dataBroker(Model unknown, Model source, Model target)
Composer	• Collection<Model> decompose(Model goal)
Consumer Representative	• Model getGoal(String query) • Model getGoal(URL url) • Model getGoal(File file) • String formatResults(Pref userPref, Collection<Object> results)
Global Monitor	• int evaluate(AgentStatus as) • Action compensation(int errorCode)
Invoker	• String callservice(String url, String method, String ns, C2I_Message m)
Matchmaker	• ServiceList selectServices(Model goal, Map<String, String> repositories)
Monitor	• int evaluate(Contract c, ExecutionStatus es) • Action compensation(int errorCode)
Negotiator	• LinkedList<CFP> generateCFP (Map<AID, Service> receivers) • Proposal generateProposal(CFP cfp, ServicePreferences sp, ProviderPreferences pp)
Ontology Manager	• Collection<String> getServices(String s) • ServiceList getInfoServices(Collection<String> servicesURI) • Model loadOntology(URL o)
Platform manager	• boolean addNewAgent(int agentType, String agentName, SystemStatus ss) • void deleteAgent(int agentType, String agentName)
Provider Representative	• Proposal getProposal(Service service)
Selector	• ServiceListList sort(ServiceListList sll, QueryPreferences qp, ConsumerPreferences cp)
Service Representative	• Proposal generateProposal(CFP cfp, ServicePreferences sp, ProviderPreferences pp)

Tabla 12. API de los roles en el sistema

V.3.2. Tareas básicas

Una vez descrita en detalle la arquitectura general del sistema, constituida por el SMA y la aplicación Web, y los principales elementos que conforman el SMA (esto es, agentes, comportamientos y roles), es necesario indicar qué técnicas y algoritmos han sido implementados para el procesamiento de las tareas básicas concernientes a la gestión de los Servicios Web. De acuerdo con lo afirmado en la sección anterior, la implementación de las tareas básicas se realiza en los distintos roles identificados. Además, esta implementación básica inicial de los roles tiene, obligatoriamente, que cumplir las interfaces que se definieron en la Tabla 12. De esta forma se mantiene la compatibilidad con el sistema, permitiendo incluso el desarrollo de más de un rol de un mismo tipo. Esto supone una clara ventaja, ya que permite resolver situaciones en las que, por ejemplo, el sistema deba tratar con (alguna de) las distintas especificaciones de Servicios Web Semánticos (esto es, OWL-S, WSMO, WSDL-S, etc.) a la vez.

En este apartado, se describen los mecanismos empleados para tareas tales como el procesamiento de las consultas de los usuarios, el descubrimiento y la composición de servicios, la selección y la invocación. De esta forma, por tanto, se cubre prácticamente todo el ciclo de vida de una interacción con el usuario.

V.3.2.1. Procesar las consultas de los usuarios

Uno de los mayores problemas asociado a los entornos de ejecución desarrollados hasta el momento en torno a especificaciones de Servicios Web Semánticos como, OWL-S, WSMO, WSDLS-S, etc., es su carencia de una interfaz de usuario capaz de tratar con los distintos perfiles de clientes disponibles. En particular, mientras algunos de los usuarios de la plataforma es posible que sean capaces de describir un objetivo por medio de una ontología conforme al modelo esperado por el sistema, otros (la mayoría) nunca se habrán enfrentado al desafio de desarrollar una ontología (por muy simple que sea). Por ende, es necesario ofrecer a este tipo de usuarios otros mecanismos alternativos para facilitar su interacción con el sistema.

La aplicación software desarrollada en esta tesis doctoral incorpora tres métodos para que un cliente, consumidor de servicios, pueda indicar su objetivo al sistema. Dos de ellos son muy básicos, y están destinados a lo que se podría denominar “usuarios

expertos". Estos usuarios desarrollan una ontología con el objetivo que persiguen y se lo envían al sistema de este modo. A este tipo de usuario se le ofrecen dos alternativas principales: transmitir un archivo que contiene la ontología del objetivo y que está albergado en el equipo de sobremesa del usuario, o indicar la URL donde se encuentra disponible tal ontología. En ambos casos, el problema se reduce a generar un modelo ontológico interno a partir de los datos de entrada. Como ya se destacó previamente, se ha hecho uso de la librería "Jena" (ver sección V.2.2.2) para la gestión y el manejo de las ontologías por parte del sistema.

En el caso de los "usuarios no expertos", se ha incorporado un mecanismo para que la plataforma acepte como entrada consultas a través de texto en lenguaje natural. Para esto, se ha empleado el reconocedor de lenguaje natural KAText, descrito en detalle en la sección V.2.2.4. Este procesador de lenguaje natural transforma una frase escrita en lenguaje natural en una ontología con los elementos de conocimiento (esto es, conceptos, relaciones, atributos y axiomas) que se derivan de dicha frase. Por tanto, con este mecanismo, se ofrece la posibilidad a los "usuarios no expertos" de introducir su consulta a partir de texto en lenguaje natural. Esta consulta es, posteriormente, procesada por la herramienta KAText, transformándola en una ontología con el objetivo del usuario. Una vez obtenida la ontología, el sistema comienza el proceso de cumplimiento del objetivo.

La ontología del objetivo sigue fielmente el modelo definido en la Fig. 82 (pag. 175). Este modelo consta de un conjunto de salidas y un conjunto de entradas. Las salidas representan lo que el usuario desea obtener como resultado de su interacción con el sistema, mientras que las entradas son los datos que proporciona el usuario para restringir el conjunto de posibles elementos de salida. En la Fig. 107, se representa gráficamente este modelo de objetivo que se utiliza en la mayor parte de las fases de que consta el proceso que el sistema lleva a cabo para la consecución de un objetivo.

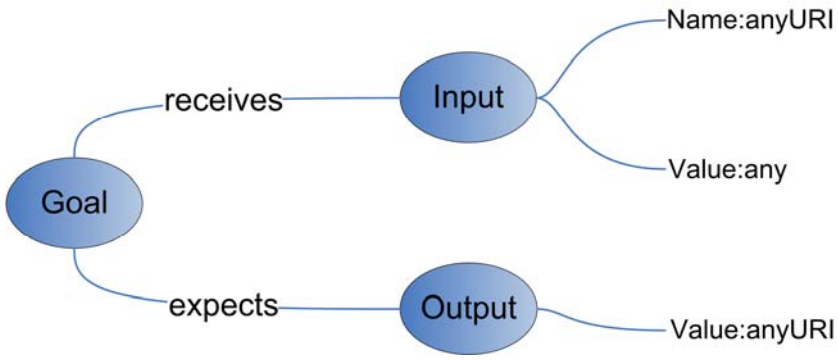


Fig. 107. Representación gráfica del modelo ontológico de objetivo

V.3.2.2. Descubrimiento y composición

Tal y como se especifica en el capítulo III, el descubrimiento y la composición son dos tareas estrechamente ligadas. Es una realidad generalmente aceptada que la composición sólo tiene lugar en caso de que el descubrimiento no tenga éxito. De este modo, y con el objetivo de que el flujo de trabajo fluya apropiadamente entre ambos procesos, tanto la composición como el descubrimiento son parte de un único agente, el '*Discovery Agent*'. Concretamente, estos procesos se llevan a cabo dentro de un mismo comportamiento, el '*Matchmaker Behaviour*', que, como se puede apreciar en la Tabla 11, es un comportamiento que debe incorporar obligatoriamente el '*Discovery Agent*'. Por tanto, como se puede observar en la Tabla 10, entre los roles básicos que asume el comportamiento '*Matchmaker Behaviour*', se encuentran el '*Matchmaker*', responsable del descubrimiento, y el '*Composer*', encargado de la composición.

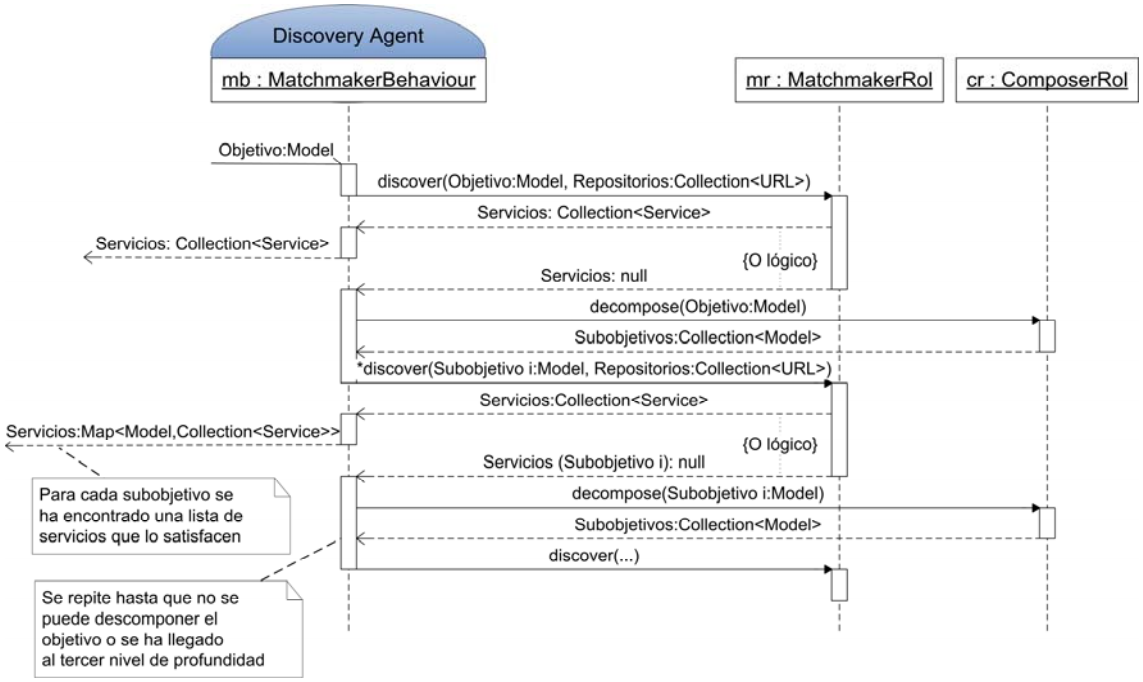


Fig. 108. Diagrama de secuencia para las tareas de descubrimiento y composición

En la Fig. 108, se presenta el diagrama de secuencia que muestra el mecanismo que interconecta las tareas de descubrimiento y composición. En la aplicación software desarrollada en esta tesis doctoral, la composición de servicios toma forma mediante la descomposición de servicios. La idea principal detrás de esta técnica es que si, para cada subobjetivo derivado de la descomposición del objetivo original, se ejecuta un servicio que lo satisface, entonces se puede afirmar que se ha alcanzado el objetivo inicial (ver Fig. 109). Por tanto, como se puede apreciar en la figura, en el caso en que el rol '*Matchmaker*' no sea capaz de encontrar, en ninguno de los repositorios conocidos, servicios que puedan cumplir lo que se pretende como objetivo, entonces el '*Matchmaker Behaviour*' tiene que emplear el rol '*Composer*' para descomponer dicho objetivo en una lista de sus subobjetivos constituyentes. Es importante destacar que, en este proceso de descomposición y búsqueda de servicios para los subobjetivos, sólo se toman en consideración divisiones del objetivo en las que todos y cada uno de los subobjetivos se tienen que satisfacer para que el objetivo original se pueda entender como alcanzado. En otras palabras, el proceso de descomposición genera como resultado un árbol Y/O en el que todos los enlaces son de tipo Y, sin considerarse, por el

momento, la existencia de enlaces O (esto es, enlaces que representan problemas que pueden resolverse de varias maneras alternativas).

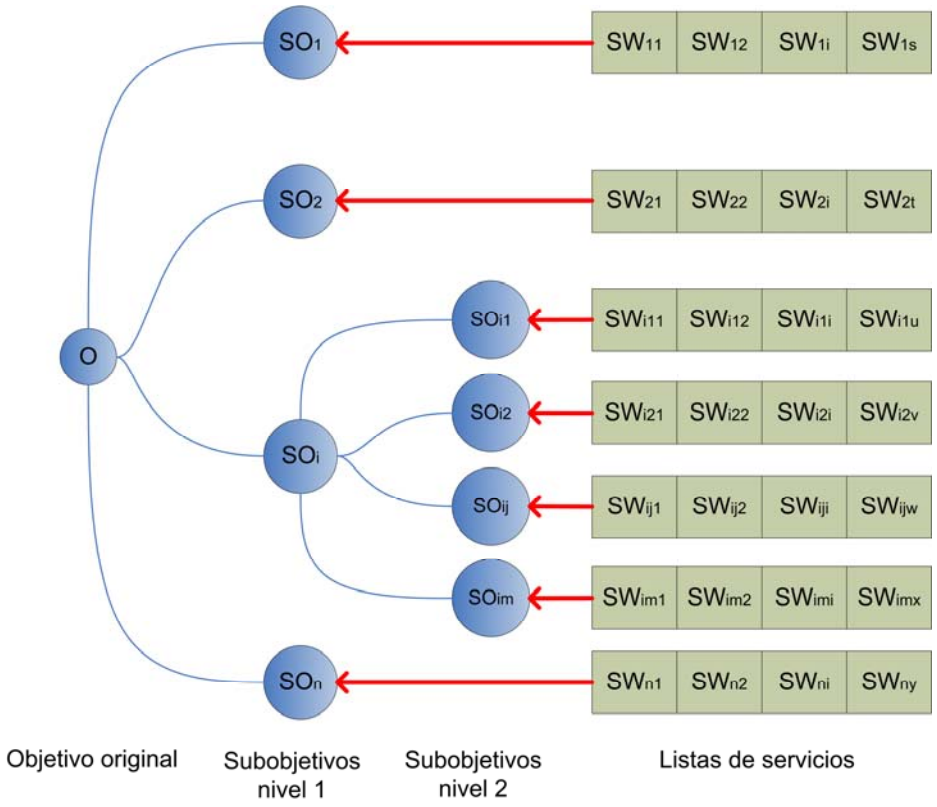


Fig. 109. Descomposición de objetivos y descubrimiento de servicios

Como parte del desarrollo de la aplicación software, se ha realizado una implementación básica de los roles '*Matchmaker*' y '*Composer*'. El proceso que se realiza por parte del '*Matchmaker*' está dividido en dos etapas principales. La primera etapa consiste en obtener, a partir del objetivo, lo que el usuario desea obtener como resultado de su consulta. Como se indica en la sección anterior, esta información se encuentra en la parte de '*Outputs*' del modelo ontológico de los objetivos (ver Fig. 107). Tal y como se aprecia en el diagrama de secuencia de la Fig. 108, el objetivo llega al '*Matchmaker Behaviour*' en forma de un '*Model*' (o sea, ontología según el API de Jena). Por tanto, para obtener la información requerida, en esta primera etapa se realiza una consulta en SPARQL, como la que se presenta en la Fig. 110, sobre la ontología del objetivo representada conforme al API de Jena. En la segunda etapa de proceso de descubrimiento, se debe interactuar con distintos tipos de repositorios de ontologías y

distintos modelos ontológicos de servicios. Como consecuencia, esta tarea se lleva a cabo dentro del rol '*Ontology Manager*', a través del método '*getService*'. Este método, a partir de otra consulta SPARQL (ver Fig. 111), esta vez sobre el repositorio Sesame donde se encuentran almacenadas las descripciones semánticas de los servicios, devuelve una lista con la URI de todos los servicios cuya salida coincide con el/los elemento/s '*Output*' del objetivo.

```
PREFIX table:<http://klt.inf.um.es:8080/services/goal.owl#>
SELECT ?y
WHERE {?x table:outputValue ?y}
```

Fig. 110. Consulta SPARQL para obtener los elementos '*Output*' de un objetivo[35]

```
PREFIX
s:<http://klt.inf.um.es:8080/services/description/getProtein.owl#>
PREFIX service:<http://www.daml.org/services/owl-s/1.2/Service.owl#>
PREFIX process:<http://www.daml.org/services/owl-s/1.2/Process.owl#>
SELECT ?type
WHERE {?return process:parameterType ?type . ?z process:hasOutput
?return . s:getProteinService service:describedBy ?z}
```

Fig. 111. Consulta SPARQL de ejemplo para obtener los elementos que se producen como salida de un servicio

La composición, por su parte, también está compuesta por varias fases. En la primera fase, si un objetivo tiene relaciones con diversos conceptos '*Output*', se descompone el objetivo en tantos subobjetivos como elementos '*Output*' tienen relación con el objetivo original, donde cada uno de ellos contiene una única relación con un concepto '*Output*' diferente. La segunda fase, que se inicia cuando el objetivo a descomponer sólo tiene una relación a un elemento '*Output*', consiste en utilizar un planificador al estilo de STRIPS. El planificador, que ha sido descrito en la sección V.2.2.5, es capaz de generar, a partir de un estado inicial vacio, un conjunto de elementos de salida (operadores) que, combinándose, dan lugar al objetivo, consistente en el resultado esperado por la consulta del usuario (o sea, el elemento '*Output*'). Un ejemplo típico que permite aclarar este procedimiento es el que parte de un objetivo como "*Asistir a un congreso*", y concluye en una descomposición en subobjetivos como la siguiente: "*Comprar billete de avión del lugar origen al lugar destino para antes de la fecha de inicio del congreso*", "*Reservar hotel cercano al lugar donde se celebre el congreso en las fechas en que tenga lugar el congreso*" y

[35] El modelo ontológico de objetivo, mostrado en Fig. 82 (pag. 176), se encuentra disponible desde la dirección http://klt.inf.um.es:8080/services/goal.owl

"*Comprar billete de avión del lugar destino al lugar origen para después de la fecha de fin del congreso*".

V.3.2.3. Selección

Durante la tarea de selección, se realiza una ordenación total o parcial de los servicios, que pueden cumplir los requisitos de un objetivo, que han sido descubiertos. En esta fase del procesamiento de la consulta de un usuario, el sistema utiliza conceptos y técnicas provenientes de los Sistemas de Ayuda a la Decisión y la Teoría de la Utilidad. Así, tal y como se indicó anteriormente, la selección se basa en modelos de preferencias definidos tanto para usuarios consumidores de servicios como para proveedores, y en las propiedades no funcionales de las descripciones semánticas de los servicios (si existen). Se distinguen cuatro modelos principales: (i) modelo de preferencias generales del consumidor, (ii) modelo de preferencias para una consulta, (iii) modelo de reglas estratégicas organizacionales, y (iv) modelo de condiciones particulares del servicio. Un usuario consumidor debe definir los dos primeros modelos: el primero, que se tendrá en cuenta para cada consulta que realice este usuario, durante su registro en la plataforma; y el segundo, en el momento de realizar una consulta. De forma similar, un usuario proveedor de servicios define las preferencias estratégicas en el momento del registro en el sistema y, si lo considera necesario, puede asociar a los servicios que proporciona un conjunto de condiciones particulares que no hayan podido ser expresadas como propiedades no funcionales en las descripciones semánticas de los mismos.

La aplicación software implementada incorpora un modelo básico para cada tipo de preferencias. Las preferencias se representan como ficheros XML, de forma que, para cada modelo, la estructura y sintaxis que deben cumplir los ficheros se han definido mediante documentos DTD ('*Document Type Definition*') –modelos de preferencias más complejos podrían requerir la definición mediante XML Schema, una recomendación del W3C que posee ciertas ventajas sobre DTD. En las figuras Fig. 112 y Fig. 113, se muestran los esquemas de los distintos tipos de modelos de preferencias. El modelo de preferencia general del consumidor consiste, básicamente, en el tipo de pago admitido (Visa o Mastercard). Entre los elementos del modelo de preferencia de la consulta, se encuentran la garantía que se espera sobre el servicio y el tiempo de entrega

de los resultados de dicho servicio (valores máximo y mínimo). Todos los elementos de los modelos de preferencias de los consumidores tienen asociados unos atributos que indican el factor de impacto que debe tener el elemento en cuestión a la hora de generar el valor de utilidad del servicio, y la obligatoriedad del elemento, es decir, si el elemento constituye una condición de obligado cumplimiento (si no se cumple, el servicio debe ser descartado) o no. Por su parte, el modelo de preferencias estratégicas del proveedor incorpora un elemento que identifica los métodos de pago aceptados, y, entre las preferencias del servicio, se incluyen el tiempo de reparto (máximo, mínimo y promedio) y la garantía que se ofrece en los resultados del servicio.

```
<!DOCTYPE consumer [

<!ELEMENT generalPreferences (payment+)>
<!ELEMENT payment>
<!ATTLIST payment
      type ( Visa | Mastercard ) #REQUIRED
      impact (1|2|3|3|4|5|6|7|8|9|10) #REQUIRED
      mandatory ( yes | no) #REQUIRED>

<!ELEMENT queryPreferences (warranty?) (deliveryTime?)>
<!ELEMENT warranty (#PCDATA)>
<!ATTLIST warranty
      impact (1|2|3|3|4|5|6|7|8|9|10) #REQUIRED
      mandatory ( yes | no) #REQUIRED>
<!ELEMENT deliveryTime>
<!ATTLIST deliveryTime
      max CDATA #REQUIRED
      min CDATA #REQUIRED
      impact (1|2|3|3|4|5|6|7|8|9|10) #REQUIRED
      mandatory ( yes | no) #REQUIRED>
]>
```

Fig. 112. Esquema DTD de los modelos de preferencia de los consumidores

```
<!DOCTYPE provider [

<!ELEMENT strategicPreferences (payment+)>
<!ELEMENT payment>
<!ATTLIST payment
      type ( Visa | Mastercard ) #REQUIRED

<!ELEMENT servicePreferences (deliveryTime) (warranty)>
<!ELEMENT deliveryTime>
<!ATTLIST deliveryTime
      max CDATA #REQUIRED
      avg CDATA #REQUIRED
      min CDATA #REQUIRED>
<!ELEMENT warranty (#PCDATA)>
]>
```

Fig. 113. Esquema DTD de los modelos de preferencia de los proveedores

En base a las preferencias establecidas por usuarios consumidores y proveedores de acuerdo con los modelos mostrados y a las propiedades no funcionales, se calcula un *valor de utilidad* para cada uno de los servicios. Existen dos casos posibles, que no se haya producido composición de servicios y, por tanto, sólo se encuentre una lista de servicio que cumplen el objetivo original (ver Fig. 114-a), o que, debido a la descomposición del objetivo en numerosos subobjetivos, haya una lista de servicios para cada subobjetivo (ver Fig. 114-b). En ambos casos, el procedimiento de selección es similar, con la única excepción de que si se ha dividido el objetivo en subobjetivos, el proceso de ordenación se debe repetir tantas veces como listas de servicios existan.

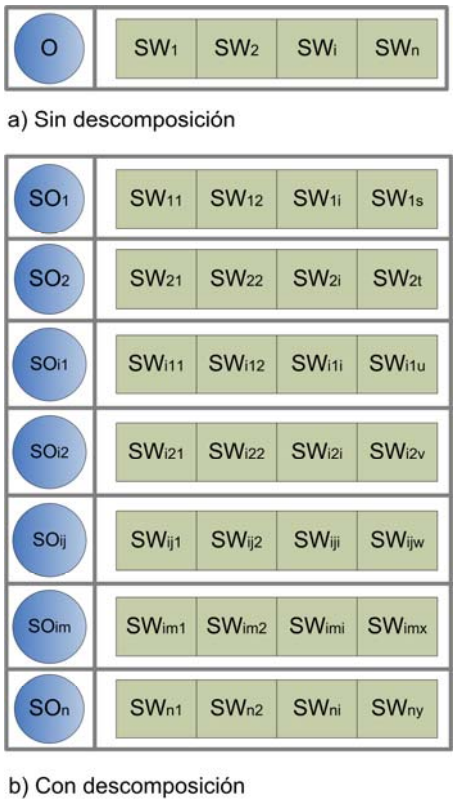


Fig. 114. Lista de servicios asociadas a los (sub)objetivos[36]

La solución para producir los valores de utilidad de cada servicio con respecto a las preferencias de consumidores y proveedores es, al igual que los modelos de preferencias

[36] La descomposición está basada en la mostrada en la Fig. 109.

en sí, bastante rudimentaria. Ambos elementos, el algoritmo para la ordenación de los servicios, y los modelos para los distintos tipos de preferencias pueden ser redefinidos por los desarrolladores para ajustarlos a los requisitos particulares de la aplicación que tengan que desarrollar. En la Fig. 115, se presenta el mecanismo implementado como parte del rol '*Selector*' que genera el valor de utilidad de un servicio en base a la información del propio servicio (esto es, preferencias estratégicas del proveedor del servicios y condiciones particulares del servicio) y las preferencias establecidas por el usuario consumidor.

```
int utility(consumer, service){
   int value = 0;
   //Payment type
   for (String c_type: consumer.paymentType){
      boolean found = false;
      for (String p_type: service.paymentType){
         if c_type.equalsIgnoreCase(p_type){
            value += consumer.paymentImpact;
            found = true;
         }
      }
      if (!found & consumer.mandatoryPayment)
         return 0;
   }
   // Warranty Time
   value += (service.warranty – consumer.warranty) *
             consumer.warrantyImpact;
   // Delivery Time
   Value += (consumer.delivery – service.delivery) *
             consumer.deliveryImpact;
   return value;
}
```

Fig. 115. Cálculo del valor de utilidad

Sin embargo, antes de realizar la valoración de las condiciones impuestas por los proveedores de servicios para la ejecución de los servicios que éstos suministran, es necesario llevar a cabo un proceso de negociación con el que se determinen cuáles son, efectivamente, estas condiciones. Con este propósito, el '*Selection Agent*' comienza una comunicación con todos los agentes '*Service Agent*' involucrados en la consulta actual. Para que esto pueda llevarse a cabo con éxito, es necesario, en primer lugar, que tanto los agentes '*Service Agent*' como los '*Provider Agent*' correspondientes sean instanciados dentro del entorno multi-agente. Con este propósito, en una primera fase, el '*Selection Agent*' solicita al '*Framework Agent*' la creación de tales agentes. El '*Framework Agent*', entonces, evalúa el estado del sistema en relación a los recursos

disponibles actualmente, y determina si es posible instanciar los agentes solicitados. En caso afirmativo, una vez que se hayan creado todos los agentes pertinentes, se informa al '*Selection Agent*' de que ya puede comenzar a interactuar con los agentes. Esta primera fase de que consta la tarea de selección, se muestra gráficamente, mediante un diagrama de secuencia, en la Fig. 116.

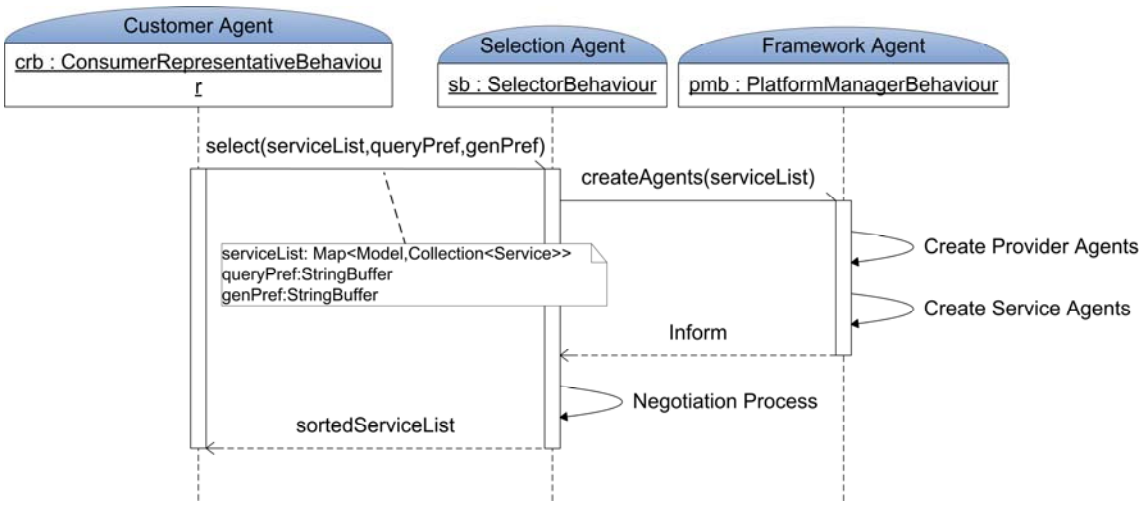


Fig. 116. 1ª Fase Selección: creación de los agentes implicados

La segunda fase que se realiza en la tarea de selección es la negociación. El '*Selection Agent*', una vez identificados todos los agentes '*Service Agent*' con los que tiene que interactuar, envía un "*Call for proposals*" (*CFP*) a éstos, con la intención de pedir propuestas para la ejecución de los servicios que representan. Los '*Service Agent*', entonces, se ponen en contacto con los agentes '*Provider Agent*' correspondientes para disponer de las preferencias estratégicas de los proveedores representados, y poder hacer uso de las mismas, junto con las preferencias particulares de los servicios, para generar las propuestas (*Proposal*) a enviar al '*Selection Agent*'. Cuando el '*Selection Agent*' ha recibido todas las propuestas, las evalúa conforme al mecanismo descrito anteriormente por medio del rol '*Selector*', y devuelve al '*Customer Agent*' la lista ordenada de servicios. En la Fig. 117, se representa gráficamente todo este proceso mediante un diagrama de secuencia.

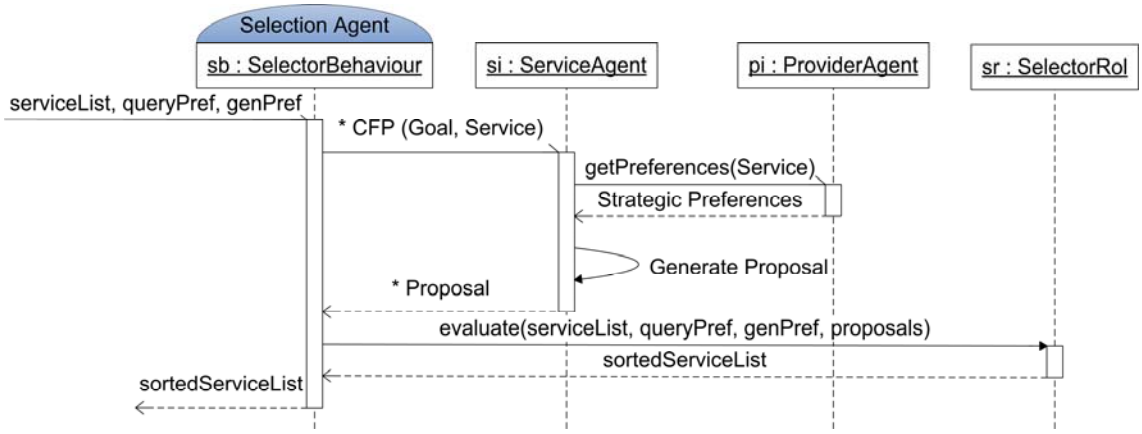


Fig. 117. 2ª Fase Selección: negociación y evaluación de propuestas

V.3.2.4. Invocación

Una de las últimas fases en el procesamiento de una consulta de usuario es la ejecución de los servicios que satisfacen las necesidades requeridas. La librería Axis de Apache ha sido utilizada para esta tarea (algunos detalles de esta librería se describen en la sección V.2.2.3). El proceso comienza después de que el usuario haya decidido los servicios que desea ejecutar. El '*Customer Agent*' recibe la lista con los servicios elegidos y comienza la interacción con los distintos agentes '*Service Agent*' que representan a los servicios en cuestión. Una vez que una petición de invocación llega al '*Service Agent*', éste emprende la tarea de ejecución por medio de su comportamiento '*Invoker Behaviour*' y, más concretamente, el rol '*Invoker*'. Los pasos a seguir para llevar a cabo la invocación del servicio son los siguientes:

1. Obtención de los parámetros de entrada para la ejecución del servicio a partir de la descripción semántica del mismo. Esto se realiza mediante una simple consulta SPARQL sobre el modelo que representa la descripción semántica del servicio en cuestión. En la Fig. 118, se muestra una consulta de ejemplo que permite determinar los parámetros de entrada de un método particular que se encuentran representados en la parte del '*Grounding*' de una descripción en OWL-S.
2. Obtención de los valores de cada uno de los parámetros identificados en el paso anterior. Los valores se recuperan, si se encuentran disponibles, de la parte de '*Input*' del objetivo indicado por el usuario, cuyo modelo ontológico (ver Fig. 107, pag. 248) ha sido enviado al '*Service Agent*' por parte del '*Customer Agent*'.

Para esto, se obtienen todos los elementos '*Input*' del objetivo (ver Fig. 119) y se compara uno tras otro con los parámetros. Si se ha utilizado la misma ontología tanto para describir el servicio, como para definir el objetivo, la correspondencia entre la descripción de los parámetros de entrada del servicio y los elementos que constituyen el '*Input*' del objetivo será perfecta. En otro caso, se deberá hacer uso de las reglas de mapeo para determinar si, efectivamente, un elemento '*Input*' del objetivo se corresponde con un parámetro solicitado para la ejecución del servicio.

```
PREFIX grounding:<http://www.daml.org/services/owl-
s/1.2/Grounding.owl#>
PREFIX
z:<http://klt.inf.um.es:8080/services/description/getProtein.owl#>
SELECT ?parameter
WHERE {?grounding grounding:owlsProcess z:getProteinProcess .
?grounding grounding:wsdlInput ?input . ?input
grounding:wsdlMessagePart ?parameter}
```

Fig. 118. Consulta SPARQL de ejemplo para obtener los parámetros de entrada de un servicio

```
PREFIX table:<http://klt.inf.um.es:8080/services/goal.owl#>
SELECT ?y ?z
WHERE " +
{?x table:inputValue ?y . ?x table:inputName ?z}
```

Fig. 119. Consulta SPARQL para obtener los nombres y valores de los elementos '*Input*' de un objetivo

3. Si todos los parámetros han sido cubiertos por la información procedente del objetivo, se realiza la invocación del servicio (toda la información necesaria se extrae, al igual que los parámetros, de la sección '*Grounding*' de la descripción semántica del servicio) y, una vez éste ha devuelto los resultados, éstos se remiten al '*Customer Agent*'. Cuando el '*Customer Agent*' recibe correctamente los resultados de todos los servicios que debía ejecutar, los combina y se los presenta al usuario como respuesta a la consulta realizada por el mismo.
4. En el caso en que el valor de alguno de los parámetros necesarios para la ejecución del servicio no haya podido determinarse, el rol '*Invoker*' remite el error al '*Invoker Behaviour*' (del '*Service Agent*'), que solicita inmediatamente al '*Customer Agent*' este tipo de información para poder concluir con éxito la invocación del servicio.
5. Si el '*Customer Agent*' recibe un informe de error solicitando información no recibida de alguno de los servicios que ha intentado ejecutar, comienza un proceso

de búsqueda de esta información en las distintas fuentes de datos a las que tiene acceso, referentes al usuario al que representa. En particular, intenta determinar si los datos solicitados se encuentran entre las preferencias, generales o de la consulta, que había establecido el usuario. Además, espera la respuesta del resto de servicios que ha mandado invocar, si se da el caso, para determinar si la información no recibida se encuentra entre los resultados de alguno de los otros servicios ejecutados.

6. Si, finalmente, el '*Customer Agent*' ha sido capaz de determinar los valores de los parámetros restantes de los servicios, ya sea a partir de las preferencias del usuario o a partir de los resultados de los servicios que se han podido ejecutar con éxito, se envía esta información a los agentes '*Service Agent*' correspondientes y se llega al paso 3. En caso contrario, el '*Customer Agent*' solicita estos datos al usuario final, quien, introduciendo los elementos restantes, permite al sistema que se concluya con éxito el procesamiento de su consulta.

En la Fig. 120, se presenta el diagrama de secuencia que representa gráficamente estos pasos.

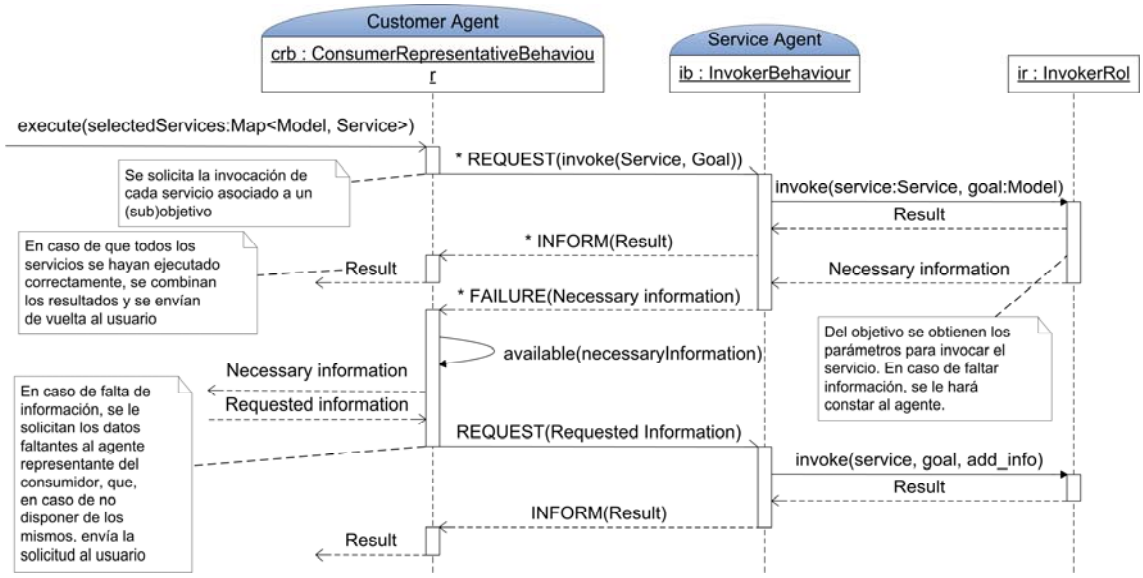


Fig. 120. Diagrama de secuencia para la tarea de invocación

V.3.3. Arquitectura de la aplicación Web

Por último, como complemento al SMA y con la finalidad de permitir la interacción con los distintos usuarios externos al sistema, se ha desarrollado una aplicación Web. Haciendo uso de esta aplicación, los usuarios, independientemente de su cual sea su perfil (desarrollador, consumidor de servicios o proveedor de servicios), con los únicos requisitos de tener acceso a Internet y disponer de un navegador Web en su equipo de sobremesa, pueden sacar provecho de toda la funcionalidad ofrecida por la plataforma.

Como se resaltó anteriormente, para el desarrollo de la aplicación Web se emplearon las tecnologías de Java en torno a JSP y Servlet. Además, el diseño de la aplicación se ha realizado siguiendo el patrón STRUTS. El propósito fundamental de este patrón es conseguir separar claramente la lógica de negocio de la lógica de presentación en aplicaciones Web desarrolladas utilizando Servlets y páginas JSP. Para esto, el patrón propone realizar todo lo concerniente a los procesos de negocio por medio de clases Java albergadas en el servidor, mientras que las páginas JSP sólo se encargan de presentar los resultados del modo deseado. Por otra parte, con este mecanismo se consigue diferenciar claramente los roles de desarrollador de aplicaciones y de diseñador Web.

Con arreglo al patrón STRUTS, la aplicación Web consta de un único Servlet, denominado '*FrontController*', que es el responsable de evaluar todas las peticiones que le llegan a la aplicación. Para procesar una petición particular, el Servlet hace uso de alguna de las *acciones* disponibles en el sistema (existe un fichero que establece una relación unívoca entre peticiones y acciones). Entonces, la *acción* se encarga de realizar todo el procesamiento que concierne a la lógica de negocio y, finalmente, devuelve los resultados que deberán ser mostrados como respuesta a la petición original. Los componentes encargados de utilizar los resultados devueltos por las *acciones* para generar la presentación, son las páginas JSP. Con esto, se obtiene una aplicación Web que cumple con el esquema de diseño Modelo-Vista-Controlador. El modelo lo conforman las distintas *acciones*, la vista la constituyen las páginas JSP y el navegador Web, y el controlador está implementado en el Servlet '*FrontController*'. En la Fig. 121, se muestra gráficamente la estructura general de la aplicación Web.

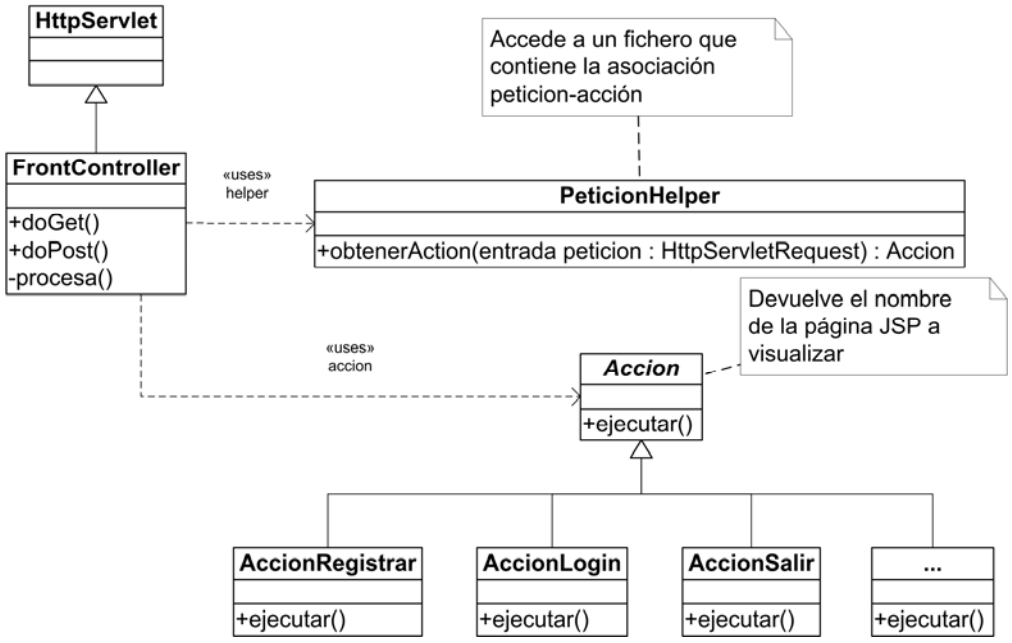


Fig. 121. Estructura de la aplicación Web conforme al patrón Struts

El proceso comienza cuando un usuario realiza una determinada acción sobre la interfaz gráfica que genera un evento que provoca el envío de un mensaje al servidor Web. Este mensaje, denominado *petición*, es recibido por el '*FrontController*', Servlet que se encarga de la recepción de todos los mensajes que llegan de la interfaz. La primera tarea del '*FrontController*' es determinar qué acción es necesario llevar a cabo para procesar la petición en cuestión. Para esto, hace uso del '*PetitionHelper*', que, accediendo a un fichero donde se encuentran disponibles las asociaciones entre los eventos de la interfaz gráfica y las acciones a tomar, devuelve una instancia de la clase acción pertinente. A continuación, el '*FrontController*' ejecuta la acción pasándole como parámetros la petición (*HttpServletRequest*), el elemento donde el navegador espera encontrar la respuesta (*HttpServletResponse*), y el contexto de la aplicación (*ServletContext*). Las clases *acción*, por su parte, se encargan del acceso a la base de datos y de la comunicación con los agentes de la plataforma. Una vez procesada la petición y obtenido el resultado a mostrar al usuario, la clase *accion* asigna valores a ciertos parámetros y devuelve la *página JSP*, a partir de la cual se tiene que generar el documento HTML a ser presentado al usuario. Para generar este documento, la *página JSP* hace uso de los parámetros establecidos por la clase *acción*.

La propia aplicación Web es quien lleva a cabo la gestión de los usuarios (proveedores y consumidores de servicios) y sus preferencias. Cuando un usuario se registra en el sistema, ya sea como entidad proveedora de servicios o como cliente, sus datos, así como sus preferencias, son almacenados en una base de datos. Actualmente, la aplicación hace uso, como se comentó anteriormente, del sistema gestor de base de datos MySQL. Sin embargo, y con la intención de satisfacer uno de los objetivos clave establecidos en el desarrollo de esta tesis doctoral, a saber, máxima flexibilidad, el diseño del mecanismo de acceso a la fuente de datos se ha elaborado en base al patrón DAO ('*Data Access Object*'). DAO es un patrón de acceso a datos que permite almacenar y recuperar información persistente de distintas fuentes (p.ej., bases de datos relacionales, LDAP, XML, etc.). Entre los beneficios que aporta esta solución, se incluyen los siguientes: favorecer la transparencia, facilitar la migración de los componentes, reducir la complejidad y proporcionar un mecanismo de acceso a datos centralizado en una capa. No obstante, esta solución también conlleva ciertos inconvenientes, como la necesidad de diseñar una jerarquía de clases (provocando una explosión de clases) y la introducción de una nueva capa en la infraestructura.

En la Fig. 122, se muestra gráficamente la estructura de clases que supone la implementación del patrón DAO. En dicha figura, se comprueba, además, la sencillez con la que se puede incorporar el soporte a un nuevo tipo de fuente de datos. En la figura, a modo de ejemplo, se presenta el caso de XML.

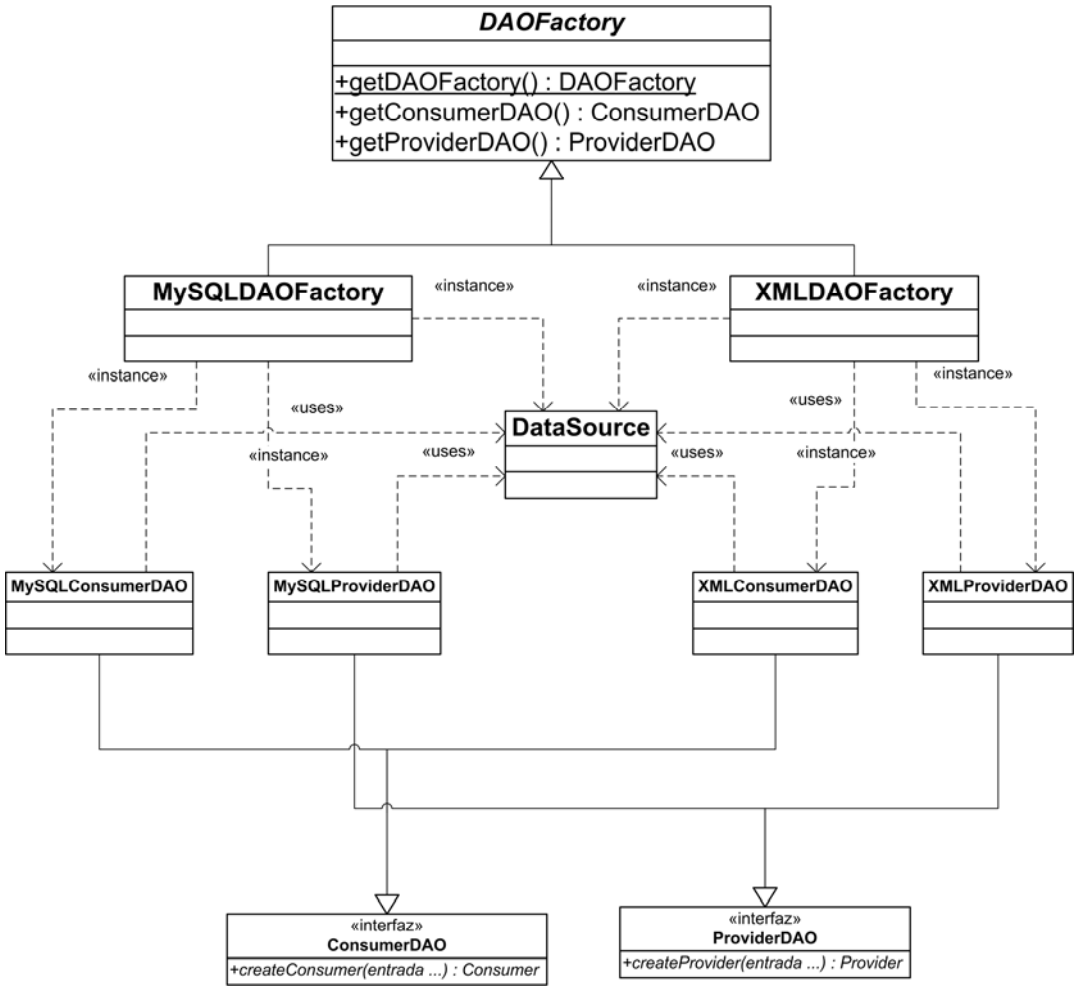


Fig. 122. Implementación del patrón DAO

V.4. Uso de la Aplicación

En esta sección, se muestran las interfaces Web que ofrece la aplicación software desarrollada para los distintos tipos de usuarios: interfaz con el desarrollador, interfaz con el proveedor de servicios e interfaz con el consumidor de servicios. Para cada una de las interfaces, se presentan las distintas funciones disponibles mediante las cuales los usuarios pueden interactuar con la plataforma. Las funciones a las que un usuario tiene acceso dependen, directamente, del perfil que el usuario en cuestión haya tomado con respecto a la plataforma.

V.4.1. Interfaz con el Desarrollador

En este apartado, se enumeran las acciones que un desarrollador de aplicaciones puede llevar a cabo para ajustar el marco de trabajo genérico implementado a las condiciones particulares de una aplicación específica sobre un dominio determinado. Entre estas acciones, se encuentran: la adición y eliminación de fuentes de conocimiento y ontologías, la incorporación de nuevas implementaciones de los distintos tipos de roles identificados, y la instanciación de los agentes (y los roles que éstos asumen) que van a participar en la plataforma.

V.4.1.1. Gestión de repositorios: añadir y eliminar

La primera función que se les ofrece a los desarrolladores de aplicaciones es la adición y eliminación de repositorios de ontologías. Mediante la interfaz para la gestión de los repositorios de ontologías (ver Fig. 123), el desarrollador puede indicar la localización de las bases de conocimiento a las que la plataforma debe acceder para su correcto funcionamiento.

Los datos que el desarrollador tiene que aportar para cada repositorio que desee incorporar son los siguientes:

- *URL* donde se encuentra localizado el repositorio y que permite el acceso a través de Internet.
- Identificador (*ID*) de la base de conocimiento dentro del repositorio que es de interés para las tareas que debe de llevar a cabo la aplicación software.
- *Tipo* de repositorio (p. ej. Sesame, MySQL, YARS). La precisión de este dato es crucial, dado que el mecanismo para acceder a cada tipo de repositorio es diferente y, por tanto, la asociación de un repositorio a uno de los tipos identificados conlleva la restricción del rol '*Ontology Manager*' que puede utilizarse para acceder al mismo.

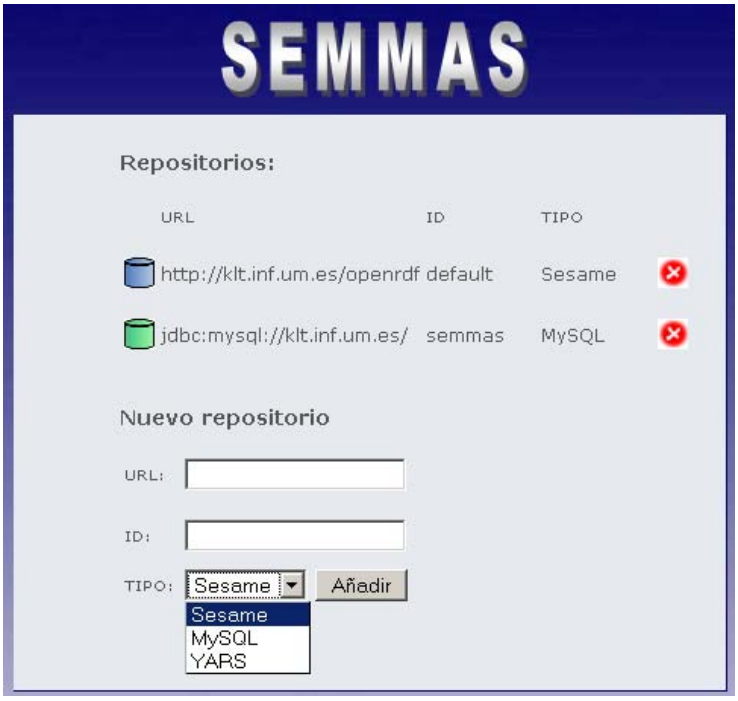


Fig. 123. Interfaz desarrollador – gestión de repositorios

La interfaz para la gestión de los repositorios presenta, en primer lugar, una lista de todos los repositorios que ya han sido registrados en la plataforma, permitiendo, en el caso de que el desarrollador así lo decida, la eliminación de los mismos. Por otra parte, en la parte baja de la ventana, es posible incluir los datos de los nuevos repositorios y sumarlos a los ya presentes en la plataforma.

V.4.1.2. Gestión de ontologías: añadir y eliminar

Uno de los mecanismos fundamentales de los que tiene disponibles el desarrollador para la adaptación del marco de trabajo a una aplicación y un dominio concreto, es la gestión de las ontologías que se van a utilizar internamente. La aplicación Web desarrollada para permitir a los usuarios externos la interacción con la plataforma, incorpora una interfaz que permite al desarrollador realizar tareas como la adición y eliminación de ontologías (ver Fig. 124).

Entre los datos que el desarrollador tiene que aportar para incorporar una nueva ontología al sistema, se incluyen los siguientes:

- *Tipo* de ontología. Como se indicó en la sección III.4.4.1, el desarrollador es el responsable de llevar la gestión de cuatro tipos de ontologías, a saber, ontología

del dominio, ontología de la aplicación, ontología de negociación, y ontología de reglas de mapeo.

- *Fichero* donde está contenida la ontología propiamente dicha.
- *Repositorio*, de entre los que tiene registrados la plataforma, donde va a ser almacenada la ontología.

Fig. 124. Interfaz desarrollador – gestión de ontologías

El sistema, además de guardar la ontología en el destino indicado, genera una entrada en la base de datos local donde se asocia cada ontología incluida con la localización donde ha sido almacenada, y el tipo de ontología de que se trata.

Al igual que en el caso anterior, la ventana de la interfaz para la gestión de las ontologías está dividida en dos secciones. En la sección superior, se muestra un listado de las ontologías que han sido registradas previamente en la plataforma y se ofrece la posibilidad de borrarlas individualmente. En la parte inferior, el desarrollador se encuentra con los campos que debe rellenar para añadir una nueva ontología a la plataforma.

V.4.1.3. Gestión de roles: implementación

El desarrollador también tiene la posibilidad de añadir, para los distintos tipos de roles identificados, nuevas implementaciones que permitan llevar a cabo las tareas básicas del sistema con mayor eficiencia y precisión. El desarrollador dispone de una interfaz para establecer las implementaciones de los roles a las que la aplicación va a tener acceso (ver Fig. 125).

Fig. 125. Interfaz desarrollador – gestión de roles

Para incorporar una nueva implementación de un tipo determinado de rol, el desarrollador debe aportar la siguiente información acerca de la misma:

- *Nombre* que se le ha dado a la implementación del rol.
- *Fichero* que contiene la clase Java compilada (*.class*) con la implementación del rol.
- *Tipo* de rol que implementa la clase (ésta tiene que cumplimentar los requisitos establecidos en el API genérico del tipo de rol indicado).

En sintonía con los contenidos en el resto de la aplicación Web, la interfaz para la gestión de los roles está compuesta por dos secciones. En la primera de las secciones, se muestra la lista de implementaciones de roles de las que la plataforma dispone en cada momento. En la parte inferior, se encuentra el formulario que debe rellenar el desarrollador con la información pertinente a la nueva implementación de rol que éste quiere incorporar al sistema.

V.4.1.4. Gestión de agentes: instanciación

El último mecanismo a disposición de los desarrolladores para ajustar el marco de trabajo a sus necesidades, es la instanciación de agentes. Si bien ciertos agentes, como los '*Customer Agent*', '*Provider Agent*' y '*Service Agent*', se crean en función de las necesidades del sistema en cada momento, existe un conjunto básico de agentes que constituyen el núcleo de la aplicación y que es necesario arrancar antes de poner en funcionamiento el sistema. En este sentido, el desarrollador es el responsable de decidir qué agentes instanciar y qué roles debe asumir cada uno de estos agentes instanciados. Con este propósito, se ha desarrollado una interfaz que permite al desarrollador, de una manera sencilla, gestionar los agentes que van a estar presentes en la plataforma, el tipo de los mismos, y los roles que éstos asumen (ver Fig. 126).

Los datos requeridos al desarrollador para la instanciación de un nuevo agente son los siguientes:

- *Nombre* por el que se va a identificar al agente en la plataforma. Este *nombre* debe ser único para cada agente del sistema.
- *Tipo* de agente a instanciar de entre los que conforman el núcleo de la plataforma (esto es, '*Broker Agent*', '*Discovery Agent*', '*Framework Agent*' y '*Selection Agent*'). Asimismo, es posible identificar distintas instancias de los otros tipos de agente ('*Customer Agent*', '*Provider Agent*' y '*Service Agent*'). Esta opción es particularmente interesante para el caso del '*Service Agent*' cuando se desea tratar, al mismo tiempo, con varias de las aproximaciones de Servicios Web Semánticos.
- *Roles* que se asocian al agente instanciado. Como se destacó anteriormente, el número y tipo de roles que asume un agente determina la funcionalidad que va a mostrar el mismo en la plataforma.

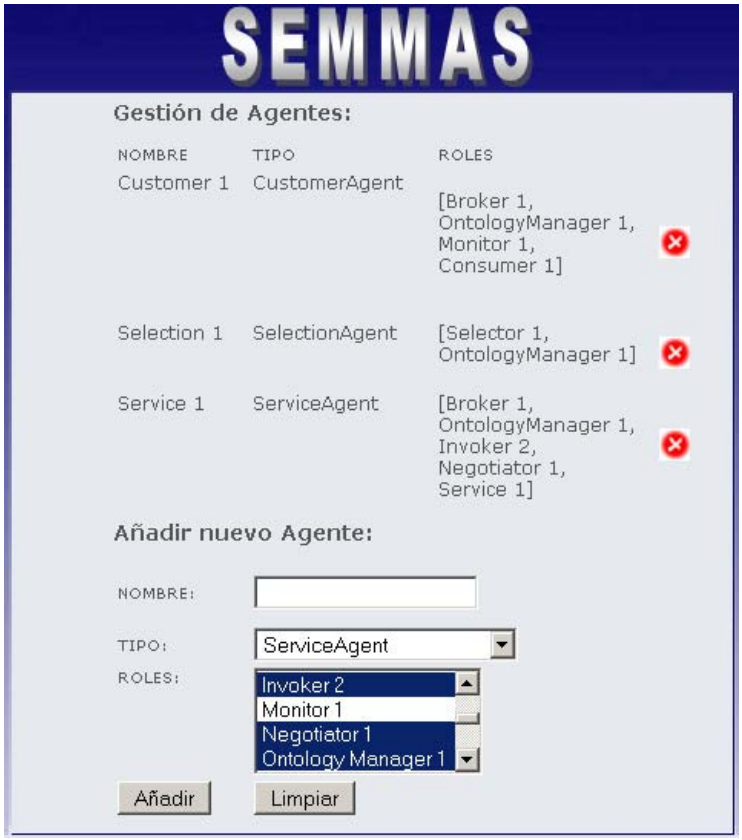


Fig. 126. Interfaz desarrollador – gestión de agentes

De igual forma que en los casos anteriores, la ventana de la interfaz está separada en dos partes bien diferenciadas. En la parte superior de la misma, se enumeran todos los agentes que han sido instanciados, así como su tipo y los roles que asumen. En la sección inferior, el desarrollador tiene a su disposición el formulario a través del cual puede indicar las características de un nuevo agente a instanciar.

V.4.2. Interfaz con el Usuario: Proveedor de Servicios

Como se destacó anteriormente, no es obligatorio que los proveedores de servicios se registren en la plataforma. Sin embargo, aquellos que lo consideren oportuno, se podrán beneficiar de las numerosas ventajas que conlleva el estar registrado. En particular, los proveedores de servicio, a través de la interfaz gráfica de la aplicación Web, tienen la posibilidad de, además de registrarse, gestionar los servicios que proporcionan y determinar las preferencias estratégicas (o sea, aplicables a todos los servicios suministrados) y particulares de cada servicio.

V.4.2.1. Gestión de proveedores: altas, bajas y modificaciones

Para acceder a las distintas posibilidades que ofrece la aplicación, un proveedor de servicios debe, en primer lugar, darse de alta en el sistema. Para esto, se ha elaborado un formulario que el proveedor debe cumplimentar con datos referentes a su identidad como entidad proveedora de servicios (ver Fig. 127). Los datos que debe aportar el proveedor son los siguientes:

- *Nombre* de la entidad. Debe corresponder con la razón social de la compañía proveedora de servicios.
- Código de Identificación Fiscal (*CIF*) de la entidad. Mediante este código se identifica de forma unívoca a cada entidad proveedora de servicios.
- La inclusión de la URL de la *página Web* de la organización es opcional.
- *Dirección de correo electrónico* del representante legal de la empresa o de la persona encargada de la interacción con la plataforma.
- *Contraseña* que le será solicitada al usuario, junto a su identificador (*CIF* en este caso), cada vez que intente acceder al sistema.
- *Perfil* o preferencias estratégicas que establece la organización y que deben de satisfacerse en todo momento para la ejecución de alguno de los servicios suministrados por la misma. Esta información constituye el nexo de unión entre la aplicación y las líneas estratégicas que persigue la organización para conseguir el éxito en sus operaciones. Las preferencias estratégicas, así como las particulares de cada servicio, son aplicadas en el momento de llevar a cabo un proceso de negociación con consumidores potenciales. Es necesario, además, que el fichero donde vienen definidas las preferencias estratégicas, se haya elaborado de acuerdo con el formato esperado por el sistema y definido mediante un documento DTD (ver Fig. 113, pag. 254).

Fig. 127. Interfaz proveedor – registro en el sistema

Una vez que un usuario se ha registrado en el sistema como proveedor de servicios, cada vez que quiera realizar alguna operación haciendo uso de este registro, debe acceder al sistema indicando su código de usuario (*CIF* de la organización) y la contraseña establecida. Para distinguir proveedores y consumidores, que acceden al sistema mediante la misma interfaz, se debe indicar también el tipo de usuario, en este caso '*Provider*'. En la Fig. 128, se muestra la interfaz gráfica para realizar tal operación.

Fig. 128. Interfaz proveedor – acceso al sistema

Por último, en lo que respecta a la gestión de los proveedores, el usuario tiene la posibilidad de modificar sus datos de registro como proveedor de servicios. Para esto, se ha elaborado una interfaz, muy similar a la utilizada para el registro de los proveedores, que permite examinar los datos de la entidad tal y como están registrados en la actualidad, y actualizar los mismos pertinentemente (a excepción del *CIF*, información que no se permite modificar) (ver Fig. 129). De este modo, los usuarios también podrán revisar los datos del perfil de la empresa, esto es, las preferencias estratégicas vigentes en ese momento.

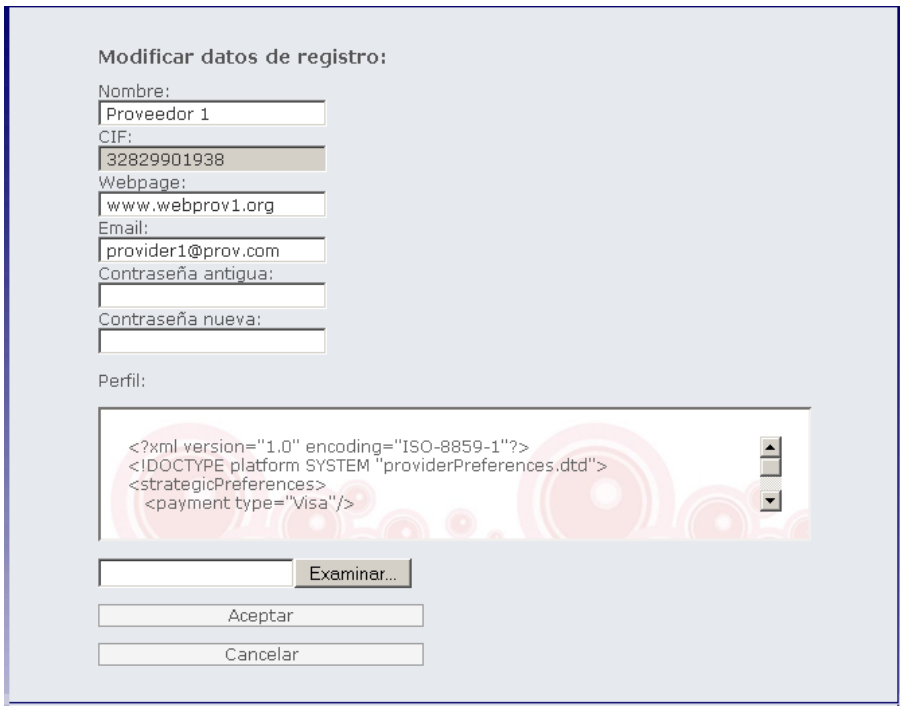


Fig. 129. Interfaz proveedor – modificación datos de registro

V.4.2.2. Gestión de servicios: altas, bajas y modificaciones

De forma adicional a la gestión de los datos de registro de la entidad, un proveedor de servicios puede indicar los servicios que desea poner a disposición del sistema para su consideración a la hora de satisfacer las necesidades de los usuarios. Con este propósito, en la página principal a la que tienen acceso los usuarios proveedores de servicios una vez han accedido al sistema a partir del código de usuario y la contraseña, se listan todos aquellos servicios suministrados por la entidad proveedora en cuestión. En la Fig.

130, se muestra la interfaz gráfica de la página principal de los proveedores con la lista de servicios que proporciona, y enlaces para eliminar los servicios individualmente ('*x*', a la izquierda de cada servicio), editar y actualizar los servicios ('+', a la derecha de cada servicio), añadir nuevos servicios, o modificar los datos de registro (mostrado anteriormente).

Fig. 130. Interfaz proveedor – página principal & listado de servicios aportados

Si se pulsa el botón '*Nuevo Servicio*' (Fig. 130), aparece una nueva pantalla que contiene el formulario que debe cumplimentar el proveedor para añadir un nuevo servicio al sistema (ver Fig. 131). Entre los datos que el proveedor tiene que aportar para incorporar un nuevo servicio a los ya disponibles en el sistema, se encuentran los siguientes:

- *Descriptor*, que identifica unívocamente al servicio entre todos los que suministra esta entidad proveedora.
- *Tipo* de la descripción semántica. En principio, se contemplan las cinco especificaciones de Servicios Web Semánticos vigentes (o sea OWL-S, WSMO, SWSF, WSDL-S y SAWSDL), que fueron discutidas en detalle en la sección I.4.3.1.

- *Descripción semántica* del servicio en cuestión de acuerdo con la especificación elegida previamente.
- *Preferencias* particulares aplicables al servicio que se añade. Al igual que para las preferencias estratégicas del proveedor, es necesario que el fichero donde vienen definidas las preferencias particulares del servicio se haya elaborado de acuerdo con el formato esperado por el sistema y definido mediante un documento DTD (ver Fig. 113, pag. 254).

Fig. 131. Interfaz proveedor – añadir nuevo servicio al sistema

Se ofrece al proveedor una posibilidad añadida que le permite añadir una gran cantidad de servicios a la plataforma de un modo rápido y efectivo. Para esto, el proveedor de servicios sólo debe indicar la URL del repositorio donde se encuentran almacenadas las descripciones semánticas de estos servicios.

Como se mencionó anteriormente, es posible editar la información que dispone el sistema de los servicios que ofrece el proveedor de un modo individual. De esta forma, es posible modificar tanto la descripción semántica que está asociada con el servicio, como las preferencias y condiciones que restringen su uso. Para este propósito, se ha elaborado una interfaz (ver Fig. 132) mediante la cual el usuario proveedor puede visualizar la descripción semántica del servicio y establecer una nueva descripción a través de un fichero, cambiar la aproximación a Servicios Web Semánticos que se está

utilizando, y cargar un nuevo fichero de preferencias particulares para el servicio. Al igual que se indicó anteriormente, es necesario que el fichero donde están definidas las preferencias particulares del servicio se haya elaborado de acuerdo con el formato esperado por el sistema y definido mediante un documento DTD (ver Fig. 113, pag. 254).

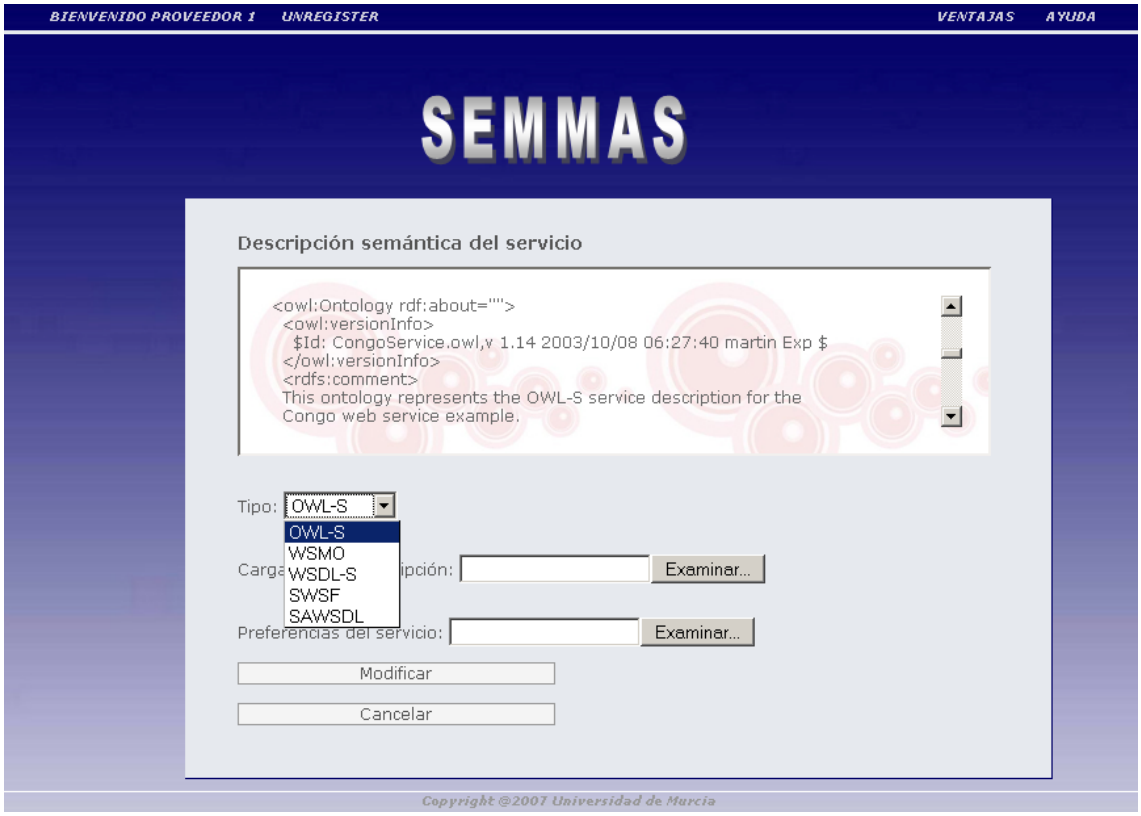


Fig. 132. Interfaz proveedor – modificación datos del servicio

V.4.3. Interfaz con el Usuario: Consumidor de Servicios

Los usuarios consumidores de servicios también tienen a su disposición un conjunto de interfaces para interaccionar con el sistema de un modo apropiado. Como sucede con los proveedores de servicios, no es obligatorio para los consumidores el registro en el sistema con el fin de poder utilizarlo (esto es, realizar consultas). Sin embargo, el usuario puede verse beneficiado de ciertas funciones de valor añadido en caso de registrarse.

Además de las altas y bajas en el sistema, un usuario consumidor de servicios puede realizar consultas para que sean resueltas por el sistema, seleccionar, de entre los servicios propuestos por el sistema para resolver la tarea requerida, uno o varios servicios para su ejecución, y, finalmente, el usuario recibe por pantalla el resultado obtenido de la ejecución del (de los) servicio (servicios).

V.4.3.1. Gestión de consumidores: altas, bajas y modificaciones

En este apartado, se examinan los mecanismos a disposición de un usuario, que se identifique como consumidor de servicios, para registrarse en la plataforma, modificar sus datos de registro, darse de baja en el sistema, y acceder a la aplicación por medio de un nombre de usuario y una contraseña que le identifiquen unívocamente en el sistema, pudiendo beneficiarse, así, de algunas características de "personalización" que ofrece la aplicación.

Un usuario que desee registrarse en la plataforma como cliente consumidor de servicios, es preciso que rellene un formulario con algunos de sus datos personales (ver Fig. 133). Entre la información a aportar por parte del usuario, se encuentra la siguiente:

- *Nombre* completo (nombre y apellidos) del usuario.
- *Dirección de correo electrónico* del usuario. Éste será el dato por el cual el usuario es identificado de manera unívoca en la plataforma.
- *Contraseña* que le será solicitada al usuario, junto a su identificador (*email* en este caso), cada vez que intente acceder al sistema.
- *Perfil* del usuario, que constituye las preferencias generales del consumidor y que son aplicadas cada vez que el usuario introduzca una consulta. Mediante este conjunto de preferencias generales (también las particulares que se indican en cada consulta), el sistema restringe el ámbito de la búsqueda de los servicios que pueden satisfacer las necesidades del usuario. Es necesario, además, que el fichero donde vienen definidas las preferencias generales se haya elaborado de acuerdo con el formato esperado por el sistema y definido mediante un documento DTD (ver Fig. 112, pag. 253).

Fig. 133. Interfaz consumidor – registro en el sistema

Una vez que un usuario se ha registrado en el sistema como consumidor de servicios, cada vez que desee realizar alguna operación haciendo uso de este registro, debe acceder al sistema indicando su código de usuario (*email* personal) y la contraseña establecida. Para distinguir proveedores y consumidores, que acceden al sistema mediante la misma interfaz, se debe indicar también el tipo de usuario, en este caso '*Consumer*'. En la Fig. 134, se muestra la interfaz gráfica para realizar tal operación.

Fig. 134. Interfaz consumidor – acceso al sistema

Por último, en lo que respecta a la gestión de la información personal de los consumidores, el usuario tiene la posibilidad de modificar sus datos de registro como

consumidor de servicios. Para esto, se ha elaborado una interfaz, muy similar a la utilizada para el registro de los consumidores, que permite examinar los datos del usuario tal y como están registrados en la actualidad, y actualizar los mismos pertinentemente (a excepción del *email*, información que no se permite modificar) (ver Fig. 135). De este modo, los usuarios también pueden revisar los datos del perfil de la empresa, esto es, las preferencias estratégicas vigentes en ese momento.

Fig. 135. Interfaz consumidor – modificación datos de registro

V.4.3.2. Gestión de consultas: tipos de entrada

El principal propósito de la aplicación software desarrollada en lo que respecta a los usuarios consumidores de servicios, es el de proporcionar servicios al usuario que permitan satisfacer las necesidades que los mismos puedan tener en un momento determinado. Para esto, es primordial disponer de un mecanismo efectivo que permita a los usuarios indicar el objetivo que persigue.

La aplicación software implementada, en concordancia con lo también establecido en el marco teórico, identifica distintos perfiles posibles para los usuarios consumidores de servicios. Si bien es posible que algunos de los usuarios que utilicen el sistema sean "expertos" y, por tanto, capaces de elaborar personalmente una ontología que represente el objetivo que persiguen conforme al modelo propuesto por la aplicación (ver Fig. 107, pag. 248), la mayoría no son capaces de llevar a cabo tal tarea. Así, se ha incorporado,

para su uso por parte de los "usuarios no expertos", una herramienta de reconocimiento de lenguaje natural que permite a estos usuarios interactuar con el sistema por medio de simples consultas a través de texto en lenguaje natural.

Con todo esto, la interfaz de usuario para la introducción de consultas se muestra en la Fig. 136. El formulario que debe rellenar el consumidor consta de los siguientes elementos:

- *Consulta en lenguaje natural* es el campo en el que el usuario puede introducir texto en lenguaje natural que contenga la necesidad que desea satisfacer en un momento dado. Es el primero de los tres mecanismos de los que los usuarios disponen para el establecimiento de los objetivos.
- Alternativamente a la consulta en lenguaje natural, un usuario experto puede indicar la *URL* donde se encuentra localizada la ontología que representa el *objetivo* del consumidor. Esta ontología, como se indicó anteriormente, debe basarse en el modelo ontológico estipulado por la aplicación (ver Fig. 82, pag. 175).
- La tercera opción para establecer el objetivo que debe perseguir el sistema es a través de un *fichero* almacenado en el equipo del usuario, que contiene la representación ontológica del objetivo (también de conforme al modelo ontológico de los objetivos).
- *Preferencias* particulares aplicables únicamente a la consulta actual. Al igual que las preferencias generales, las preferencias particulares de la consulta se aplican para restringir el número de servicios que pueden satisfacer la consulta en cuestión. Asimismo, tal y como sucede para el caso de las preferencias generales, es necesario que las preferencias particulares se elaboren en conformidad con el formato esperado por el sistema y definido mediante un documento DTD (ver Fig. 112, pag. 253).

Fig. 136. Interfaz consumidor – realizar consulta

V.4.3.3. Selección y ejecución de servicios

En general, cuando la aplicación procesa la consulta del usuario, encuentra numerosos servicios que satisfacen los requisitos del consumidor. Si bien sería posible que el sistema determinase, de forma automática, qué servicio ejecutar de acuerdo con las preferencias del usuario, se ha considerado más apropiado dejar la última decisión al propio usuario. La aplicación software, siguiendo las pautas de un sistema de ayuda a la decisión, calcula un valor de utilidad para cada servicio identificado, y utiliza este valor para generar una ordenación parcial de los mismos. La lista de servicios se muestra, entonces, al usuario en el orden en que el sistema lo ha determinado, quién deberá decidir, en base a la información de que se dispone, qué servicios ejecutar.

En la Fig. 137, se muestra la interfaz del usuario donde se presenta el listado de los servicios encontrados. En caso de que haya sido preciso descomponer el objetivo original para generar una composición de servicios válida que resuelva el objetivo en cuestión, los servicios se clasifican por los subobjetivos para los que son de interés. Además, situando el ratón sobre el botón '+' a la derecha de los servicios, se muestran

los detalles de ese servicio en particular. Una vez que el usuario ha decidido los servicios a ejecutar, los selecciona (elementos '*option*' a la izquierda de cada servicio), y pulsa el botón '*Aceptar*' para que el sistema proceda a su ejecución.

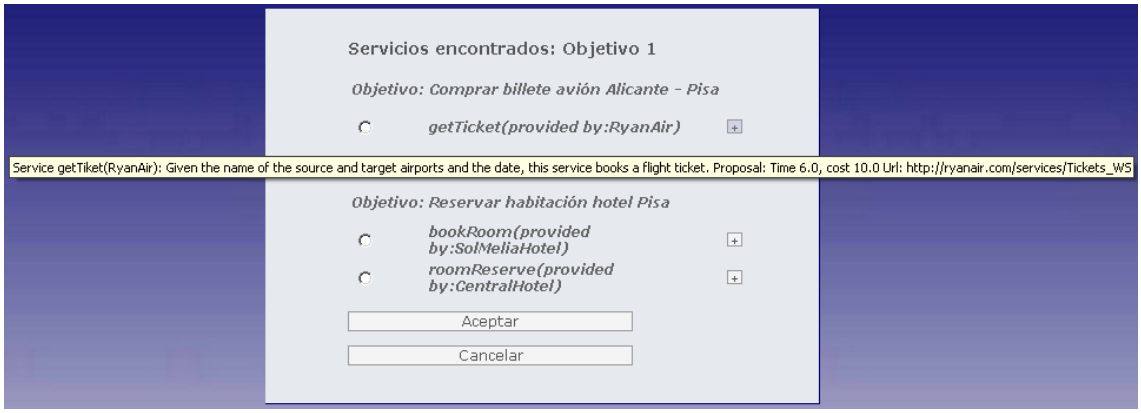


Fig. 137. Interfaz consumidor – listado de servicios por subobjetivos para su selección

V.4.3.4. Presentación de resultados

Cuando los servicios han sido ejecutados, y suponiendo que todo el proceso se ha producido satisfactoriamente y sin errores, los resultados son presentados al usuario. La interfaz diseñada para tal tarea es muy sencilla, ya que se basa en la impresión de las cadenas de texto que constituyen la respuesta del sistema (ver Fig. 138).

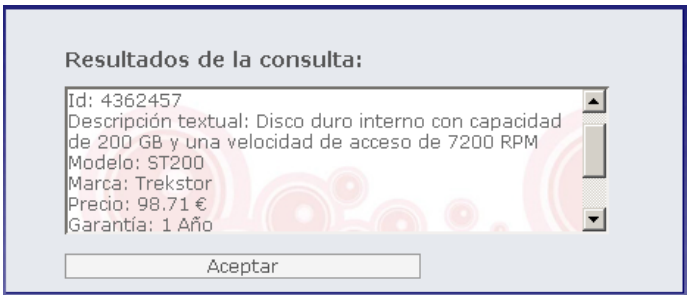


Fig. 138. Interfaz consumidor – presentación de resultados

V.4.3.5. Servicio Web

De forma complementaria a la aplicación Web, los usuarios consumidores de servicios disponen de un Servicio Web a través del cual pueden interactuar con el entorno multi-

agente. Este servicio permite que componentes software externos puedan actuar como consumidores de servicios.

Con el fin de limitar la complejidad en la interacción con el servicio y evitar coreografías confusas, se ha optado por implementar un Servicio Web 'sin estado'. El servicio, por tanto, recibe como entrada el objetivo del usuario (a partir de texto en lenguaje natural o con el modelo ontológico de Jena), y devuelve el resultado de la ejecución del servicio o servicios seleccionados. De este modo, si la interacción se produce a través del Servicio Web diseñado a tal efecto, es la aplicación software, y no el usuario, la que se encarga de seleccionar el servicio que se envía a ejecución.

A continuación, se muestran los métodos que constituyen el Servicio Web:

```
   Resultado ejecutar(String consulta, QueryPreferences qp,
ConsumerPreferences cp);
   Resultado ejecutar(Model consulta, QueryPreferences qp,
ConsumerPreferences cp);
```

El resultado está compuesto por un código de respuesta con el valor -1 si se ha producido un error y 0 en caso contrario, y una cadena de texto que contiene la descripción del error o la serialización de los resultados, respectivamente.

V.5. Resumen

En primer lugar, se han identificado las herramientas y librerías que han sido fundamentales para el desarrollo de la aplicación software objeto de descripción en este capítulo. Entre las herramientas, se destacan las que han servido para el desarrollo de la aplicación Web y su publicación a través de Internet, y las utilizadas para implementar el Sistema Multi-Agente conforme al diseño y especificación descritos en capítulos anteriores. La librería más importante de entre las que han sido empleadas para el desarrollo de la aplicación es JENA, elegida para manejar el acceso a los repositorios de ontologías y controlar la aplicación y aprovechamiento de las propias ontologías. En este capítulo, se han descrito, además, otras librerías como JadeGateway y AgentOWL, también básicas para la realización de determinadas actividades por parte de la plataforma.

Seguidamente, se ha mostrado la arquitectura general del sistema implementado, destacando las características generales de la aplicación así como las limitaciones introducidas con respecto al marco de trabajo que constituye la base teórica.

Por último, se ha descrito de forma global la aplicación a partir de las interfaces con los tres usuarios externos identificados: desarrolladores de aplicaciones, proveedores de servicios y consumidores de servicios. Se ha indicado cómo los desarrolladores, encargados de ajustar la plataforma a los requisitos y necesidades de una aplicación particular en un domino predeterminado, pueden hacer uso de numerosas opciones para moldear el sistema para su utilización en un entorno determinado, tal como la adición y eliminación de ontologías y repositorios de ontologías, incorporación de nuevas implementaciones de los roles, e instanciación de los distintos tipos de agentes. Por otra parte, se ha descrito el modo en que los proveedores pueden gestionar los servicios que éstos proporcionan. Finalmente, se ha mostrado como los consumidores pueden realizar consultas, seleccionar y ejecutar servicios, e interactuar con la aplicación por medio de un Servicio Web.

CAPÍTULO VI. APLICACIÓN DEL PROTOTIPO

VI.1. Introducción

En este capítulo, se plantean tres experimentos sobre tres dominios distintos mediante los que se evalúa el prototipo implementado. El propósito principal del capítulo, por tanto, es demostrar la utilidad de la aplicación software desarrollada para la resolución de los distintos tipos de problemas que pueden surgir en dominios tan divergentes como los estudiados, a saber, Administración Digital (*eGovernment*), Bioinformática (en el ámbito de *eScience*) y Comercio Electrónico (*eCommerce*).

El primer experimento trata sobre la Administración Digital, también conocida como Gobierno Electrónico o eGovernment). Sobre este dominio, se propone un escenario en el que un ciudadano desea realizar los trámites pertinentes para cambiar de residencia. A través de este experimento, se puede valorar la utilidad de la aplicación en diversas tareas, entre las que se incluyen el descubrimiento, composición e invocación de servicios, y la presentación combinada de los resultados al usuario.

El segundo experimento aborda el dominio de la Bioinformática. El problema que se afronta en este dominio es la gran cantidad de datos disponibles y la heterogeneidad de los mismos. Mediante un escenario de ejemplo, se comprueba la validez del sistema para proporcionar un acceso integrado a distintas fuentes de datos en biomedicina. En este experimento, además de las tareas más básicas concernientes a la gestión de los Servicios Web (descubrimiento, selección e invocación), se constata la utilidad del uso de las ontologías para desambiguar el contexto.

Finalmente, en el tercer experimento se explora el dominio del Comercio Electrónico y las transacciones comerciales. En este dominio, uno de los problemas que se han tratado en la investigación realizada en la tesis en cuestión es el de la negociación para encontrar el proveedor que ofrece las mejores condiciones de venta. El escenario de ejemplo propuesto consiste en tres proveedores de productos informáticos y un usuario que desea adquirir un componente de ordenador. Mediante dicho escenario, se llevó a cabo un experimento en orden a evaluar la utilidad del prototipo descrito en el capítulo

anterior para la realización de transacciones de Comercio Electrónico que requieren funciones como la identificación de proveedores y de servicios (descubrimiento), la negociación y generación de contratos (selección), la realización de transacciones (invocación), y la evaluación de los productos y cumplimiento del contrato (monitorización).

VI.2. Aplicación en Gobierno Electrónico (eGovernment)

En este experimento, la aplicación software implementada se emplea en el dominio de la Administración Digital (también conocido como Gobierno Electrónico ó eGovernment). En particular, lo que se pretende demostrar con este experimento, es la validez del sistema desarrollado para permitir a los ciudadanos y empresas acceder a los servicios que proporcionan las administraciones públicas a través de la red.

En primer lugar, se ofrece una descripción detallada acerca del dominio en que se va a aplicar el sistema, estableciendo con claridad los problemas a los que se tiene que enfrentar la aplicación en este dominio y los objetivos que se persiguen. A continuación, se muestra el modo en que el marco de trabajo ha sido adaptado a los requisitos del dominio y la aplicación donde se desea utilizar, y se comprueba la funcionalidad del sistema obtenido en el entorno planteado.

VI.2.1. Descripción del Problema y Objetivos

VI.2.1.1. Introducción al eGovernment

En el momento actual de madurez de las TICs, está apareciendo en escena de un modo muy decidido el eGovernment o Administración Digital. Se trata de una de las eTecnologías de mayor relevancia en la actualidad, que pretende remodelar el modo de prestar servicios desde la administración pública.

No se trata, por tanto, de ofrecer los mismos servicios bajo una nueva interfaz. Por el contrario, se hace necesaria una completa reestructuración de los procesos productivos que pongan al ciudadano como eje central del modelo de servicio.

Numerosas organizaciones de diferentes ámbitos han dedicado gran cantidad de tiempo y esfuerzos a la provisión de este tipo de soluciones. Las propuestas

desarrolladas hasta el momento se pueden clasificar en dos grandes grupos: las iniciativas nacionales y los proyectos internacionales. Con respecto a las iniciativas nacionales, se puede destacar que numerosos países han desarrollado sus propias plataformas y entornos para la provisión de soluciones de eGov: SAGA[37] en Alemania, e-GIF[38] en el Reino Unido, ADEA[39] en Francia, EIF[40] dentro de un marco europeo, y FEAF[41] en EE.UU. La mayoría de los países considerados como avanzados han desarrollado sus propias plataformas o modelos de servicio para la provisión de servicios de eGovernment. En (Guijarro, 2004), se ofrece un interesante análisis comparativo de los mismos.

Asimismo, también se pueden encontrar iniciativas fuera del marco de las administraciones nacionales. Estos proyectos, normalmente financiados por convocatorias europeas, pueden clasificarse en función de sus áreas de interés del siguiente modo:

- Semántica: usan la semántica como elemento clave (Ontogov, Terregov, EPRI, etc.).
- Interoperabilidad: buscan servicios que faciliten la compatibilidad de plataformas (EUPubli, QUALEG, SmartGov, etc.).
- Servicios finales: orientados a proveer servicios directamente al ciudadano (eGovernment Good Practice Framework, FASME, eGoia, etc.).

Queda, por tanto, patente, que en la actualidad existe gran cantidad de soluciones, ya sea en fase de desarrollo o de producción, en el dominio de eGovernment. Éstas, en general, están orientadas hacia la provisión de soluciones puntuales que no suelen considerar su compatibilidad con entornos similares. Esto se ve agravado por el enfoque basado en datos, sin soporte semántico, del que se hace uso. Como consecuencia, el nivel de compatibilidad que se da es muy restringido. Entre las deficiencias principales

[37] http://www.kbst.bund.de/saga

[38] http://www.govtalk.gov.uk/

[39] www.adae.gouv.fr/adele

[40] http://ec.europa.eu/idabc/en/document/3473/5585

[41] http://www.whitehouse.gov/omb/egov/a-1-fea.html

de las propuestas realizadas hasta el momento, se pueden destacar los problemas relacionados con la interoperabilidad y la escalabilidad.

Para avanzar hacia soluciones que superen estas dificultades, investigadores de todo el mundo proponen una nueva formulación de problema. Esta nueva tendencia radica en sustituir las soluciones a nivel de datos para un problema en particular, por un enfoque basado en servicios semánticos expresados en términos que tengan en cuenta las necesidades del cliente (Álvarez Sabucedo et al., 2007).

VI.2.1.2. Life Events – aproximación semántica

Una estrategia común para representar los servicios procedentes de la Administración pública, es expresarlos mediante *Life Events* (LEs). El concepto LE se refiere, específicamente, a aquellas situaciones que impulsan al ciudadano a interaccionar con la administración pública. Este estímulo para el ciudadano puede deberse a un interés particular (p.ej., acceso a una beca) o a una imposición por parte de la administración (p.ej., pago de un impuesto). Algunos ejemplos de tipos de LEs pueden ser: mudarse, casarse, perder la cartera, etc. Todos ellos generan una necesidad en el ciudadano que debe ser satisfecha, de un modo transparente, por parte de los procesos administrativos involucrados.

La aplicación de LEs pretende proporcionar una solución orientada hacia la descripción de servicios de un modo centrado en el ciudadano. La primera alusión implícita que encontramos de un modo maduro en el dominio en cuestión a este concepto es en el ámbito del proyecto eGov[42], en el cual se definen dichos LEs como "*situaciones de seres humanos que desencadenan servicios públicos*". La descripción de estos LE se realiza, dentro de este marco, mediante una especificación propia de XML denominada GovML[43]. La finalidad básica de éstos es facilitar la búsqueda de los servicios provistos y guiar, en cierto modo, la implementación de los servicios propuestos.

Estas ideas fueron recogidas por diferentes portales Web y, de un modo más o menos elaborado, puestas a disposición del público. Ejemplos de ello son la página web del

[42] http://www.egov-project.org/

[43] http://www.egov-project.org/egovsite/KM_GovMLDataVocabularies_f.pdf

gobierno de Ontario[44], Canadá, la página oficial del gobierno de Nueva Escocia[45], la del gobierno irlandés a través de su Web Oasis[46]. En estos portales, se hace uso de los LE para organizar los contenidos y ofrecerlos, de un modo más amigable, hacia el ciudadano. En esta misma línea, merece una mención especial la iniciativa del gobierno finlandés, que consiste en poner al alcance de sus ciudadanos un soporte de servicios basado en LE apoyándose en una taxonomía[47].

Se puede señalar, pues, que, a estos niveles mencionados de funcionalidad, los LEs se emplean esencialmente como un mecanismo de navegación en portales web. A través de ellos, se permite una búsqueda de servicios de un modo más sencillo.

En un estudio[48] llevado a cabo por Capgemini para la Unión Europea, se identifican los servicios más habituales que están presentes en los portales Web de las administraciones públicas (ver Tabla 13).

Ciudadanos	Empresas
Pago de Impuestos	Contribuciones a la Seguridad Social
Búsqueda de empleo	Pago de impuestos corporativos
Beneficios de la seguridad social	Pago del IVA
Documentos personales	Registro de una nueva empresa
Registro de un coche	Envío de información para estadística
Solicitud de permiso de obra	Declaración de impuestos
Declaración a la policía	Permisos relacionados con el entorno
Bibliotecas públicas	*Public procurement*
Certificados de Matrimonio y Nacimiento	
Matriculación en enseñanza superior	
Anuncios de cambio de domicilio	
Servicios relacionados con la salud	

Tabla 13. Servicios considerados relevantes en el dominio

Existen varios proyectos en los que se ha abordado la aplicación de Servicios Web Semánticos en entornos de eGovernment. Un ejemplo a destacar es DIP[49] ("*Data,*

[44] http://www.gov.on.ca/

[45] http://www.gov.ns.ca/snsmr/lifeevents/e/

[46] http://www.oasis.gov.ie/siteindex/by_life_event.html

[47] http://www.suomi.fi/suomi

[48] http://www.epractice.eu/files/media/media_854.pdf

Information, and Process Integration with Semantic Web Services"), un proyecto financiado de la Unión Europea desarrollado entre Enero de 2004 y Diciembre de 2006 con una financiación total de 16.3 millones de Euros. Uno de los objetivos principales de este proyecto fue la aplicación de la tecnología de los Servicios Web Semánticos al sector del eGovernment para proporcionar un sistema de gestión inteligente de información y un mecanismo para la integración de aplicaciones.

VI.2.1.3. Escenario de ejemplo

El experimento que se propone en esta sección, se basa en la utilización de Servicios Web Semánticos para proporcionar los servicios disponibles por parte de las administraciones públicas. En particular, se plantea un escenario de ejemplo donde un ciudadano se enfrenta a los trámites necesarios que son necesarios llevar a cabo para conseguir un cambio de residencia.

La operación de cambiar el domicilio habitual requiere una gran cantidad de trámites y esfuerzo administrativo por parte del ciudadano, quien, normalmente, debe realizar multitud de trámites redundantes ante la misma administración. Sin embargo, con el objetivo de demostrar las cualidades del prototipo implementado y, por ende, el marco teórico desarrollado, sin correr el peligro de desviar la atención con los complicados trámites administrativos asociados a este tipo de operaciones, se ha simplificado el problema. De esta manera, el escenario resultante está compuesto por dos administraciones públicas (se supone que la nueva residencia se encuentra bajo el amparo de un ayuntamiento distinto al original) y son precisas dos operaciones para llevar a cabo el cambio de residencia: (1) darse de baja la residencia en la administración local de origen, y (2) dar de alta la nueva localización de la residencia en la administración local de destino.

En la Fig. 139, se muestra el escenario que contempla los servicios propios de cada una de las dos administraciones públicas consideradas, y el ciudadano que tiene la necesidad de realizar el cambio de residencia y que, por tanto, hace uso de la aplicación Web desarrollada en esta tesis doctoral.

[49] http://dip.semanticweb.org/

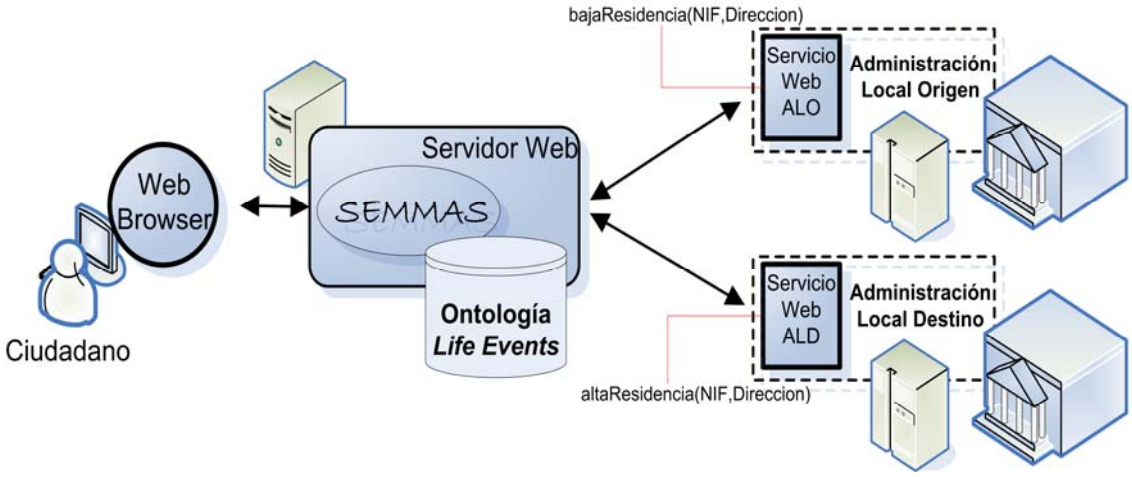


Fig. 139. Escenario de ejemplo para eGovernment

Para satisfacer los requisitos que se plantean en este entorno, son necesarios los servicios apropiados que proporcionen la funcionalidad que se destaca a continuación:

1. Localizar los LE: uno de los problemas más recurrentes en el dominio de eGovernment es la dificultad de los ciudadanos para la búsqueda de aquellos servicios en los que se muestran interesados. Esto se corresponde con las facilidades de descubrimiento, composición y selección aportados por el prototipo implementado.
2. Invocar los LE: una vez que el usuario potencial (esto es, el ciudadano) conoce el LE que se ajusta a sus necesidades, el sistema se enfrenta a la necesidad de usarlo realmente. Para ello, se debe invocar el LE en la administración correspondiente. Con este propósito, se debe hacer uso del servicio asociado a dicho LE proporcionado por la administración en cuestión. Esto se corresponde con la utilidad de invocación aportada por el prototipo implementado.
3. Notificación de los resultados: una vez cumplidos los trámites pertinentes a la operación solicitada por el ciudadano, se le debe remitir a éste una notificación con los resultados obtenidos. La aplicación software implementada, asimismo, dispone de facilidades para la presentación de los resultados de la invocación de los servicios a los usuarios.

VI.2.2. Adaptación del Prototipo a los Requisitos del Dominio

La aplicación software desarrollada ha sido adaptada con éxito para su aplicación en un entorno de Administración Digital (eGovernment). La aplicación del prototipo en este ámbito proporciona a los usuarios una interfaz Web única y amigable, a partir de la que pueden acceder a los distintos servicios que las administraciones públicas ofrecen a ciudadanos y empresas. Con esto, se consigue evitar que los usuarios se tengan que enfrentar a arduos trámites administrativos "manualmente". En cambio, la aplicación desarrollada lleva a cabo de forma automática las gestiones pertinentes en nombre de los usuarios haciendo uso de los Servicios Web que, se asume, han dejado disponibles las distintas administraciones involucradas. Además, dotando al sistema del enfoque semántico y formal característico de aplicaciones como la desarrollada a través de esta investigación, se consiguen superar los más que probables problemas de interoperabilidad que, en otro caso, supondría la interacción con servicios de distintas administraciones.

Para llevar a cabo la adaptación de la plataforma a los requisitos del eGovernment para el problema de cambio de residencia, es preciso llevar a cabo los siguientes pasos:

1. Diseñar una ontología del dominio que contenga los conceptos y relaciones de interés para la aplicación a obtener.
2. Desarrollo de un Servicio Web por cada recurso de información/funcionalidad al que se pretende tener acceso.
3. Anotar semánticamente los Servicios Web haciendo uso de la ontología del dominio y de alguna de las aproximaciones actuales de Servicios Web Semánticos.
4. Instanciar los agentes necesarios del entorno multi-agente para acceder a los servicios proporcionados por las distintas Administraciones Públicas a través de los Servicios Web que han sido descritos semánticamente.

En este apartado, se muestran, en primer lugar, algunas de las características de la ontología del dominio que ha sido empleada en el experimento. Posteriormente, se discuten los detalles de los Servicios Web Semánticos implementados para permitir el acceso a los servicios proporcionados por las Administraciones Públicas. Finalmente, se presenta un ejemplo donde un usuario realiza una consulta sobre la aplicación software

que, haciendo uso de la ontología del dominio diseñada y los Servicios Web desarrollados, es capaz de satisfacer las necesidades del usuario.

VI.2.2.1. Ontología del dominio

Para el escenario de ejemplo planteado, es necesario el desarrollo de una ontología que contemple los conceptos y relaciones en el dominio del eGovernment. Se han desarrollado numerosas ontologías en este dominio que modelan un amplio rango de aspectos: legales, organizacionales, LE, etc.

Para el experimento que se describe en esta sección, se ha hecho uso de una ontología desarrollada en el marco del proyecto DIP que se ajusta, en gran medida, a los requisitos del problema. La ontología, denominada '*Change of circumstances Ontology*', tiene el propósito de modelar cómo diversas agencias o administraciones públicas tienen que ser notificadas acerca del cambio de dirección, o de cualquier otra circunstancia, de una persona que viva en el área de interés de las administraciones en cuestión. Esta ontología, que se centra en modelar una pequeña parte del dominio del eGovernment, fue creada con el propósito de ayudar a definir Servicios Web Semánticos que simplifiquen los trámites que deben llevar a cabo los ciudadanos para notificar, entre otras circunstancias, el cambio de residencia. Por tanto, tanto el propósito de la ontología, esto es, permitir anotar Servicios Web, como el dominio en el que se centra (esto es, la notificación de cambio de dirección en particular y cambio de circunstancias en general), se corresponden perfectamente con el objetivo de este experimento.

La ontología, escrita en OCML pero exportada a RDF(S) y OWL, se describe en (Domingue et al., 2004). En la ontología, se modelan los conceptos, tanto tangibles como abstractos, relevantes en entornos gubernamentales y que tienen relación con un cambio de circunstancias en la situación de un ciudadano particular, y los propios datos de las personas involucradas en dicho cambio. Se definen un total de 95 clases, 1 axioma, 3 reglas y 64 instancias

VI.2.2.2. Servicios Web y anotación semántica

Con la intención de simular un escenario como el presentado, en el que un ciudadano tiene que interactuar con dos instituciones públicas para tramitar una operación administrativa, se han desarrollado dos Servicios Web. Éstos simulan las operaciones

que llevan a cabo sendas administraciones públicas para satisfacer las necesidades de los ciudadanos. Los Servicios Web se han implementado en Java y haciendo uso de la librería Axis2 de Apache.

Cuando los servicios se han implementado, comienza el proceso de anotación semántica de los mismos. Para esto, se emplea la especificación OWL-S y el editor "*OWL-S Editor*"[50], un *plugin* de Protégé desarrollado en el *Stanford Research Institute* (SRI). En este punto, es importante recordar que la aplicación software desarrollada es totalmente independiente de la especificación de Servicios Web Semánticos (p. ej. OWL-S, WSMO, WSDL-S, etc.) utilizada para la anotación semántica de los servicios. De hecho, el soporte a las distintas aproximaciones depende de la implementación de los roles, permitiendo, incluso, que la aplicación proporcione soporte para varias aproximaciones al mismo tiempo.

A continuación, se muestran algunos detalles de los Servicios Web implementados y la descripción semántica de los mismos.

a) Servicio Web Semántico para la Administración Local de Origen

El Servicio Web de la Administración Local de Origen (ALOWS) es el encargado de proporcionar acceso a las funciones suministradas por la entidad administrativa a la que representa. Este servicio, que simula el funcionamiento de la institución pública en su relación con los ciudadanos, dispone de varias operaciones que pueden ser ejecutadas remotamente a través de Internet. A los efectos de este experimento, la operación de interés para un ciudadano que desea cambiarse de residencia es la siguiente:

- *Decision unsubscribeResidence(Citizen, Address)*:
    - Mediante este método, un usuario ('**Citizen**') puede darse de baja como residente en una dirección determinada ('**Address**') en el área adscrita a la administración en cuestión. Como resultado, el método devuelve la decisión ('**Decision**') tomada por la institución pertinente al respecto de la solicitud realizada.

[50] http://owlseditor.semwebcentral.org/

Para poder acceder al Servicio Web, es necesario suministrar una descripción del mismo mediante un documento WSDL. Con WSDL es posible describir sintácticamente las operaciones que proporciona un servicio. En la Fig. 140, se muestra una porción del fichero WSDL del servicio ALOWS.

```
− <wsdl:definitions targetNamespace="http://klt.inf.um.es/">
    <wsdl:documentation> ALOWS </wsdl:documentation>
  − <wsdl:types>
    − <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified" targetNamespace="http://klt.inf.um.es/xsd">
      − <xs:element name="unsubscribeResidence">
        − <xs:complexType>
          − <xs:sequence>
              <xs:element name="citizen" nillable="true" type="ns1:Citizen"/>
              <xs:element name="residence" nillable="true" type="ns1:Address"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      − <xs:element name="unsubscribeResidenceResponse">
        − <xs:complexType>
          − <xs:sequence>
              <xs:element name="return" nillable="true" type="ns1:Decision"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:schema>
      <...>
  </wsdl:types>
+ <wsdl:message name="unsubscribeResidenceMessage"></wsdl:message>
+ <wsdl:message name="unsubscribeResidenceResponse"></wsdl:message>
− <wsdl:portType name="ALOWSPortType">
  − <wsdl:operation name="unsubscribeResidence">
      <wsdl:input message="axis2:unsubscribeResidenceMessage" wsaw:Action="urn:unsubscribeResidence"/>
      <wsdl:output message="axis2:unsubscribeResidenceResponse"/>
    </wsdl:operation>
  </wsdl:portType>
```

Fig. 140. Extracto del fichero WSDL del servicio ALOWS

WSDL proporciona una descripción meramente sintáctica de la funcionalidad ofrecida por el servicio. Para que las aplicaciones software puedan utilizar los servicios de forma automática, es necesario que aquellas tengan acceso a una descripción semántica y formal de los mismos. Éste es, precisamente, el propósito de los Servicios Web Semánticos. Las distintas especificaciones de Servicios Web Semánticos (OWL-S, WSMO, SWSF, WSDL-S, SAWSDL) permiten anotar semánticamente las capacidades de los servicios, habilitando, de este modo, a otros componentes software para procesar esta información y a hacer uso de los servicios.

En el experimento de ejemplo, se ha empleado la especificación OWL-S para describir los servicios. La anotación se ha realizado utilizando la herramienta '*OWL-S*

Editor' y la ontología del dominio presentada en la sección anterior ('*Change of Circumstances Ontology*'). En la Fig. 141, se muestra una porción del fichero con la descripción semántica generada por la herramienta.

```
<?xml version="1.0"?>
<rdf:RDF
…
xml:base="http://klt.inf.um.es:8080/services/description/unsubscribe
Residence.owl">
…
<service:Service rdf:ID="unsubscribeResidenceService">
 <service:presents>
  <profile:Profile rdf:ID="unsubscribeResidenceProfile"/>
 </service:presents>
 <service:describedBy>
  <process:AtomicProcess rdf:ID="unsubscribeResidenceProcess"/>
 </service:describedBy>
 <service:supports>
  <grounding:WsdlGrounding rdf:ID="unsubscribeResidenceGrounding"/>
 </service:supports>
 </service:Service>
<profile:Profile rdf:about="#unsubscribeResidenceProfile">
 <service:presentedBy rdf:resource="#unsubscribeResidenceService"/>
  <profile:serviceName>unsubscribeResidence</profile:serviceName>
  <profile:hasInput>
    <process:Input rdf:ID="citizen">
      <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://dip.semanticweb.org/resources/changeCircumstances.owl#Citizen
</process:parameterType>
    </process:Input>
  </profile:hasInput>
  <profile:hasInput>
    <process:Input rdf:ID="residence">
      <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://dip.semanticweb.org/resources/changeCircumstances.owl#Address
</process:parameterType>
    </process:Input>
  </profile:hasInput>
  <profile:hasOutput>
    <process:Output rdf:ID="return">
      <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://dip.semanticweb.org/resources/changeCircumstances.owl#Decisio
n</process:parameterType>
      </process:Output>
    </profile:hasOutput>
  </profile:Profile>
…
</rdf:RDF>
```

Fig. 141. Extracto de la descripción semántica del servicio AOLWS

b) Servicio Web Semántico para la Administración Local de Destino

El Servicio Web de la Administración Local de Destino (ALDWS) es el encargado de proporcionar acceso a las funciones suministradas por la entidad administrativa a la que

representa. Este servicio, que simula el funcionamiento de la institución pública en su relación con los ciudadanos, dispone de varias operaciones que pueden ser ejecutadas remotamente a través de Internet. A los efectos de este experimento, la operación de interés para un ciudadano que desea cambiarse de residencia a una que se encuentre en un área adscrita a la Administración Local de Destino es la siguiente:

- *Decision subscribeResidence(Citizen, Address)*:
    - Mediante este método, un usuario ('**Citizen**') puede darse de alta como residente en una dirección determinada ('**Address**') en el área adscrita a la administración en cuestión. Como resultado, el método devuelve la decisión ('**Decision**') tomada por la institución pertinente al respecto de la solicitud realizada.

El procedimiento a seguir una vez implementado el servicio es el mismo que se indicó previamente. En primer lugar, se genera el fichero WSDL que contiene la descripción sintáctica de las operaciones que constituyen el servicio. Con esto, y haciendo uso de la ontología del dominio, se produce la anotación semántica del servicio. Los Servicios Web Semánticos aportan, en última instancia, el soporte necesario para que los agentes inteligentes que constituyen la aplicación software implementada sean capaces de hacer un uso efectivo de los servicios suministrados. Para el desarrollo de este experimento, ha sido necesario implementar los Servicios Web y, posteriormente, definir la anotación semántica de los mismos. Estas tareas, idealmente, deben ser llevadas a cabo por las propias entidades que proporcionan los servicios (esto es, los proveedores de servicios).

VI.2.2.3. Ejemplo de aplicación

Una vez definida la ontología del dominio, los Servicios Web dispuestos, y los agentes que constituyen el núcleo de la aplicación software instanciados (no existen requisitos especiales a este respecto), se elaboró un experimento consistente en que un usuario realiza consultas al sistema, que debe resolverlas mediante el uso de los distintos servicios disponibles. El escenario del caso de ejemplo se presentó en la Fig. 139 (pag. 291).

Para estudiar el comportamiento de la aplicación en este entorno, se estudia, paso a paso, el proceso de resolución de una consulta en particular. Se supone que un usuario del sistema, ciudadano residente en el área adscrita a la Administración Local Origen, desea llevar a cabo los trámites concernientes al cambio de residencia. En esta situación, el primer paso a seguir por parte del usuario es indicar este requerimiento al sistema a través de una consulta. La consulta, introducida a partir de texto en lenguaje natural, puede ser la siguiente: "*I want to move from Address 1 to Address 2*". Cuando la consulta ha sido realizada, el agente personal del usuario toma el control del proceso, y el usuario sólo debe esperar la respuesta por parte del sistema. A continuación, se describe la secuencia de pasos que lleva a cabo la aplicación software para conseguir los resultados de la consulta realizada por el usuario. Los pasos se han agrupado según su relación con los retos planteados en este escenario de ejemplo.

1. *Localizar los LE*: descubrimiento, composición y selección de servicios.
   1.1. El primer paso que lleva a cabo el sistema para determinar los servicios que pueden satisfacer los requisitos indicados por el usuario, es procesar la consulta que ha realizado el usuario. Este análisis preliminar permite a la aplicación disponer de una representación formal interna del objetivo planteado por el usuario. Para esto, el '*Customer Agent*' representante del usuario emplea la herramienta KAText, descrita en la sección V.2.2.4. La ontología que se obtiene tras esta primera fase se presenta en la Fig. 142. En la figura, se destaca la relación existente entre los conceptos que constituyen la ontología del objetivo, con los elementos del conocimiento que constituyen la ontología del dominio. Esto refleja el hecho de que la herramienta de procesamiento de lenguaje natural, KAText, se entrena utilizando como base la ontología de componentes informáticos en cuestión.

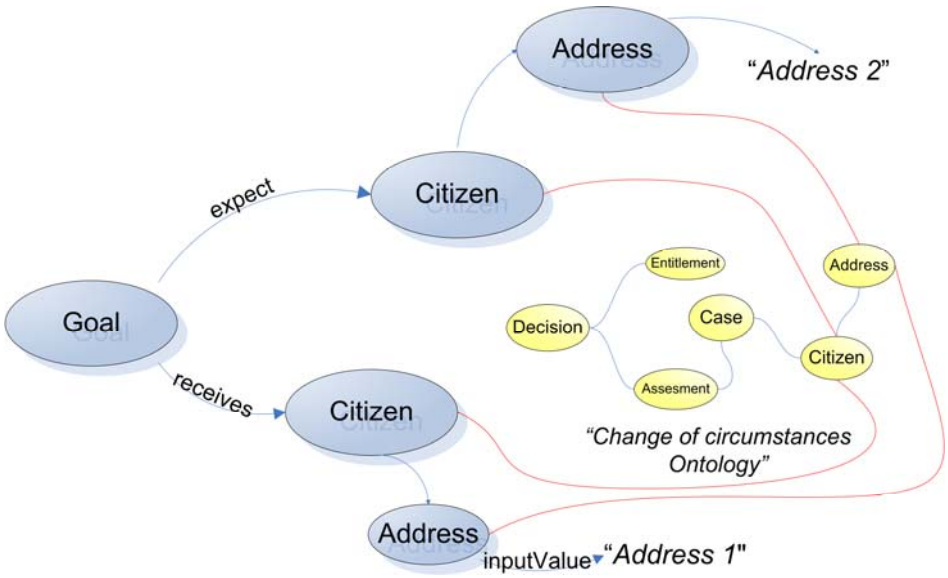


Fig. 142. eGovernment – ontología del objetivo resultante

1.2. Una vez que el objetivo del usuario ha sido representado formalmente, el agente personal lo envía a alguna de las instancias del agente '*Discovery Agent*' disponibles en la plataforma para que se proceda al descubrimiento de los servicios que pueden permitir satisfacer la consulta.

1.3. El '*Discovery Agent*', entonces, accede a los distintos repositorios a los que tiene acceso y, realizando consultas SPARQL sobre todos los servicios almacenados en dichos repositorios, identifica los que pueden ser válidos conforme al objetivo perseguido. Como se describió en detalle en la sección V.3.2.2, el mecanismo utilizado para el descubrimiento se fundamenta en la correspondencia entre las salidas esperadas de los servicios descritos y la salida requerida por el objetivo. En este ejemplo, no existe ninguna operación que devuelva como resultado lo que espera el usuario (o sea, que la dirección pase a ser "*Address 2*"). Por tanto, el agente debe buscar una composición de servicios tal que se obtenga los que han sido solicitados por el usuario.

1.4. La composición la lleva a cabo el propio '*Discovery Agent*' a través del rol '*Composer*'. El mecanismo implementado en el prototipo para la composición de servicios se basa en el empleo de un planificador STRIP para descomponer el objetivo original en diversos subobjetivos. Para definir completamente el planificador, es necesario indicar un conjunto de estados posibles, los

operadores que transforman el mundo de un estado a otro, el estado inicial del que se parte, y el objetivo o estado final esperado. En la Fig. 143, se presenta la adaptación del planificador al escenario del ejemplo en cuestión.

```
;Estados Posibles:
   Residence(C, A)     ;el ciudadano C reside en la dirección A
   OldResidence(C, A) ;el ciudadano C quiere darse de baja como
                       ; residente en A
   NewResidence(C, A) ;el ciudadano C quiere registrase como
                       ; residente en A
   NoResidence(C)      ;no consta ninguna dirección para el ciudadano
                       ; C
   Decision(D)         ;la decisión D ha sido tomada con respecto a
                       ; algún caso
;Operadores:
   UnsubscribeResidence
      Preconditions: OldResidence(C, A)
      Delete: OldResidence(C, A)
      Add: NoResidence(C), Decision(D)
   SubscribeResidence
      Preconditions: NoResidence(C), NewResidence(C, A)
      Delete: NoResidence(C), NewResidence(C, A)
      Add: Residence(C, A), Decision(D)
;Estado Inicial:
   Init: OldResidence('Citizen 1', 'Address 1'),
         NewResidence('Citizen 1', 'Address 2')
;Objetivo:
   Goal: Residence('Citizen 1', 'Address 2')
;Plan:
   Plan: [UnsubscribeResidence, SubscribeResidence]
;Estado final del mundo
   World state: Residence('Citizen 1', 'Address 2'),
                Decision('Unsubscribe OK'), Decision('Subscribe OK')
```

Fig. 143. Plan para la descomposición del objetivo

1.5. En base a la descomposición propuesta por el rol '*Composer*', el '*Discovery Agent*' genera los subsiguientes objetivos que el rol '*Matchmaker*' intenta hacer corresponder de nuevo con los servicios. En este caso, se encuentra la correspondencia adecuada constituida por la composición de la operación '*unsubscribe*' del servicio ALOWS y la operación '*subscribe*' del ALDWS (ver Fig. 144).

Fig. 144. eGovernment – servicios encontrados tras la descomposición

1.6. El '*Discovery Agent*', entonces, envía esta información al agente personal del usuario, que presenta la lista de servicios al usuario para su confirmación (en este caso de ejemplo, no se lleva a cabo el proceso de selección puesto que sólo se ha encontrado un posible servicio para cada uno de los subobjetivos) (ver Fig. 145).

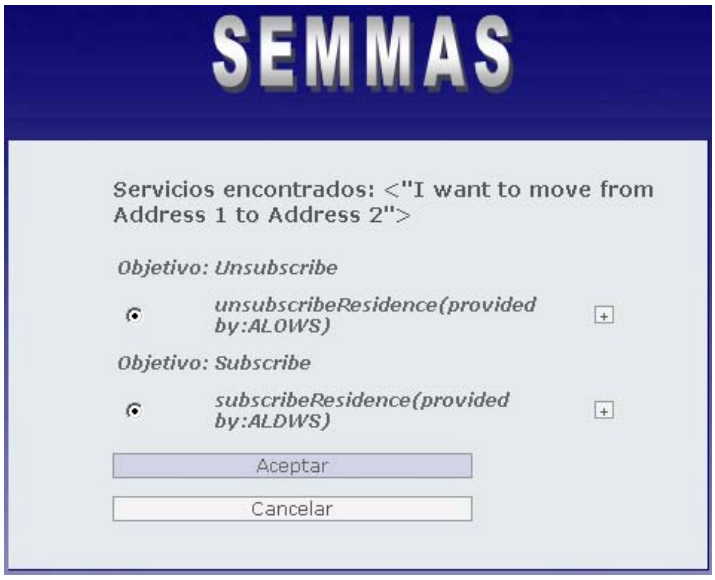


Fig. 145. eGovernment – selección de servicios

2. *Invocar los LE*: invocación de los Servicios Web

2.1. Cuando el usuario da el visto bueno en relación a los servicios a ejecutar, se procede a la ejecución de los mismos. Para esto, el '*Customer Agent*', que recibe la confirmación por parte del usuario, se pone en contacto con los agentes '*Service Agent*' que representan a los servicios seleccionados y solicita la invocación de los mismos.

2.2. Cuando un '*Service Agent*' recibe una petición por parte del agente personal del usuario, comienza el procedimiento de invocación del servicio que representa. En primer lugar, analiza la descripción semántica del servicio para determinar cuáles son los parámetros de entrada para la invocación del servicio y qué resultado debe esperar. Para la ejecución de los dos servicios, la información

requerida es la misma, y se encuentra disponible en la ontología que representa el objetivo del usuario. De este modo, el '*Service Agent*' puede hacer efectiva la invocación del servicio, empleando como parámetros de entrada los datos obtenidos de la ontología. Cuando recibe el resultado, genera el objeto apropiado y lo devuelve al agente personal del usuario.

3. *Notificación de los resultados*: presentación de los resultados al usuario
   3.1. El '*Customer Agent*' recibe los resultados de ambos servicios, los combina, y muestra el resultado final de las operaciones al usuario (Fig. 146).

Fig. 146. eGovernment – resultado de la consulta

VI.3. Aplicación en Bioinformática

En el experimento descrito a continuación, se ha hecho uso de la aplicación software implementada para el dominio de la Bioinformática. En particular, lo que se pretende demostrar con este experimento, es la validez del sistema desarrollado para proporcionar acceso integrado a fuentes de datos heterogéneas.

En primer lugar, se ofrece una descripción detallada acerca del dominio en que se ha aplicado el sistema, estableciendo con claridad los problemas a los que se tiene que enfrentar la aplicación en este dominio y los objetivos que se persiguen. A continuación, se muestra el modo en que el marco de trabajo ha sido adaptado a los requisitos del dominio y la aplicación donde se desea utilizar, y se comprueba la funcionalidad del sistema obtenido en el entorno indicado.

VI.3.1. Descripción del Problema y Objetivos

En este apartado, se introduce el concepto de "*Bioinformática*" y se describe su relación con disciplinas tan variadas como la Medicina, la Biología y las TIC, al tiempo que se destacan los problemas con los que se tiene que enfrentar esta nueva área de investigación. A continuación, se listan algunas de las numerosas fuentes de datos que ofrecen información biológica, clasificándolas en dos grandes grupos, bases de datos biológicas y bio-ontologías. Finalmente, se presenta el escenario de ejemplo al que se tiene que enfrentar el prototipo implementado una vez ha sido adaptado a las condiciones particulares del problema en cuestión.

VI.3.1.1. Introducción a la Bioinformática

La *bioinformática* consiste en la aplicación de las TIC en biología molecular y genómica, de forma que se aporten las herramientas y recursos necesarios para favorecer la investigación en biomedicina. La *bioinformática* se está convirtiendo en un elemento fundamental para la investigación y el desarrollo de sistemas que permitan entender el flujo de información de genes a estructuras moleculares. De esta forma, *bioinformática* se puede definir como un área de investigación multidisciplinar que sirve de puente entre diversas disciplinas, entre las que se incluyen Biología, las TIC y Medicina. La *bioinformática* permite la integración de los datos y la información contenida en estas dispares áreas de conocimiento. Además, facilita el descubrimiento de nuevo conocimiento, así como el desarrollo de perspectivas globales desde las que se pueden derivar algunos de los principios unificadores de las distintas áreas. En este sentido, la colaboración entre las áreas de medicina y biología, hace de la bioinformática la herramienta integradora y el soporte tecnológico de estas disciplinas (Martin-Sanchez et al., 2004).

La era de la genómica viene acompañada de un incremento considerable en la cantidad de información biológica disponible. Esto es debido a los grandes avances en los campos de la biología y la genómica molecular. Sin embargo, los datos se encuentran distribuidos en numerosas fuentes de datos biológicos heterogéneas, que proporcionan poca o ninguna información respecto a la organización de la información. De este modo, las tareas de organización e integración de tales datos se han convertido en tareas cruciales y estratégicas en esta área de investigación. Este hecho queda aún

más patente en el instante en que se determina que el rendimiento de los métodos computacionales para el análisis de datos (biológicos) depende, en gran medida, del modo en que estén organizados los datos. Más crítica, incluso, es esta necesidad para las tareas de interpretación de los datos. De hecho, el aprovechamiento de la integración de las fuentes de datos biológicas actuales, puede suponer un gran beneficio para la investigación en estos campos. La necesidad de disponer de mecanismos de integración de información tiene impacto, además de en *bioinformática*, en otros muchos campos de aplicación.

En los últimos años, se han propuesto numerosas soluciones para llevar a cabo la integración de recursos de información biológicos. En primer lugar, es necesario aclarar que herramientas tales como '*Entrez*'[51] no proporcionan acceso integrado a la información, sino un mecanismo de navegación amigable para explorar los contenidos de diversas bases de datos biológicas que están interrelacionadas. Entre las particularidades que se tienen que tener en cuenta para la integración de información, se incluyen las siguientes: (1) identificación de los objetos y conceptos comunes; (2) integración de los datos que están representados en diferentes formatos; (3) resolución de los conflictos entre los recursos; (4) sincronización de los datos; y (5) visualización unificada. Algunos de los intentos recientes para realizar integración de datos, combinan las tecnologías de almacenamiento de datos (*data warehouse*), el uso de ficheros indexados, sistemas de gestión de bases de datos relacionales, búsquedas individuales en recursos, y envolturas asociadas a los recursos. Ejemplos de este tipo de sistemas son DiscoveryLink (Hass et al., 2001), SRS (Etzold et al., 2003), y BioMart (Durinck et al., 2005).

Sin embargo, el uso de esquemas heterogéneos para el almacenamiento de los datos, diseñados primordialmente para asegurar la optimización del espacio de almacenamiento, complica la realización de consultas sobre las distintas fuentes de datos de un modo integrado. Recientemente, se ha hecho uso de soluciones basadas en servicios semánticos para proporcionar acceso integrado a los recursos en *bioinformática*. Un ejemplo de esto es BioMoby (Wilkinson et al., 2005), que define un mecanismo estándar de paso de mensajes basado en ontologías a través del que un

[51] http://www.ncbi.nlm.nih.gov/entrez/

cliente puede descubrir e interactuar automáticamente con proveedores de servicios de datos biológicos y funciones analíticas apropiados para una tarea determinada, sin requerir una manipulación manual del formato de los datos.

La Web Semántica proporciona un marco común que hace posible la integración de datos, compartiendo y reutilizando datos de múltiples fuentes. En este sentido, un dominio de aplicación puede ser descrito utilizando diversas terminologías, incluyendo diferencias tanto sintácticas como semánticas. Debido a la heterogeneidad propia de los recursos de información biológicos, se planteó en *bioinformática* la utilización de ontologías, que pueden ayudar a representar lo que se conoce acerca de un dominio particular de un modo explícito, modular y completo. En consecuencia, los Servicios Web Semánticos juegan un rol vital para permitir el acceso integrado a los recursos de información, toda vez que cada recurso define los servicios que proporciona haciendo uso de una ontología común que los sistemas de consulta pueden entender. Además, mediante el empleo de Servicios Web Semánticos, no es necesario que los usuarios humanos realicen las tareas de búsqueda de los servicios disponibles para obtener la información relevante, sino que esta tarea se delega en sistemas software como, por ejemplo, agentes inteligentes. BIRD (Gómez et al., 2007), por ejemplo, es una aplicación que simula un entorno de ejecución de Servicios Web Semánticos, y saca provecho de los servicios que proporcionan acceso a fuentes de datos biomédicas para proporcionar a los usuarios un entorno mediante el que es posible el acceso integrado a estos repositorios de datos.

VI.3.1.2. Fuentes de Información Biológica

Existen, en la actualidad, una gran cantidad de fuentes de datos heterogéneas independientes que son accesibles a través de Internet. Estos recursos de información cubren, por ejemplo, la información genómica y otras áreas de biología molecular y medicina. La mayoría de la información está almacenada en bases de datos, aunque, en los últimos años, se han desarrollado algunas ontologías para armonizar la terminología utilizada en las diferentes bases de datos y para promover el acceso integrado y la interoperabilidad entre diferentes recursos de información y aplicaciones. En este apartado, se discuten varias bases de datos biológicas y ontologías en el área en cuestión

(esto es, biología molecular y genómica), con especial énfasis sobre las que han sido utilizadas en el experimento que se describe en esta sección.

a) ***Bases de Datos biológicas***: en los últimos 30 años, se han desarrollado una incontable cantidad de bases de datos biológicas. Las bases de datos biológicas pueden ser clasificadas en base a diferentes propiedades, como el tipo de información que contiene (información acerca de proteínas o genes), el origen de la información (bases de datos primarias –resultados experimentales-, bases de datos secundarias –resultados obtenidos de analizar las bases de datos primarias), o el modo en que la información se encuentra estructurada ("flat file" o bases de datos). La mayoría de las bases de datos se representan utilizando "flat files", de forma que los procesos de gestión de información asociados a las mimas son menos efectivos que los que se llevan a cabo a través de bases de datos relacionales, dado que cada base de datos "flat file" tiene su propio formato.

   En general, las instituciones biomédicas tienen a su disposición un catálogo con distintas bases de datos. Este es el caso del Centro Nacional de Información Biotecnológica ('*National Center for Biotechnology Information*', NCBI) o el Instituto Europeo de Bioinformática ('*European Bioinformatics Institute*', EBI). Para el experimento en cuestión, se ha hecho uso de la colección de bases de datos ofrecidas por el NCBI. Esta colección puede consultarse de forma uniforme a través de la interfaz Entrez. Sin embargo, esta interfaz única reenvía la consulta a cada base de datos individual y muestra los resultados para cada base de datos. Entre las bases de datos a las que se tiene acceso a través de Entrez, para el experimento se ha hecho uso de las de genes y proteínas. La base de datos de proteínas contiene secuencias de datos de las regiones de codificación traducidas de secuencias de ADN en GenBank, EMBL, y DDBJ, así como secuencias de proteínas registradas en '*Protein Information Resource*' (PIR), SWISS-PROT, '*Protein Research Foundation*' (PRF), y '*Protein Data Bank*' (PDB) (secuencias de estructuras resueltas). Por otro lado, la base de datos de genes proporciona un entorno de consulta unificado para genes definidos por secuencia y/o en el visor de mapas del NCBI. Las consultas pueden ser elaboradas para nombres, símbolos, publicaciones, términos GO (*Gene Ontology*), etc.

b) ***Bio-ontologías***: El número creciente de bases de datos heterogéneas e independientes supone un problema para los grupos de investigación cuando éstos desean buscar información en diversas bases de datos. En particular, las búsquedas son ineficientes dada la diversidad terminológica. Debido a esta situación, varias de las entidades que habían desarrollado sus propias bases de datos, trabajaron conjuntamente para definir un vocabulario común que permitiese modelar el modo en que productos genéticos pueden ser anotados. Este fue el origen de la '*Gene Ontology*' (GO) (The Gene Ontology Consortium, 2000), que ofrece información acerca de los atributos de productos genéticos. La GO proporciona un modo de resolver la heterogeneidad semántica asociada a las anotaciones de productos genéticos en diversas bases de datos: anotaciones de diferentes bases de datos se enlazan con el mismo término de la GO. Los principales componentes de la GO son los términos y sus relaciones. Cada término tiene un único identificador, aparte del nombre del término. La GO está dividida en tres ontologías independientes: funciones moleculares, procesos biológicos y componentes celulares. La ontología de funciones moleculares describe el rol básico molecular de los genes. Cada proceso biológico, por su parte, se compone de diferentes funciones moleculares y define su rol a un nivel conceptual. La ontología de componentes celulares representa la estructura de la célula eucariota. Se contemplan dos tipos de relaciones: *Es_Un*, que describe un término como subclase de una clase superior, y *Parte_De*, que significa que un hijo es un subproceso o un componente estructural de su predecesor. La GO puede explorarse a través de varias herramientas. La interfaz Web '*AmiGO*' es el mecanismo de navegación más utilizado. Existen versiones de estas tres ontologías en lenguajes ontológicos como OWL y OBO, además de ser exportable a MySQL.

En los últimos años, la creación de métodos para definir y mantener modelos de domino compartido en biología se ha considerado un aspecto crítico. El diseño, desarrollo y utilización de ontologías en bioinformática se está convirtiendo cada vez en un hito más relevante. Actualmente, las ontologías se han aplicado con éxito en biología para tareas como el acceso a la información. En este contexto, el

proyecto *Open Biomedical Ontology* (OBO)[52] se propuso como un intento de tener un portal que contuviese ontologías así como enlaces a vocabularios controlados para el uso compartido entre los dominios médico y biológico. Las ontologías que se encuentran en OBO son ortogonales y pueden ser combinadas entre ellas incorporando nuevas relaciones y generando nuevas ontologías. Los investigadores en el proyecto OBO han desarrollado, además, el lenguaje OBO para representar ontologías biomédicas. Sin embargo, el lenguaje OWL es más popular que OBO, de forma que con este lenguaje existen mejores herramientas para realizar la gestión, reutilización, compartición, y explotación de contenidos semánticamente enriquecidos.

En este experimento, se hace uso de información acerca de proteínas y genes que puede ser recuperada de las anotaciones de la GO.

VI.3.1.3. Escenario de ejemplo

El escenario de ejemplo, que constituye el experimento con el que se trata de evaluar la utilidad y eficacia de la aplicación software para proporcionar un acceso integrado a diversas fuentes de datos biomédicas, se muestra en la Fig. 147. El escenario está compuesto por diversas bases de datos biomédicas a las que tiene acceso la aplicación software a través de sus respectivos Servicios Web. Un usuario, haciendo consultas sobre la aplicación software, puede tener acceso a las distintas bases de datos y tener su consulta resuelta en base a la información almacenada en estas bases de datos.

[52] http://obo.sourceforge.net/

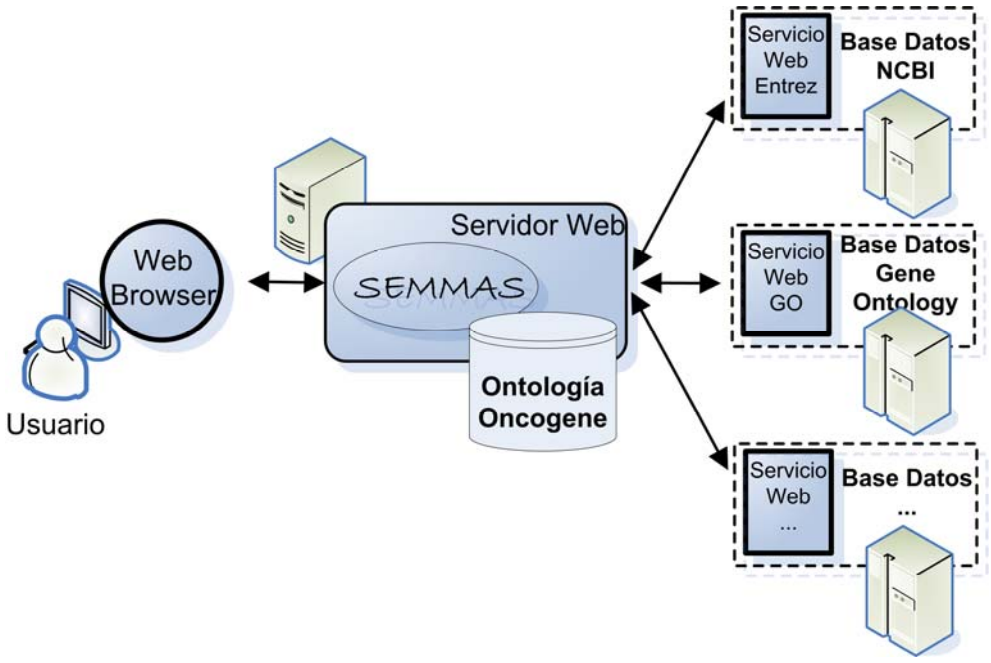


Fig. 147. Escenario de ejemplo para Bioinformática

Se ha limitado la dificultad del experimento haciendo uso de la misma ontología del dominio, que emplea la aplicación software para definir también las anotaciones semánticas de los servicios. De esta forma, se evita la ambigüedad semántica y el problema se restringe a permitir un acceso integrado a distintas fuentes de datos biomédicas.

VI.3.2. Adaptación del Prototipo a los Requisitos del Dominio

La aplicación software desarrollada ha sido adaptada con éxito para aplicarla en la integración de varias fuentes de información heterogéneas en biomedicina. La aplicación del prototipo en este ámbito proporciona a los usuarios una interfaz Web única y amigable para los usuarios, a partir de la que pueden realizar consultas a los distintos recursos, sin tener que preocuparse por la armonización de las terminologías que son utilizadas en cada una de las fuentes de datos.

Para llevar a cabo la adaptación de la plataforma a los requisitos del dominio de la bioinformática, es preciso llevar a cabo los siguientes pasos:

1. Diseñar una ontología del dominio que contenga los conceptos y relaciones de interés para la aplicación a obtener.

2. Desarrollo de un Servicio Web por cada recurso de información/funcionalidad al que se pretende tener acceso.
3. Anotar semánticamente los Servicios Web haciendo uso de la ontología del dominio y de alguna de las aproximaciones actuales de Servicios Web Semánticos.
4. Instanciar los agentes necesarios del entorno multi-agente para recuperar información biomédica de las fuentes de datos a través de los Servicios Web que han sido descritos semánticamente.

En este apartado, se muestran, en primer lugar, algunas de las características de la ontología del dominio que ha sido empleada en el experimento. Posteriormente, se discuten los detalles de los Servicios Web Semánticos implementados para permitir el acceso a los distintos recursos de información biomédica. Finalmente, se presenta un ejemplo donde un usuario realiza una consulta sobre la aplicación software que, haciendo uso de la ontología del dominio diseñada y los Servicios Web desarrollados, es capaz de satisfacer las necesidades del usuario.

VI.3.2.1. Ontología del dominio

El experimento propuesto pretende permitir a los usuarios acceder a información relacionada con los *oncogenes* que se encuentre disponible en alguna de distintas fuentes de datos identificadas. Un *oncogen* es un gen modificado, o un conjunto de nucleótidos, que codifican una proteína y que provocan el surgimiento de tumores malignos. Algunos oncogenes, que están involucrados en algunas de las primeras etapas en el desarrollo del cáncer, incrementan las posibilidades de que una célula normal se convierta en una célula tumoral, dando como resultado, posiblemente, un cáncer. Existen diversos métodos para clasificar los oncogenes, pero ninguno de ellos es considerado un estándar. Para la elaboración del experimento, se ha diseñado y desarrollado la ontología '*Oncogene*'[53]. Esta ontología fue desarrollada debido a la inexistencia en la literatura del área de una representación formal y estándar de los aspectos biológicos y médicos de los oncogenes.

[53] http://klt.inf.um.es/ontologies/oncogene.htm

El modelo utilizado para representar el conocimiento de oncogenes, presentado en (Fernandez-Breis et al., 2004), está basado en la representación de múltiples dominios restringidos jerárquicos y establece que: (1) todos los conceptos se definen a través de un conjunto de atributos y no se considera la presencia de axiomas entre estos atributos; (2) los conceptos se relacionan por medio de diferentes relaciones semánticas. Los atributos conforman la estructura interna de los conceptos, mientras que las relaciones del dominio constituyen su estructura externa. Cada atributo se define por su nombre y su tipo. El tipo es el rango de valores a partir del que el atributo toma su valor. En la ontología en cuestión, se seleccionó una estructura del estilo tipo de valores. En este modelo, un atributo puede ser específico o heredado. Un atributo se identifica únicamente por su nombre y no tiene estructura interna. Entre los tipos de relaciones considerados, se incluyen las relaciones taxonómicas (*ES_UN*) y mereológicas (*PARTE_DE*). Las taxonomías clasifican el conocimiento del dominio, y las mereologías indican la relación entre las partes y el todo. El conjunto de relaciones en el modelo define una serie de axiomas que son el resultado de las relaciones: "concepto tiene atributo", "es una clase de", "es una parte de", o "está conectado con", entre otras. Algunos ejemplos de estas relaciones son "oncogene es una mutación", "el ADN se transcribe en ARNm", "los aminoácidos son una cadena de polipéptidos".

Con el propósito de compartir esta ontología con la comunidad, '*Oncogene*' se ha desarrollado utilizando OWL, la recomendación del W3C para el intercambio de contenido semántico en la Web. En la Fig. 148, se muestra una vista parcial de la ontología. Esta ontología es principalmente una taxonomía, y cada concepto se define a través de las siguientes propiedades: su nombre, sus sinónimos, sus propiedades heredadas de los conceptos padres en la taxonomía, sus propiedades específicas, su fuente de datos original, y su enlace con fuentes externas. La fuente de datos original se refiere al origen de la información: una base de datos, el nombre de un laboratorio, un investigador, etc. El enlace con fuentes externas relaciona el concepto con otras bases de datos que contienen información acerca del mismo. La inclusión de relaciones terminológicas como los sinónimos es importante debido a que la ontología se utiliza, entre otras cosas, para guiar la entrada de datos en lenguaje natural. Como se puede comprobar en la Fig. 148, los conceptos de más alto nivel de la ontología son cáncer,

carcinogénesis, célula, ciclo celular, cromosoma, gen, mutación y proteína. Esta ontología contiene 203 clases, 10 relaciones mereológicas, 55 atributos, 6 propiedades inversas, 202 relaciones taxonómicas, y 41 instancias de oncogenes.

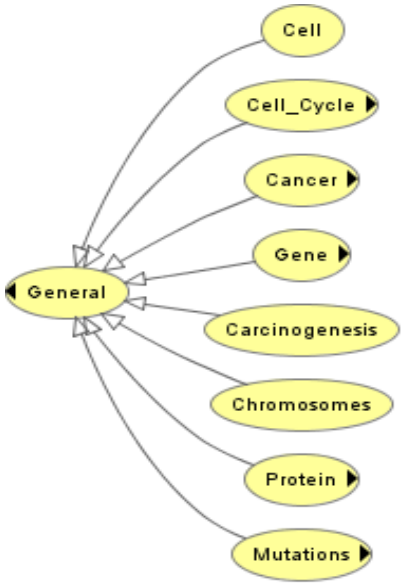


Fig. 148. Jerarquía de alto nivel de la ontología '*Oncogene*'

En la Fig. 149, se muestra otra vista de la ontología '*Oncogene*'. Esta representación alternativa de la ontología permite observar las relaciones del concepto 'Cáncer' en este dominio. De este modo, se puede afirmar que el cáncer se activa por oncogenes. En la figura también se presentan diferentes tipos de cáncer y de oncogenes. Ciertos tipos de oncogenes activan tipos específicos de cáncer.

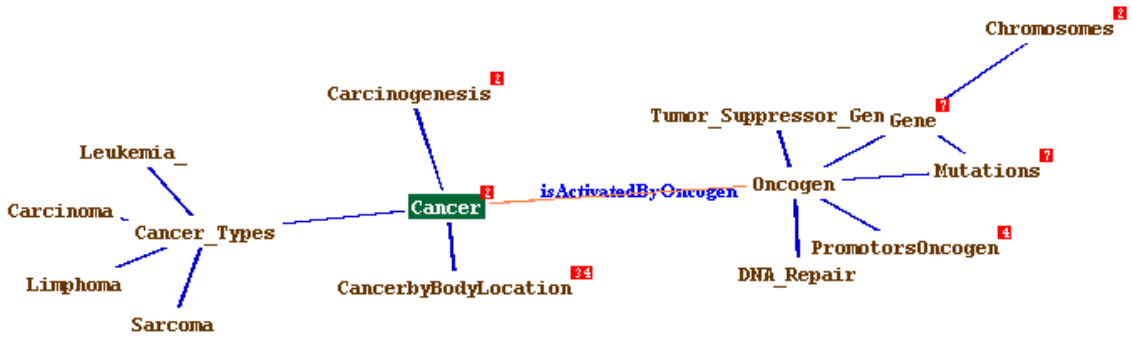


Fig. 149. El concepto Cáncer en la ontología '*Oncogene*'

VI.3.2.2. Servicios Web y anotación semántica

Para permitir el acceso a las fuentes de información en biomedicina, se han desarrollado Servicios Web que son capaces de realizar consultas sobre la interfaz Web de estas fuentes de información, y parsear las páginas con los resultados para generar determinados objetos. Para la implementación de estos Servicios Web, se ha hecho uso del lenguaje de programación Java y la librería Axis2 de Apache (descrita en la sección V.2.2.3).

Cuando los servicios se han implementado, comienza el proceso de anotación semántica de las capacidades de los mismos. Esta descripción semántica se realiza haciendo uso del *plugin* de Protégé "*OWL-S Editor*", que permite la anotación de servicios mediante la especificación OWL-S.

En este apartado, se describe la implementación y posterior anotación semántica de dos de los servicios implementados en el dominio de la bioinformática.

c) Servicio Web Semántico para Gene Ontology

El Servicio Web Semántico de '*Gene Ontology*' (GOSWS) utiliza la vista de base de datos de la '*Gene Ontology*' en vez de la vista ontológica. El GOSWS se define sobre el Servicio Web de '*Gene Ontology*' (GOWS)[54]. El servicio GOWS está compuesto por tres métodos que exponen parte de la funcionalidad proporcionada por el portal Web de '*Gene Ontology*'[55]:

- Obtener información acerca del término '*term*' (*Get_information*):
  - http://amigo.geneontology.org/cgi-bin/amigo/go.cgi?action=query&view=query &session_id=864b1175276845&query=***term***&search_constraint=terms
- Obtener información acerca del gen '*geneName*' (*Get_Gene*):
  - http://amigo.geneontology.org/cgi-bin/amigo/go.cgi?action=query&view=query=***geneName*** &search_constraint=gp
- Obtener información acerca de la proteína '*proteinName*' (*Get_Protein*):
  - http://amigo.geneontology.org/cgi-bin/amigo/go.cgi?action=query&view=query& query=***proteinName***&search_constraint=gp

[54] Disponible en la URL "http://klt.inf.um.es:8080/axis2/services/GO_WebService"

[55] http://www.geneontology.org/

Estos tres métodos del GOWS devuelven la proteína o el gen cuyo identificador se ha introducido como entrada, o el identificador de un elemento biológico. Una vez implementado el servicio GOWS, se desarrolló el fichero WSDL asociado (ver Fig. 150). Un fichero WSDL describe sintácticamente las operaciones enumeradas anteriormente que constituyen la funcionalidad del servicio.

```
<wsdl:operation name="getProtein">
   <wsdl:input message="axis2:getProteinMessage" wsaw:Action="urn:getProtein"/>
   <wsdl:output message="axis2:getProteinResponse"/>
</wsdl:operation>
<wsdl:operation name="getInformation">
   <wsdl:input message="axis2:getInformationMessage" wsaw:Action="urn:getInformation"/>
   <wsdl:output message="axis2:getInformationResponse"/>
</wsdl:operation>
<wsdl:operation name="getGene">
   <wsdl:input message="axis2:getGeneMessage" wsaw:Action="urn:getGene"/>
   <wsdl:output message="axis2:getGeneResponse"/>
</wsdl:operation>
```

Fig. 150. Fichero WSDL del servicio GOWS

El siguiente paso es la generación del GOSWS, la anotación semántica del servicio GOWS. Para esto, como se indicó anteriormente, se ha empleado el plugin de Protégé '*OWL-S Editor*' y la ontología '*Oncogene*'. Se ha generado, para cada uno de los métodos que contiene el servicio, un fichero OWL que contiene la descripción semántica de la operación en OWL-S.

En la Fig. 151, se muestra un extracto del fichero '*getProtein.owl*'. Es posible comprobar, examinando el extracto presentado, que los parámetros de entrada y salida de la operación están estrechamente ligados con la ontología del dominio, esto es, con la ontología '*Oncogene*'. De esta forma, un consumidor de servicios que tenga acceso tanto a la ontología '*Oncogene*' como a los ficheros que contienen la anotación en OWL-S de los Servicios Web, podrá hacer uso de los servicios superando la heterogeneidad presente en las fuentes de datos originales y la ambigüedad asociada.

```
<?xml version="1.0"?>
<rdf:RDF
…
xml:base="http://klt.inf.um.es/services/owls/getProtein.owl">
 <owl:Ontology rdf:about="">
  <owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.2/Grounding.owl"/>
```

```
  <owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.2/Profile.owl"/>
  <owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.2/Process.owl"/>
  <owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.2/Service.owl"/>
 </owl:Ontology>
 <prof:Profile rdf:ID="getProteinProfile">
  <prof:serviceName
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >getProtein</prof:serviceName>
  <prof:hasOutput>
    <proc:Output rdf:ID="return">
      <proc:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://klt.inf.um.es/ontologies/oncogene.owl#Protein
</proc:parameterType>
      <rdfs:label
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >return</rdfs:label>
    </proc:Output>
  </prof:hasOutput>
…
        <proc:hasInput>
          <proc:Input rdf:ID="proteinName">
            <proc:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://klt.inf.um.es/ontologies/oncogene.owl#ProteinName
</proc:parameterType>
          <rdfs:label
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >proteinName</rdfs:label>
          </proc:Input>
        </proc:hasInput>
…
 </prof:Profile>
</rdf:RDF>
```

Fig. 151. Extracto de la descripción semántica del servicio GOWS

d) Servicio Web Semántico para la interfaz Entrez

El Servicio Web Semántico de '*Entrez*' (ESWS) recupera información de la base de datos de proteínas y genes de *Entrez*. El ESWS se define sobre el Servicio Web '*Entrez*' (EWS), que contiene dos métodos diferentes:

- *GetProtein (name)*:
  - http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?CMD=search&DB=protein&term=**name**
- *GetGene(name)*:
  - http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?CMD=search&DB=gene&term=**name**

El mecanismo para obtener la anotación ESWS del servicio EWS es el mismo que se ha explicado anteriormente. Una vez implementado el servicio EWS, se define el fichero WSDL que describe sintácticamente la funcionalidad del Servicio Web. A

continuación, se anota semánticamente el servicio describiendo los métodos que lo constituyen en ficheros OWL-S. Estos Servicios Web Semánticos permiten a los agentes inteligentes interactuar semánticamente con recursos de información biomédica. Para el desarrollo de este experimento, ha sido necesario implementar los Servicios Web y, posteriormente, definir la anotación semántica de los mismos. Estas tareas, idealmente, deben ser llevadas a cabo por las propias entidades que proporcionan los servicios (esto es, los proveedores de servicios).

VI.3.2.3. Ejemplo de aplicación

Una vez definida la ontología del dominio, los Servicios Web dispuestos, y los agentes que constituyen el núcleo de la aplicación software instanciados (no existen requisitos especiales a este respecto), se elaboró un experimento consistente en que un usuario realiza consultas al sistema, que debe resolverlas mediante el uso de los distintos servicios disponibles. El escenario del caso de ejemplo se presentó en la Fig. 147 (pag. 309).

Para estudiar el comportamiento de la aplicación en este entorno, se estudia, paso a paso, el proceso de resolución de una consulta en particular. Se supone que un usuario del sistema quiere obtener la descripción de la proteína que tiene el nombre '*trk2*'. En esta situación, el usuario no experto escribe una frase a partir de texto en lenguaje natural, que en inglés quedaría así: "I want to get the description of the protein with name trk2". Cuando la consulta ha sido realizada, el agente personal del usuario toma el control del proceso, y el usuario sólo debe esperar la respuesta por parte del sistema. A continuación, se describe la secuencia de pasos que lleva a cabo la aplicación software para conseguir los resultados de la consulta realizada por el usuario.

1. El agente personal del usuario, el '*Customer Agent*', analiza la consulta empleando la herramienta KAText, un procesador de lenguaje natural descrito en la sección V.2.2.4. Mediante esta herramienta, el sistema transforma la consulta realizada a partir de texto en lenguaje natural, en una ontología ligera que representa formalmente el objetivo del usuario. La ontología que se obtiene tras esta primera fase en el procesamiento de la consulta se muestra en la Fig. 152.

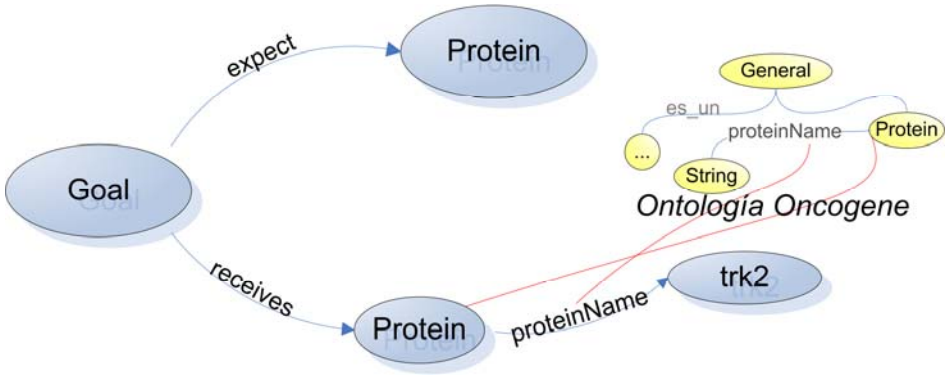


Fig. 152. Bioinformática – ontología del objetivo resultante

2. El agente personal del usuario envía esta consulta, ya representada formalmente por medio de una ontología, a una de (o a todas) las instancias del agente '*Discovery Agent*' presentes en la plataforma.
3. La instancia de '*Discovery Agent*' busca todos los servicios cuya descripción semántica de capacidades indique que son apropiados para satisfacer la consulta realizada por el usuario. Para ello, el agente sigue el procedimiento descrito anteriormente donde descubrimiento y composición se encuentran estrechamente ligados. Para este ejemplo, el agente no necesita llevar a cabo composición alguna, ya que encuentra dos servicios, "*getProtein"* del EWS y "*getProtein*" del GOWS, que cumplen los requisitos del objetivo marcado por el usuario.
4. El '*Discovery Agent*' devuelve al agente personal del usuario la lista completa de servicios identificados para la consulta especificada. A continuación, este agente envía la lista recibida a uno de los agentes '*Selection Agent*' que se encuentren instanciados en el entorno.
5. El '*Selection Agent*' es el responsable de determinar qué servicio, de entre los presentes en la lista, es el más conveniente de acuerdo con las condiciones propuestas por cada uno de ellos y en base a las preferencias del usuario. Se asume que las siguientes hipótesis son ciertas para el experimento que se plantea:
    a. Como condición estratégica, el proveedor de servicios '*Gene Ontology*' establece que la confianza de los resultados de sus servicios es de 0.8. Por su parte, el proveedor '*Entrez*' indica que la confianza de los resultados que ofrecen los servicios que suministra es de 0.6.

b. Una condición adicional que establece '*Gene Ontology*', de forma particular sobre el servicio "*getProtein*", es que el tiempo de respuesta será menor a un minuto. En cambio, el servicio "*getProtein*" de '*Entrez*' asegura un tiempo de respuesta inferior a dos minutos.
c. Entre las preferencias del usuario se establece que el factor de impacto de la confianza de los resultados es de 0.7, frente a un 0.3 de factor de impacto del tiempo de respuesta.

Cuando el '*Selection Agent*' ha negociado con los agentes '*Service Agent*' correspondientes, y en base a las hipótesis enunciadas y el algoritmo propuesto para el cálculo del valor de utilidad de los servicios, el '*Selection Agent*' ordena la lista de servicios.

6. El '*Selection Agent*' devuelve al agente personal del usuario la lista ordenada de servicios así como las condiciones que se han acordado para la ejecución de cada uno de ellos.
7. El agente personal del usuario, '*Customer Agent*', presenta al usuario la lista ordenada de los servicios identificados para que sea el usuario el encargado de determinar qué servicio debe ejecutarse, si es que le interesa alguno de los propuestos (ver Fig. 153). Para cada servicio de la lista, el usuario puede visualizar, además del proveedor del servicio, una completa descripción del mismo que incluye, entre otros elementos, las condiciones establecidas por los '*Service Agent*' para la ejecución de los servicios que representan.

Fig. 153. Bioinformática – selección de servicios

8. Cuando el usuario elige el servicio a ejecutar, que en este caso coincide con el más valorado por el '*Selection Agent*', el agente personal del usuario comienza a

interactuar con el agente '*Service Agent*' pertinente, el que representa al servicio seleccionado, para llevar a cabo, efectivamente, la invocación del servicio.

9. El '*Service Agent*' analiza la descripción semántica del servicio para determinar cuáles son los parámetros de entrada para la invocación del servicio y qué resultado debe esperar. Haciendo esto, el agente descubre que, como parámetro de entrada, el servicio espera el nombre de la proteína cuya descripción se desea obtener y, como resultado, devuelve la descripción de la proteína que, de acuerdo con la ontología '*Oncogene*', incluye elementos como el nombre, la descripción textual, el tipo y la especie. El '*Service Agent*', entonces, comprueba que la información requerida para invocar el servicio se encuentra en la ontología que contiene el objetivo del usuario, con lo que sólo tiene que recuperar tal información y generar la invocación del método. Cuando recibe el resultado, genera el objeto apropiado y lo devuelve al agente personal del usuario.
10. El agente personal del usuario, finalmente, muestra el resultado de la operación al usuario (ver Fig. 154), quien comprueba que su consulta ha sido resuelta con éxito por parte del sistema.

Fig. 154. Bioinformática – resultado de la consulta

VI.4. Aplicación en Comercio Electrónico (eCommerce)

En este experimento, se hace uso de la aplicación software implementada para el dominio de Comercio Electrónico. En particular, lo que se pretende demostrar con este experimento, es la validez del sistema desarrollado tanto para entornos B2C (*Business-to-Consumer*), como para entornos B2B (*Business-to-Business*).

En primer lugar, se ofrece una descripción detallada acerca del dominio en que se va a aplicar el sistema, estableciendo con claridad los problemas a los que se tiene que enfrentar la aplicación en este dominio y los objetivos que se persiguen. A continuación, se muestra el modo en que el marco de trabajo ha sido adaptado a los requisitos del dominio y la aplicación donde se desea utilizar, y se comprueba la funcionalidad del sistema obtenido en el entorno indicado.

VI.4.1. Descripción del Problema y Objetivos

En este apartado, se describen, en primer lugar, las fases y etapas que constituyen una transacción en entornos B2C y B2B. A continuación, en base a los modelos de comportamiento de los consumidores, relacionados estrechamente con las etapas identificadas de estos procesos de negocio, se enumeran las tareas que pueden verse beneficiadas por el uso de Agentes Inteligentes. Finalmente, se presenta el escenario de ejemplo sobre el que se ha aplicado y evaluado el sistema.

VI.4.1.1. Introducción al Comercio Electrónico

Según una definición procedente de Wikipedia[56], "*el comercio electrónico consiste principalmente en la distribución, compra, venta, mercadotecnia y suministro de información complementaria para productos o servicios a través de redes informáticas como Internet u otras.*". Dependiendo de la naturaleza de las transacciones, se pueden identificar distintos tipos de comercio electrónico:

- B2C (de organización a consumidor).
- B2B (entre organizaciones): forma de comercio electrónico en que tienen lugar transacciones que se ejecutan sobre redes privadas o públicas, incluyendo transacciones públicas y privadas que usan Internet como vehículo de comunicación. Estas transacciones incluyen transferencias financieras, intercambios on-line, subastas, entrega y reparto de productos y servicios, actividades de la cadena de suministro y redes de negocio integradas (The USHER Project, 2002b).

[56] http://es.wikipedia.org/wiki/Comercio_electr%C3%B3nico

- B2B2C (entre organizaciones y de organización a consumidor): integración de la cadena de producción (negocio electrónico).
- C2C (de consumidor a consumidor): transacciones en forma de, por ejemplo, subastas por Internet.
- B2G (de organización a gobierno): las administraciones públicas actúan como clientes de las organizaciones.
- G2C (entre gobierno y los ciudadanos): más conocido como gobierno electrónico (ya descrito en la sección VI.2).
- B2E (de organización a sus empleados): novedoso mecanismo para promocionar la identificación del empleado con la organización que consiste en ofrecer ofertas especiales a los empleados.

Para este experimento, nos centramos en los dos primeros tipos enumerados: B2C y B2B. La aplicación a los demás tipos de comercio electrónico puede requerir modificaciones en ciertos componentes del entorno.

El primer paso para diseñar un sistema de comercio electrónico, ya sea entre organizaciones (B2B) o entre consumidores individuales y una organización (B2C), es estudiar las fases y etapas que tienen lugar durante el desarrollo de una transacción de ese tipo.

En la Fig. 155, se muestra de forma gráfica y esquemática el flujo de información y material que tiene lugar en una transacción B2C. Como se introdujo anteriormente, B2C es una de las principales categorías de aplicaciones de comercio electrónico. B2C abarca las transacciones de negocio con consumidores a través de Internet. Como se aprecia en la figura, la transacción la inicia el consumidor que, a través de la Web, realiza un pedido. A este nivel, se incorporan los sistemas de pago y se ofrece a las organizaciones nuevas oportunidades de marketing. El siguiente paso supone la realización efectiva del pedido. En esta fase, se incluyen la notificación de compra, la producción del producto requerido por el usuario, y la distribución del mismo. Hasta este punto, se ha descrito todo lo necesario para llevar a cabo una transacción de comercio electrónico entre una organización y un consumidor individual. Las siguientes etapas, que aparecen en dicha figura, consistentes, básicamente, en la integración de

toda la cadena de suministro con los proveedores, suponen una evolución añadida que transforma el sistema de una aplicación de comercio electrónico (*eCommerce*) a un sistema integral de negocio electrónico (*eBusiness*).

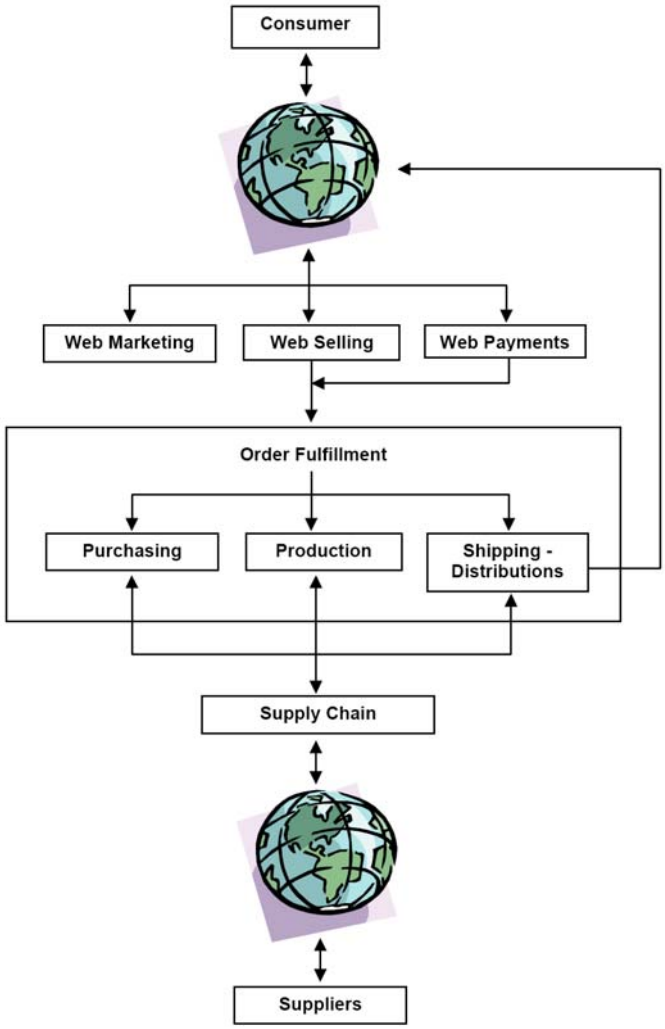


Fig. 155. Flujo de información y material en un entorno integrado de B2C (The USHER Project, 2002a)

En la Fig. 156, se muestra de forma gráfica y resumida el flujo de información y material que tiene lugar en las transacciones B2B. En este caso, el iniciador de la transacción es una organización pública o privada que quiere realizar una orden de compra de una determinada cantidad y tipo de productos de otra organización. Con este propósito, se puede hacer uso de alguno de los distintos tipos de aplicaciones existentes para este fin: sistemas de subastas, portales intermediarios, venta por Web, herramientas de CRM (*Customer Relationship Management*), etc. A través de alguno de estos

métodos, se realiza el pedido, que llega, posteriormente, a la fase de cumplimiento del pedido. Esta fase está dividida, al igual que para las transacciones B2C, en tres etapas: la notificación de compra, la producción del producto o productos solicitados, y el envío de los mismos a la organización consumidora. De forma adicional, como se explicó anteriormente, se puede integrar la transacción de comercio electrónico con toda la cadena de suministro, lo que da lugar a un sistema integral de negocio electrónico.

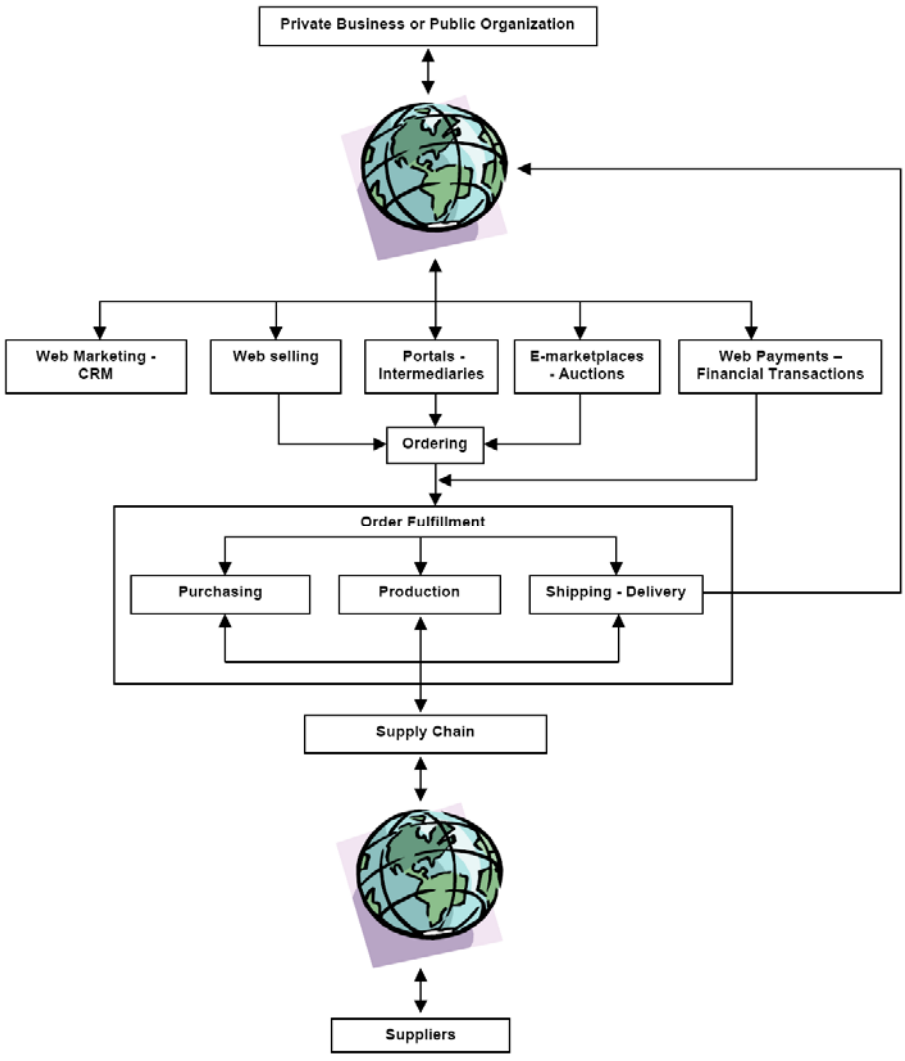


Fig. 156. Flujo de información y material en un entorno integrado de B2B (The USHER Project, 2002b)

Analizando las figuras presentadas y teniendo en cuenta el estudio realizado en esta tesis doctoral con respecto a las tecnologías de agentes y de Servicios Web, es fácil comprobar que una implementación óptima de una herramienta de comercio electrónico para B2C y B2B podría, ciertamente, basarse en una combinación de estas tecnologías.

En particular, la tecnología de Servicios Web se ajusta totalmente a las etapas de que se compone el cumplimiento de un pedido, lo que puede constituir la lógica de negocio de una organización, mientras que la tecnología de agentes, por su parte, es propicia para facilitar los procesos que tienen lugar antes de la realización de un pedido (identificación de proveedores, negociación, etc.).

VI.4.1.2. Agentes en Comercio Electrónico (B2C y B2B)

Según el modelo de comportamiento del consumidor al comprar ('*Consumer Behaviour Buying Model*', CBB) (He et al., 2003), el proceso a seguir en una transacción B2C, desde el punto de vista del consumidor, consta de 7 etapas (ver Fig. 157):

1. Identificación de una necesidad: el cliente reconoce que tiene la necesidad de un producto o servicio determinado.
2. Mediación de productos: determinar qué productos/servicios satisfacen la necesidad del cliente.
3. Formación de coaliciones: los clientes constituyen coaliciones para conseguir mejores condiciones al realizar sus pedidos.
4. Mediación de comerciantes: encontrar el proveedor que ofrece el servicio o al que se le puede comprar el producto.
5. Negociación: negociar los términos y condiciones bajo las cuales se va a entregar el producto o servicio solicitado.
6. Compra y entrega: acción por la que se hace efectiva la compra, y el producto o servicio es entregado al cliente.
7. Evaluación del producto/servicio: evalúa la satisfacción del usuario con el producto/servicio ofrecido.

Conforme al estudio realizado por He et al. (2003), los agentes pueden actuar como mediadores en cinco de las etapas mencionadas: identificación de una necesidad, mediación de productos, formación de coaliciones, mediación de comerciantes y negociación. Para la etapa de identificación de una necesidad, un agente que tenga disponible el perfil del usuario, puede utilizar esta información para notificar al usuario los productos o servicios que pueden ser de su interés. En la etapa de mediación de productos, es el agente la entidad encargada de realizar la identificación de los

productos que satisfacen las necesidades de los clientes (los autores proponen tres posibles técnicas: filtrado basado en características, filtrado colaborativo y filtrado basado en restricciones). Para la formación de coaliciones, si los agentes que representan a los clientes descubren que precisan los mismos productos/servicios o que tienen que interactuar con un mismo comerciante, aquellos se pueden agrupar para generar un pedido más voluminoso que puede conllevar mejores condiciones por parte del comerciante. Por último, en la etapa de negociación, los agentes que representen a los consumidores y a los proveedores pueden negociar para establecer los términos y condiciones bajo los que el producto o servicio va a ser entregado. En base al modelo CBB se han desarrollado diversas aplicaciones en las que agentes inteligentes permiten la realización de transacciones B2C. Un ejemplo, es el sistema descrito en (García-Sánchez et al., 2005), compuesto por un entorno multi-agente en el que agentes representantes de los consumidores y agentes representantes de los proveedores interactúan para llevar a cabo transacciones comerciales.

En cambio, el modelo teórico propuesto en esta tesis doctoral y descrito en el capítulo III, identifica una etapa adicional donde los agentes pueden ser relevantes, a saber, la evaluación del producto/servicio. Durante esta etapa, lo que se persigue es comprobar que el acuerdo al que se ha llegado durante la etapa de negociación está siendo satisfecho por ambas partes. Con este propósito, el marco teórico incorpora un mecanismo de monitorización que permite, tanto a los agentes representantes de los consumidores ('*Customer Agent*'), como a los representantes de los proveedores ('*Service Agent*' y '*Provider Agent*'), constatar que se están cumpliendo las condiciones impuestas. Sin embargo, y por motivos de simplicidad, el prototipo implementado no incluye alguno de los elementos ideados en el marco teórico (lo cual queda pendiente como trabajo futuro). Así, por ejemplo, el prototipo no incorpora, por el momento, un sistema de recomendación que, en base al perfil del usuario y a las transacciones que realice a lo largo del tiempo, sea capaz de mostrar al usuario productos o servicios que pueden ser de su interés. Del mismo modo, la aplicación software tampoco incluye la funcionalidad necesaria para la formación de coaliciones.

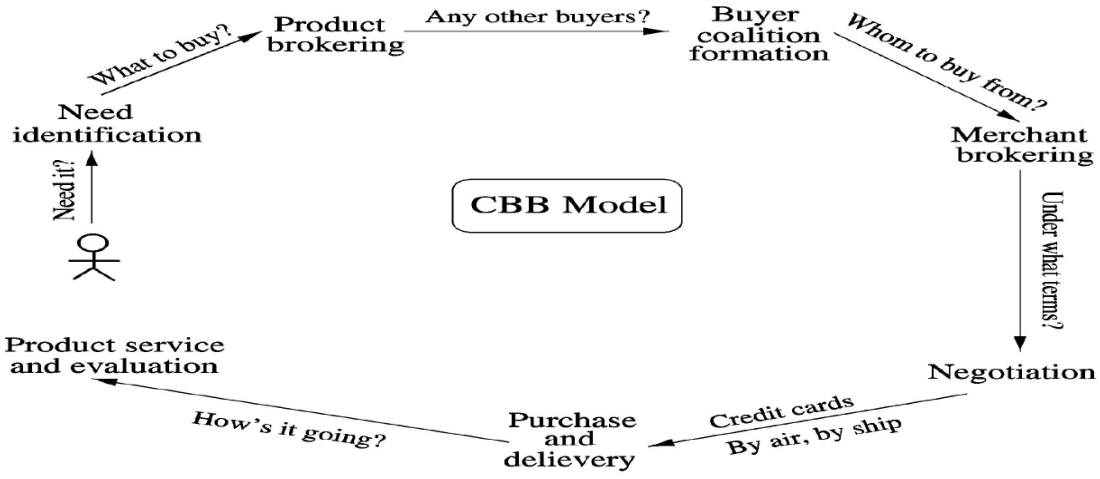


Fig. 157. Modelo de comportamiento del consumidor al comprar (He et al., 2003)

Por otra parte, según el modelo de transacciones negocio-negocio ('Business-to-Business Transaction Model', BBT) (He et al., 2003), el proceso a seguir en una transacción B2B está divido en seis etapas secuenciales (ver Fig. 158):

1. Formación de sociedades: generación de organizaciones virtuales y descubrimiento de socios que aporten los productos o servicios en una cadena de suministro.
2. Mediación: proceso por el que se hace la correspondencia entre vendedores (de forma más genérica, proveedores) que suministran bienes o servicios y compradores (consumidores) que los necesitan.
3. Negociación: etapa donde las partes tratan de llegar a un acuerdo acerca de las acciones a tomar y bajo qué condiciones se deben de tomar tales acciones.
4. Creación de contratos: marca el final del proceso de negociación e implica establecer un contrato legal con los términos acordados.
5. Cumplimiento de contratos: conlleva la realización de la transacción según las condiciones acordadas en el contrato.
6. Evaluación del servicio: etapa que comienza una vez se ha realizado la transacción y consistente en la evaluación de la satisfacción de las distintas partes implicadas en la misma.

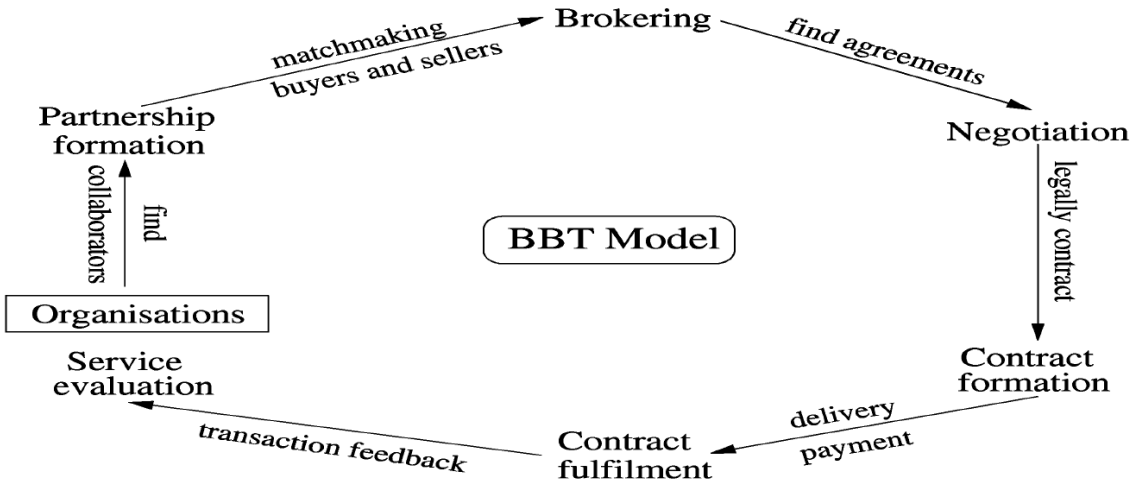


Fig. 158. Modelo del ciclo de vida Business-to-Business (He et al., 2003)

Los autores en (He et al., 2003), afirman que los agentes pueden suponer un valor añadido en las tres primeras etapas de este proceso. El marco teórico propuesto en esta tesis doctoral, sin embargo, amplía esta influencia positiva a las etapas de formación de contratos y de cumplimiento de los mismos. El modelo teórico liga, por completo, las etapas de negociación y de formación de contratos. De hecho, la generación del contrato legal que se debe de satisfacer en caso de, finalmente, llevar a cabo la transacción, es la culminación del proceso de negociación. Por tanto, esta etapa se realiza de forma automática por medio de los agentes que estén involucrados en la transacción en cuestión. Por su parte, el cumplimiento del contrato consiste, en primer lugar, en invocar de forma efectiva el servicio elegido (esto es, llevar a cabo la transacción[57]), y, en segundo lugar, comprobar que las condiciones del contrato son satisfechas por ambas partes (organización cliente y organización proveedora). Del mismo modo que para las transacciones B2C, ciertas restricciones han sido impuestas sobre el prototipo implementado para reducir su complejidad. En particular, la aplicación software implementada no incorpora, en la actualidad, ningún mecanismo que permita la generación de sociedades entre los agentes que representen a organizaciones consumidoras.

[57] Este paso puede llevar asociado un componente físico: el envío del producto al destinatario, o la realización física del servicio solicitado.

VI.4.1.3. Escenario de ejemplo

El escenario de ejemplo que se muestra en la Fig. 159, basado en el ejemplo descrito en la sección IV.2.1, constituye un experimento con el que se pretende evaluar la aplicación software desarrollada en términos de su utilidad y eficacia en un entorno B2C y B2B donde diversos proveedores ofrecen productos/servicios similares. El escenario está compuesto por el usuario (individual o representante de una organización), el prototipo implementado, y tres proveedores de componentes informáticos. Cada proveedor posee un catálogo de productos que difiere de los del resto de proveedores. Además, los proveedores permiten realizar consultas sobre el catálogo y la compra de los productos disponibles a través de un Servicio Web. El usuario, mediante la funcionalidad que le ofrece la plataforma, puede comprar los productos que, en ese momento, se encuentre en el stock de los proveedores.

Se ha reducido el grado de complejidad del experimento haciendo uso de una misma ontología (la ontología del dominio empleada por la aplicación software) para definir las anotaciones semánticas de los servicios. De esta forma, se evita la ambigüedad semántica y el problema se restringe a un correcto manejo de los Servicios Web puestos a disposición de la aplicación en el entorno.

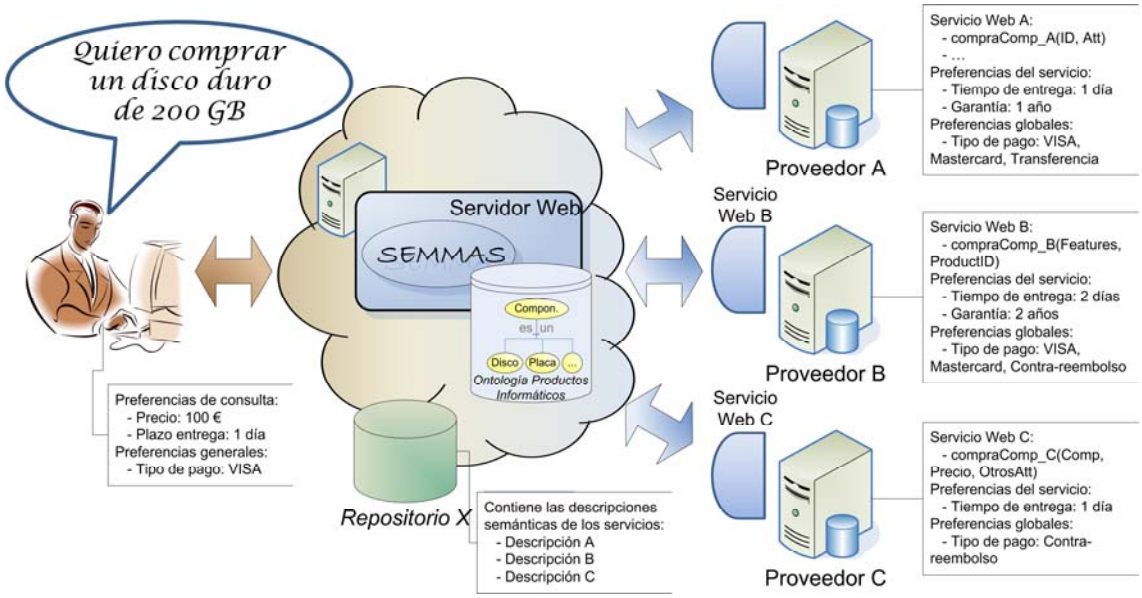


Fig. 159. Escenario de ejemplo para Comercio Electrónico

Según los procesos descritos en la sección anterior para llevar a cabo transacciones B2C y B2B, las funciones de la aplicación software desarrollada que se pretenden evaluar mediante este escenario de ejemplo son las siguientes:

- Identificación de proveedores y de servicios.
- Negociación y generación de contratos.
- Realización de las transacciones.
- Evaluación de los productos/servicios y cumplimiento del contrato.

En la nomenclatura utilizada a lo largo del documento, estas funciones corresponden a las tareas que se enumeran a continuación:

- Descubrimiento y selección de servicios.
- Negociación entre consumidores y proveedores.
- Invocación de los servicios.
- Monitorización.

VI.4.2. Adaptación del Prototipo a los Requisitos del Dominio

La aplicación software desarrollada ha sido adaptada para su aplicación en un entorno de Comercio Electrónico. La aplicación del prototipo en este ámbito, proporciona a los usuarios una interfaz Web única y amigable para los usuarios, a partir de la que pueden realizar consultas sobre los catálogos de productos de diversos proveedores de productos informáticos, sin tener que preocuparse por la posible ambigüedad existente entre las terminologías de los distintos catálogos. Adicionalmente, los usuarios, a través de la interfaz Web que proporciona la aplicación software, son capaces de iniciar la compra de alguno de los productos de los distintos proveedores.

Para llevar a cabo la adaptación de la plataforma a los requisitos del Comercio Electrónico en el dominio de los componentes informáticos, es preciso llevar a cabo los siguientes pasos:

1. Diseñar una ontología del dominio que contenga los conceptos y relaciones de interés para la aplicación a obtener.
2. Desarrollo de un Servicio Web por cada recurso de información/funcionalidad al que se pretende tener acceso.

3. Anotar semánticamente los Servicios Web haciendo uso de la ontología del dominio y de alguna de las aproximaciones actuales de Servicios Web Semánticos.
4. Instanciar los agentes necesarios del entorno multi-agente para acceder a los catálogos de proveedores de componentes informáticos a través de los Servicios Web que han sido descritos semánticamente.

En este apartado, se describen, en primer lugar, algunas de las características de la ontología del dominio que ha sido empleada en el experimento. Posteriormente, se discuten los detalles de los Servicios Web Semánticos implementados para permitir el acceso a los distintos proveedores de componentes informáticos. Finalmente, se presenta un ejemplo donde un usuario realiza una consulta sobre la aplicación software que, haciendo uso de la ontología del dominio diseñada y los Servicios Web desarrollados, es capaz de satisfacer las necesidades del usuario.

VI.4.2.1. Ontología del dominio

El experimento propuesto pretende permitir a los usuarios consultar catálogos de productos informáticos y, posteriormente, realizar la compra de los mismos. Para la elaboración del experimento, se ha diseñado y desarrollado una ontología parcial de componentes informáticos[58]. Esta ontología se ha desarrollado en OWL DL por medio de la herramienta Protégé. Además de conceptos y atributos (‘*datatype property*’ en la terminología de Protégé), se han incluido relaciones de dos tipos: taxonómicas (*ES_UN*) y mereológicas (*PARTE_DE*).

En la Fig. 160, se muestran los conceptos de primer nivel de la ontología de componentes informáticos. Se distingue entre los conceptos de ‘ordenador’ y de ‘componente de ordenador’. Se consideran dos clases de ordenadores, los ‘ordenadores personales’ y los ‘portátiles’. Un componente de ordenador es parte de un ordenador. El concepto ‘componente de ordenador’ (‘*Computer_component*’) tiene asociados los siguientes atributos: ‘identificador’, ‘descripción textual’, ‘modelo de componente’, ‘marca’, ‘precio’ y ‘garantía’. Por herencia, todas las subclases de ‘componente de ordenador’ incorporan estos atributos comunes. Adicionalmente, otros conceptos incluyen atributos específicos. Este es el caso, por ejemplo, del concepto ‘sistema de

[58] http://klt.inf.um.es/ontologies/computer_hardware.htm

almacenamiento' ('*computer_storage*'), que posee dos atributos específicos, la capacidad de almacenamiento y la velocidad de acceso al dispositivo.

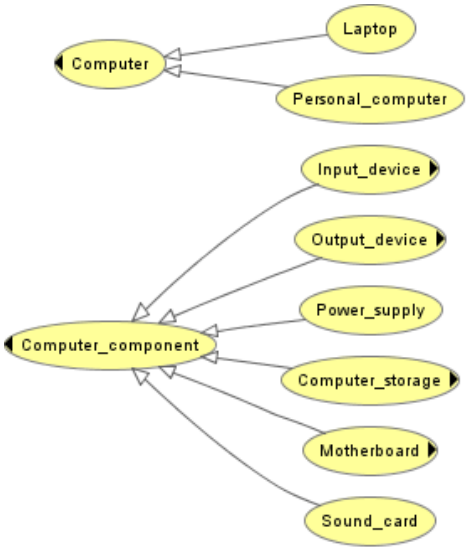


Fig. 160. Conceptos de primer nivel de la ontología de componentes informáticos

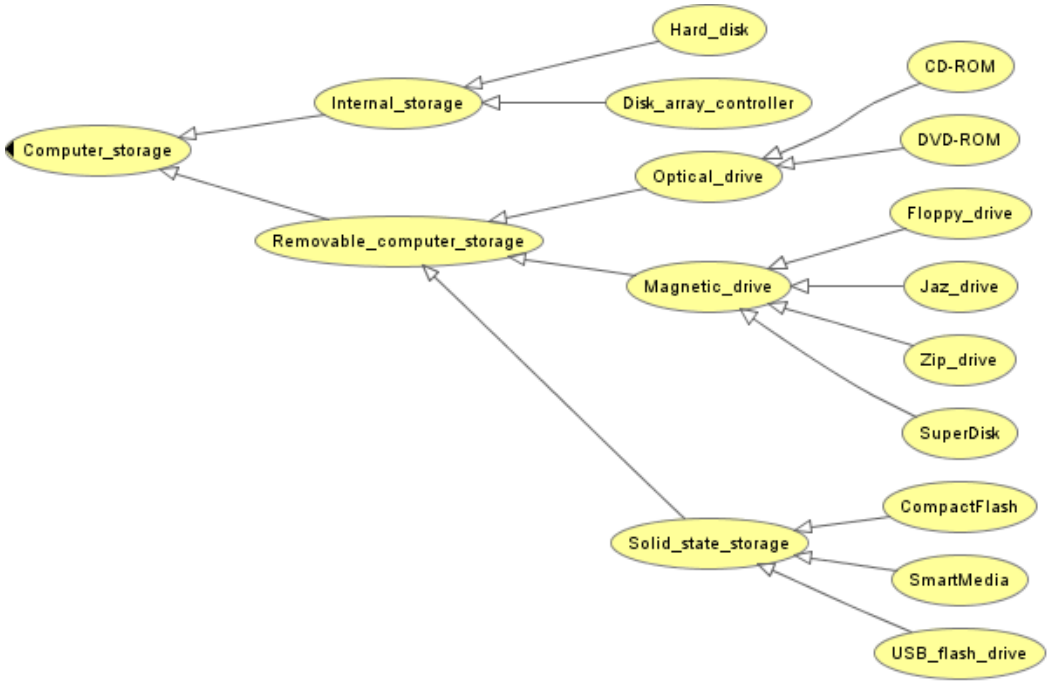


Fig. 161. Jerarquía de componentes para el almacenamiento de información

El diseño de la ontología de componentes informáticos, se basa fundamentalmente en la combinación de varias jerarquías de tipos de componentes. En la Fig. 161, se muestra la jerarquía relativa a los 'sistemas de almacenamiento'. Los sistemas de almacenamiento se dividen en dos grandes grupos, los denominados 'internos' ('*Internal_storage*'), y los 'de quita y pon' ('*Removable_computer_storage*'). Estos últimos se subdividen, a su vez, en tres subtipos, 'dispositivo óptico', 'dispositivo magnético', y los 'discos de estado sólido' (pendrive). Entre los internos, se incluyen dispositivos como el disco duro.

La relación de partes de un ordenador con el concepto 'ordenador', así como la de 'componentes de la placa base' con el concepto 'placa base', se realiza a través de relaciones mereológicas. En la Fig. 162, se muestra gráficamente un extracto de estas relaciones mereológicas entre las partes y el todo.

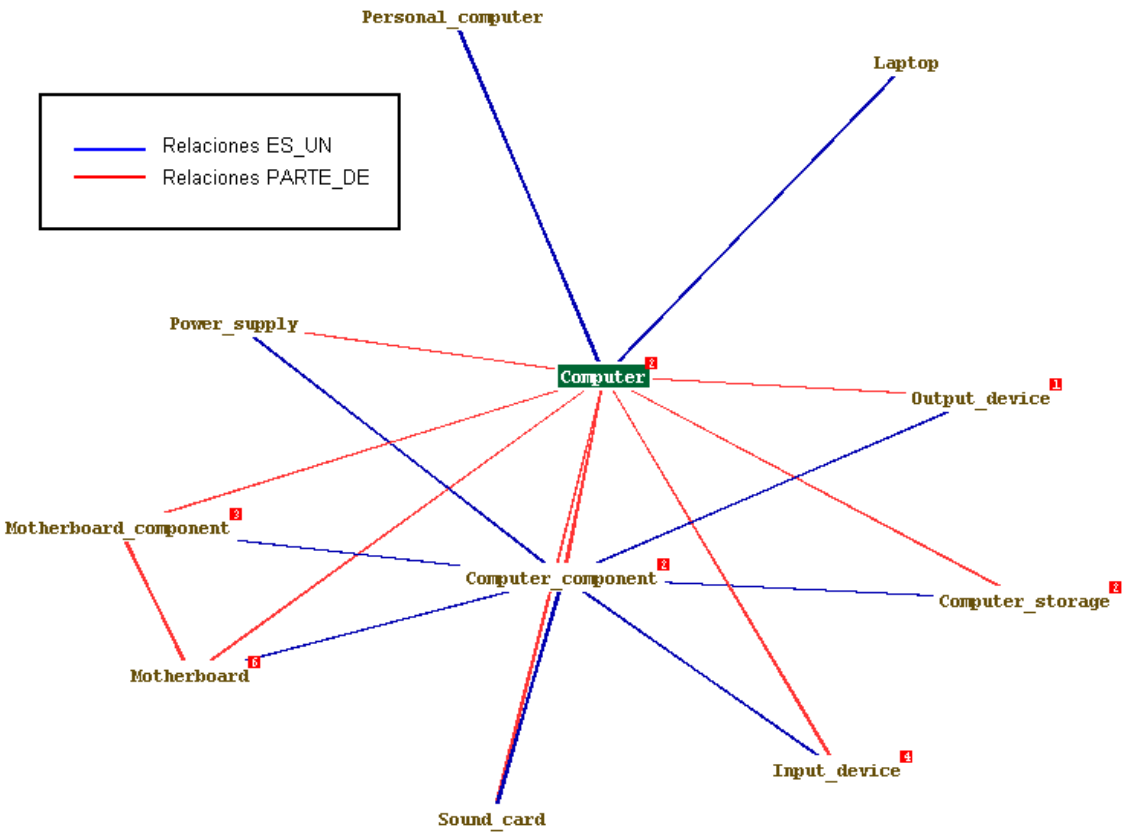


Fig. 162. Relaciones mereológicas: uniendo las partes al todo

La ontología contiene un total de 45 conceptos, 8 atributos, 43 relaciones taxonómicas, y 12 relaciones mereológicas. El experimento no pretende, por tanto,

simular un entorno real, ya que esto supondría la necesidad de disponer de una ontología del dominio mucho más detallada. Muy al contrario, mediante este experimento, y en base al escenario de ejemplo presentado, se pretende demostrar la aplicabilidad del prototipo propuesto como resultado de la investigación realizada en esta tesis doctoral, en entornos de comercio electrónico.

VI.4.2.2. Servicios Web y anotación semántica

Para permitir el acceso a los catálogos de los distintos proveedores identificados en el escenario de ejemplo y dar soporte a la realización de pedidos por parte de los usuarios, se ha desarrollado, para cada proveedor de componentes informáticos, un Servicio Web que simula la interacción con los procesos de negocio internos a los proveedores que satisfacen estas funcionalidades. Estos Servicios Web simplificados permiten simular, hasta cierto punto, transacciones B2C y B2B. Al igual que en los experimentos anteriores, los Servicios Web se han implementado en Java y haciendo uso de la librería Axis2 de Apache.

Cuando los servicios se han implementado, comienza el proceso de anotación semántica de las capacidades de los mismos. Esta descripción semántica se realizó haciendo uso del *plugin* de Protégé "*OWL-S Editor*", que permite la anotación de servicios mediante la especificación OWL-S.

A continuación, y a modo de ejemplo, se muestran algunos detalles de uno de los Servicios Web implementados y la descripción semántica del mismo. El diseño, la implementación y la anotación del resto de servicios para el escenario de ejemplo es muy similar, por lo que se omite su descripción.

Servicio Web Semántico A

El Servicio Web A (AWS), perteneciente al Proveedor A, está compuesto por varios métodos. Específicamente, el método del servicio que resulta de interés para el experimento actual, es el que permite al solicitante realizar una orden de compra:

- *compraComp_A(ID, Att)*:
    - Este método efectúa una orden de compra de un producto de la categoría indicada mediante el parámetro '**ID**', y cumple con la especificación de atributos establecida a través del parámetro '**Att**'.

Al igual que en los casos anteriores, las características sintácticas de las operaciones disponibles en el servicio se describen mediante un fichero WSDL (generado en Eclipse a partir del plugin '*Code Generator Wizard*'[59] de Apache). En las figuras Fig. 163 y Fig. 164, se muestran sendas porciones de este fichero WSDL asociado al Servicio Web AWS.

```
− <wsdl:definitions targetNamespace="http://klt.inf.um.es/">
    <wsdl:documentation> AWS </wsdl:documentation>
  − <wsdl:types>
        <...>
  − <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
      targetNamespace="http://model/xsd">
      <xs:element name="Product" type="ax24:Product"/>
    − <xs:complexType name="Product">
      − <xs:sequence>
          <xs:element name="description" nillable="true" type="xs:string"/>
          <xs:element name="id" nillable="true" type="xs:string"/>
          <xs:element name="model" nillable="true" type="xs:string"/>
          <xs:element name="price" type="xs:int"/>
          <xs:element name="trademark" nillable="true" type="xs:string"/>
          <xs:element name="type" nillable="true" type="xs:string"/>
          <xs:element name="warranty" type="xs:int"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>
    <...>
− <wsdl:portType name="AWSPortType">
  − <wsdl:operation name="compraComp_A">
      <wsdl:input message="axis2:compraComp_AMessage" wsaw:Action="urn:compraComp_A"/>
      <wsdl:output message="axis2:compraComp_AResponse"/>
    </wsdl:operation>
  </wsdl:portType>
    <...>
</wsdl:definitions>
```

Fig. 163. Extracto de fichero WSDL del servicio AWS

[59] http://ws.apache.org/axis2/tools/1_0/eclipse/wsdl2java-plugin.html

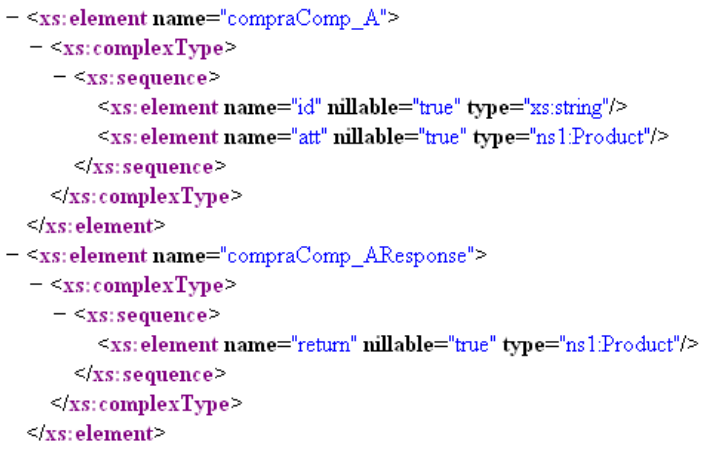


Fig. 164. Definición de los mensajes de entrada y salida de la operación *compraComp_A*

A continuación, se genera la descripción semántica de las capacidades del Servicio Web implementado, obteniendo así el Servicio Web Semántico A (ASWS) (ver Fig. 165). Para llevar a cabo este paso, es necesario hacer uso de la ontología de componentes informáticos descrita anteriormente. Siguiendo este procedimiento, se consigue evitar la ambigüedad entre los distintos catálogos de los proveedores por un lado, y entre éstos y el sistema por otro. Para cada método de los tres servicios desarrollados (a saber, AWS, BWS y CWS), se genera, a partir del plugin de Protégé '*OWL-S Editor*', un fichero OWL con la anotación semántica de la operación siguiendo la especificación OWL-S.

```
<?xml version="1.0"?>
<rdf:RDF
…
xml:base="http://klt.inf.um.es:8080/services/description/compraComp_
A.owl">
…
<service:Service rdf:ID="compraComp_AService">
  <service:presents>
    <profile:Profile rdf:ID="compraComp_AProfile"/>
  </service:presents>
  <service:describedBy>
    <process:AtomicProcess rdf:ID="compraComp_AProcess"/>
  </service:describedBy>
  <service:supports>
    <grounding:WsdlGrounding rdf:ID="compraComp_AGrounding"/>
  </service:supports>
</service:Service>
<profile:Profile rdf:about="#compraComp_AProfile">
  <service:presentedBy rdf:resource="#compraComp_AService"/>
  <profile:hasOutput>
    <process:Output rdf:ID="return">
      <rdfs:label>return</rdfs:label>
```

```
      <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://klt.inf.um.es/ontologies/computer_hardware.owl#Computer_compo
nent</process:parameterType>
    </process:Output>
  </profile:hasOutput>
  <profile:serviceName>compraComp_A</profile:serviceName>
  <profile:hasInput>
    <process:Input rdf:ID="id">
      <rdfs:label>id</rdfs:label>
      <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://www.w3.org/1999/02/22-rdf-syntax-ns#ID
      </process:parameterType>
    </process:Input>
  </profile:hasInput>
  <profile:hasInput>
    <process:Input rdf:ID="att">
      <rdfs:label>att</rdfs:label>
      <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://klt.inf.um.es/ontologies/computer_hardware.owl#Computer_compo
nent</process:parameterType>
    </process:Input>
  </profile:hasInput>
</profile:Profile>
…
</rdf:RDF>
```

Fig. 165. Extracto de la descripción semántica del servicio AWS

Los Servicios Web Semánticos ASWS, BSWS y CSWS, permiten a los agentes del sistema interactuar con los proveedores de componentes informáticos en representación de los usuarios humanos. Como se precisó anteriormente, en un entorno real de comercio electrónico, la organización que ofrece los productos debe ser la encargada tanto de la implementación de los Servicios Web, como de la anotación semántica de los mismos.

VI.4.2.3. Ejemplo de aplicación

Una vez definida la ontología del dominio, los Servicios Web dispuestos, y los agentes que constituyen el núcleo de la aplicación software instanciados (no existen requisitos especiales a este respecto), se elaboró un experimento consistente en que un usuario realiza consultas al sistema, que debe resolverlas mediante el uso de los distintos servicios disponibles. El escenario del caso de ejemplo se presentó en la Fig. 159 (pag. 328).

Para estudiar el comportamiento de la aplicación en este entorno, se estudia, paso a paso, el proceso de resolución de una consulta en particular. Se supone que un usuario del sistema, organización o persona física, quiere realizar la compra de una determinada

cantidad de discos duros. En esta situación, el primer paso a seguir por parte del usuario es indicar este deseo al sistema a través de una consulta. La consulta puede ser la siguiente: "Quiero comprar un disco duro de 200 Gb" (en inglés, "*I want to buy a 200 GB hard disk*"). Cuando la consulta ha sido realizada, el agente personal del usuario toma el control del proceso, y el usuario sólo debe esperar la respuesta por parte del sistema. A continuación, se describe la secuencia de pasos que lleva a cabo la aplicación software para conseguir los resultados de la consulta realizada por el usuario. Los pasos se han agrupado según su relación con los retos planteados en este escenario de ejemplo.

1. *Identificación de proveedores y de servicios*: descubrimiento y selección de servicios.

   1.1. El primer paso que lleva a cabo el sistema para determinar los servicios que pueden satisfacer los requisitos establecidos por el usuario, es procesar la consulta que ha realizado el usuario. Este análisis preliminar permite a la aplicación disponer de una representación interna y formalizada del objetivo planteado por el usuario. Para esto, el '*Customer Agent*' representante del usuario emplea la herramienta KAText, descrita en la sección V.2.2.4. La ontología que se obtiene tras esta primera fase se presenta en la Fig. 166. En la figura, se destaca la relación existente entre los conceptos que constituyen la ontología del objetivo, con los elementos del conocimiento que constituyen la ontología del dominio.

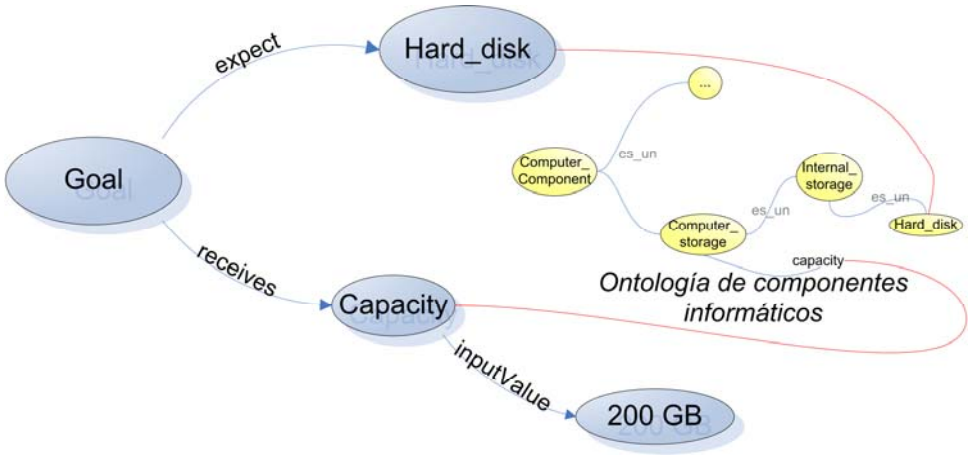


Fig. 166. Comercio Electrónico – ontología del objetivo resultante

1.2. Una vez que el objetivo del usuario ha sido representado formalmente, el agente personal lo envía a alguna de las instancias del agente '*Discovery Agent*' disponibles en la plataforma para que se proceda al descubrimiento de los servicios que pueden permitir satisfacer la consulta.

1.3. El '*Discovery Agent*', entonces, accede a los distintos repositorios a los que tiene acceso y, realizando consultas SPARQL sobre todos los servicios almacenados en dichos repositorios, identifica aquellos que pueden ser válidos conforme al objetivo perseguido. Como se describió en detalle en la sección V.3.2.2, el mecanismo utilizado para el descubrimiento se fundamenta en la correspondencia entre las salidas esperadas de los servicios descritos y la salida requerida por el objetivo. En este ejemplo, la salida del método '*compraComp_A*' del AWS (al igual que la de los métodos '*compraComp_B*' y '*compraComp_C*' en los servicios restantes) se ha anotado semánticamente con el concepto '*Computer_Component*' de la ontología de componentes informáticos. Por tanto, apoyado en las reglas de inferencia más básicas de OWL, el sistema concluye que un servicio que devuelve como salida elementos de '*Computer_Component*' es apropiado para resolver un objetivo que solicita elementos de '*Hard_disk*', una subclase de '*Computer_Component*'. En definitiva, el '*Discovery Agent*' determina que los métodos '*compraComp_X*' de los servicios de los tres proveedores presentes en el experimento pueden resolver con éxito la consulta del usuario.

1.4. Una vez finalizada la fase de descubrimiento, el '*Discovery Agent*' devuelve al agente personal del usuario la lista completa de servicios identificados. A continuación, este agente envía la lista de servicios procedente del '*Discovery Agent*' a alguno de los agentes '*Selection Agent*' presentes en la plataforma. En ese instante, comienza la segunda fase del proceso, dado que, para ordenar la lista de posibles servicios a ejecutar, es necesario llevar a cabo un proceso de negociación con cada uno de ellos.

2. *Negociación y generación de contratos*: negociación entre consumidores y proveedores.

2.1. El '*Selection Agent*' es el encargado de ordenar la lista de servicios según la utilidad que se espera de los mismos. Para calcular el valor de utilidad mencionado, el '*Selection Agent*' debe ponerse en contacto con los agentes '*Service Agent*' que representan a los servicios presentes en la lista. En particular, el '*Service Agent*' inicia un proceso de negociación con los agentes '*Service Agent*' pertinentes, donde éstos establecen las condiciones por las cuales los servicios que representan pueden ser ejecutados.

2.2. Cuando el agente que representa a uno de los servicios candidatos recibe la petición de condiciones por parte del '*Selection Agent*', éste se pone en contacto con el '*Provider Agent*' correspondiente (esto es, el que representa al proveedor que suministra el servicio) para solicitar las condiciones estratégicas que se tienen que cumplir para permitir la ejecución del servicio. Esta información, en combinación con las condiciones particulares del servicio, se utiliza para generar la propuesta que se envía al '*Selection Agent*' para su evaluación. En el caso particular del AWS, en la propuesta se indica que el tipo de pago permisible es mediante tarjeta de crédito (VISA ó Mastercard), el tiempo de entrega es de 1 día desde el momento de realizar el pedido, y el tiempo de garantía es de un año. Es importante señalar que, para que las mencionadas preferencias sean admisibles por el sistema, se ha tenido que modificar el modelo de preferencias para proveedores y consumidores (ficheros DTD correspondientes) y variar la implementación de diversos roles, entre ellos el rol '*Selector*', para su evaluación.

2.3. Una vez el '*Selection Agent*' ha recibido todas las propuestas de los agentes '*Service Agent*', las evalúa una a una, asignándoles un valor de utilidad. Para calcular el valor de utilidad, se aplica el algoritmo que se presenta a continuación:

```
utilidad=0
utilidad += (servicio.entrega – usuario.entrega) x 10 x (-1)
for (tipoPago : usuario.tipoPago) {
   if (servicio.tipoPago.exist(tipoPago))
      value += 30
}
```

Aplicando este algoritmo y utilizando los datos indicados en la figura, el valor de utilidad de cada servicio es como sigue:

- AWS: 30
- BWS: 20
- CWS: 0

2.4. En base a dicho valor de utilidad, el '*Selection Agent*' ordena la lista de servicios de mayor a menor utilidad y la envía de vuelta al '*Customer Agent*', que, finalmente, se la presenta al usuario (ver Fig. 167).

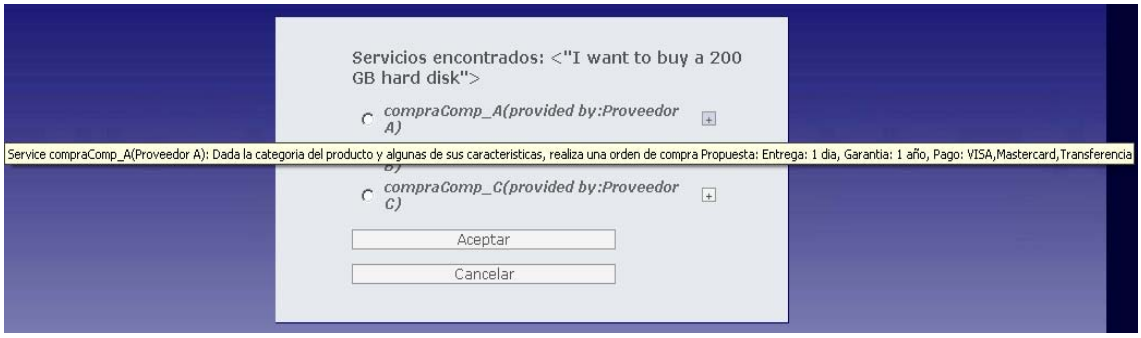


Fig. 167. Comercio Electrónico – selección de servicios

3. *Realización de las transacciones*: invocación de los servicios.

3.1. Cuando el usuario final selecciona el servicio deseado, el prototipo implementado debe poner los medios para llevar a cabo de forma efectiva la transacción, esto es, ejecutar el servicio escogido. El agente personal del usuario es el destinatario de la selección del usuario. Este agente, entonces, interactúa con el '*Service Agent*' pertinente (o sea, el que representa al servicio elegido), para realizar la invocación del mismo.

3.2. El '*Service Agent*', seguidamente, analiza la descripción semántica del servicio para determinar cuáles son los parámetros de entrada para la invocación del servicio y qué resultado debe esperar. Haciendo esto, el agente descubre que, como parámetro de entrada, la operación a ejecutar espera la categoría del producto a comprar y algunas características que limiten el rango de posibilidades. En el objetivo, está representada la categoría del producto deseado, así como una característica propia del mismo, como es la capacidad. Adicionalmente, entre las preferencias del usuario (transferidas por el

'*Customer Agent*') se identifica un parámetro que puede ser de interés para la invocación, como es el precio orientativo. Con esta información, el '*Service Agent*' realiza la invocación del servicio.

3.3. Cuando el '*Service Agent*' recibe el resultado de la operación, informa apropiadamente al agente personal del usuario, el cual muestra la información al usuario final para su consideración (ver Fig. 168).

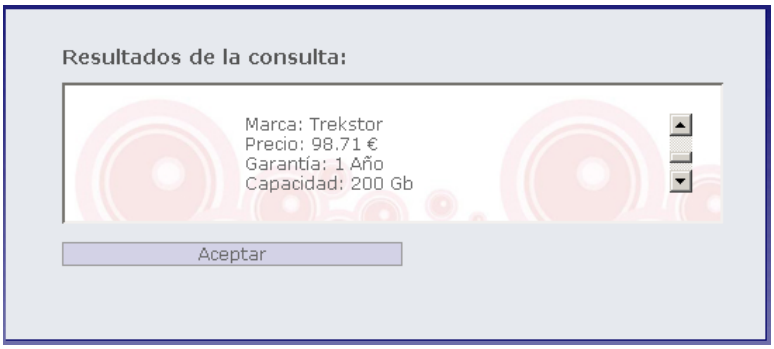


Fig. 168. Comercio Electrónico – resultado de la consulta

4. *Evaluación de los productos/servicios y cumplimiento del contrato*: monitorización

4.1. Mientras se está llevando a cabo la ejecución del servicio, se produce un proceso de monitorización, mediante el cual, el sistema se asegura de que no se producen fallos durante esta tarea, y se mantienen las condiciones de los acuerdos a los que se ha llegado durante la fase de negociación. Este proceso lo realiza el '*Framework Agent*' en conjunción con los agentes '*Customer Agent*' y '*Service Agent*'. En el ejemplo presentado, sin embargo, las condiciones establecidas en el contrato no pueden ser evaluadas durante la ejecución del servicio, de modo que el mecanismo de monitorización, en este caso, sólo se asegura de que no ocurra ninguna anomalía en la ejecución del servicio.

VI.5. Resumen

En este capítulo, se han presentado tres experimentos mediante los que se comprueba la capacidad del sistema desarrollado para hacer frente a la gran variedad de problemas que se plantean en los dominios explorados. En cada experimento, se plantea un escenario de ejemplo en el que el prototipo es aplicado a un dominio particular para

resolver una tarea en el mismo. Los tres dominios considerados son el Gobierno Electrónico, la Bioinformática y el Comercio Electrónico.

El primer experimento estudiado, relativo al dominio del Gobierno Electrónico y las Administraciones Públicas, se basa en un escenario donde el prototipo se ha aplicado para llevar a cabo, de forma automatizada, los trámites concernientes al cambio de residencia de un usuario. Para esto, el sistema explota, en primer lugar, su capacidad para descubrir y componer diversos servicios que, conjuntamente, alcanzan un objetivo determinado. Además, se comprueba la eficacia del sistema a la hora de invocar los servicios y de presentar los resultados obtenidos al usuario final.

El segundo experimento plantea otros retos adicionales. El dominio propuesto es el de la Bioinformática y, como cualquier otro campo dentro de la e-Science, supone la gestión de extensas cantidades de información y datos. Además, esta información se encuentra, generalmente, almacenada en fuentes de datos heterogéneas, de forma que el sistema tiene que lidiar con la posible ambigüedad que porten los datos. El escenario propuesto para este experimento supone el acceso a fuentes de datos reales a través de Servicios Web. A su vez, el domino se ha acotado al ámbito de los *oncogenes*, genes modificados o conjuntos de nucleótidos que codifican una proteína y que provocan el surgimiento de tumores malignos. Mediante este experimento, se ha comprobado la conveniencia del sistema para proporcionar acceso integrado a fuentes de datos heterogéneas.

El último experimento presentado, se refiere al dominio del Comercio Electrónico y las transacciones en entornos B2C y B2B. En estos escenarios, es muy común que se produzca la situación de que una innumerable cantidad de proveedores oferten servicios/productos similares o idénticos. En estos casos, por tanto, el mayor reto al que se debe enfrentar el sistema es la selección del proveedor que propone las mejores condiciones para satisfacer las pretensiones del usuario. De este modo, el proceso de negociación adquiere especial relevancia en esta situación. El escenario planteado para este experimento consta de tres proveedores de componentes informáticos que establecen determinadas condiciones asociadas a los servicios que suministran. En base a estas condiciones y a las preferencias del usuario, el prototipo es capaz de determinar el servicio que más se ajusta a los deseos del usuario. Para esto, existe una fórmula

mediante la que se calcula el valor de utilidad de cada servicio identificado. Además de la negociación y la selección de servicios, también se muestra el funcionamiento de mecanismos como el descubrimiento, la invocación y la monitorización de los servicios.

CAPÍTULO VII. CONCLUSIONES Y TRABAJO FUTURO

En este capítulo, se incluye un amplio resumen de lo descrito con anterioridad en este documento y se discuten las conclusiones finales. Asimismo, se enumeran brevemente algunas de las líneas a seguir tras la consecución del sistema desarrollado, mejoras propuestas y futuras vías de desarrollo.

VII.1. Conclusiones

Las tecnologías de agentes y de Servicios Web Semánticos, junto con la Ingeniería Ontológica, constituyen los pilares fundamentales de esta tesis doctoral. La tecnología de agentes, estudiada desde los años 70, surgió de la investigación en Inteligencia Artificial con el propósito de desarrollar aplicaciones con una tecnología que les permitiera tomar decisiones por sí mismas en lo que se refiere a lo que necesitan hacer para satisfacer sus objetivos de diseño. La tecnología de agentes en general, y la de los Sistemas Multi-Agente (SMA) en particular, ofrecen unas características muy 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. Los agentes inteligentes añaden a este tipo de sistemas propiedades como la autonomía, proactividad y comportamiento dirigido por objetivos, que permiten el desarrollo de numerosas actividades sin la necesidad de intervención humana. Adicionalmente, estos agentes pueden incorporar habilidades como el aprendizaje y el razonamiento que, junto a la aplicación de técnicas de Inteligencia Artificial, permiten mostrar un comportamiento inteligente y realizar las tareas de un modo óptimo. Sin embargo, y a pesar de las numerosas ventajas que supone la aplicación de esta tecnología, no son muchas las aplicaciones reales desarrolladas que exploten todo el potencial de la misma. El problema radica en que las plataformas multi-agente no están, en general, preparadas para comunicarse entre sí cuando es preciso traspasar las fronteras de las empresas, dado que el uso de protocolos no estándar limita las posibilidades de que los mensajes de interacción superen los cortafuegos existentes en las puertas de entrada de los sistemas software de las

organizaciones. Además, aunque el establecimiento de puentes de comunicación entre plataformas localizadas en distintos entornos organizacionales puede, en primera instancia, dar solución a este problema, implica la pérdida del dinamismo que caracteriza la Web, ya que estos puentes se deben establecer de forma estática en tiempo de diseño.

Por otra parte, la tecnología de los Servicios Web Semánticos es mucho más reciente. Surgió de la aplicación de las ideas procedentes del campo de la Web Semántica en el área de los Servicios Web. Esta tecnología se basa en la adición de contenido semántico formal y comprensible por sistemas computerizados a la descripción de las capacidades de los Servicios Web, de forma que entidades software puedan procesar estos servicios del mismo modo en que lo haría un humano. En la actualidad, no existe ningún lenguaje de especificación estándar que describa cómo anotar semánticamente las capacidades de los servicios. En cambio, el W3C está examinando cinco propuestas distintas con el objetivo de alcanzar un estándar en esta tecnología: OWL-S, WSMO, SWSF, WSDL-S y SAWSDL. Todas estas propuestas abarcan los elementos necesarios para permitir la automatización de las tareas básicas en torno a los Servicios Web, a saber, descubrimiento, selección, composición, invocación y monitorización. Para que esto se haga realidad, sin embargo, se supone la existencia de entidades software que incorporen capacidades para procesar el contenido semántico y razonar sobre el mismo. Se han desarrollado entornos de ejecución de Servicios Web Semánticos basados en componentes (p.ej. WSMX, IRS, METEOR-S, etc.) y que dan soporte, en mayor o menor grado, a la realización de estas tareas. No obstante, el uso de técnicas y herramientas procedentes de la tecnología de agentes en el diseño e implementación de este tipo de aplicaciones, podría mejorar considerablemente las capacidades de las mismas. Así, entre otras cosas, sería posible aplicar técnicas de aprendizaje para mejorar los resultados ofrecidos por la herramienta, así como hacer uso de mecanismos de negociación para permitir la aplicación de estos marcos de trabajo en entornos reales donde la competitividad es inherente.

De esto, se concluye que la combinación de agentes software y Servicios Web Semánticos puede dar lugar a la siguiente evolución de la Web. Este es un hecho generalmente aceptado por la comunidad investigadora que, sin embargo, no se pone de

acuerdo en el modo de llevarlo a cabo. Existen tres posibles escenarios para conseguir esta integración: (1) los Servicios Web proporcionan la funcionalidad básica mientras que los agentes hacen uso de estos servicios para proveer de funciones de valor añadido; (2) incorporar capacidades de agentes a los Servicios Web; y (3) crear puentes de comunicación entre ambos tipos de entidades, obteniendo un espacio heterogéneo. En lo que respecta a soluciones concretas, y en relación con las analizadas en esta tesis, el principal problema de GODO (Gómez et al., 2004; Gómez et al., 2006) radica en su restricción a la hora de interactuar con los Servicios Web Semánticos, haciendo imposible sacar provecho de las numerosas posibilidades que ofrece la tecnología de agentes para la consecución de tareas tales como descubrimiento, selección y composición de Servicios Web. Por su parte, SWF (Stollberg et al., 2004a; Stollberg et al., 2004b; Stollberg et al., 2005) está demasiado vinculado con WSMO, no contemplando, así, el amplio abanico de aproximaciones de Servicios Web Semánticos existentes en la actualidad y la inexistencia de un estándar global. El uso de agentes como "envoltorio" de Servicios Web, como proponen Buhler y Vidal (2005), conlleva también serias limitaciones como la necesidad de definir un flujo de trabajo estático que contradice el dinamismo natural de este tipo de sistemas. El Task Computing (Song et al., 2004; Masuoka et al., 2005), aunque es más cercano a la computación pervasiva, también contempla la utilización de servicios a partir de sus descripciones semánticas. Sin embargo, esta aproximación no hace uso de la tecnología de agentes, limitando, de este modo, la autonomía del sistema, así como obviando tareas tan esenciales en estos entornos como la negociación.En definitiva, del estudio de las distintas soluciones proporcionadas hasta el momento a este respecto, se concluye que ninguna de ellas es capaz de sacar el máximo provecho de los beneficios potenciales de la combinación de agentes con Servicios Web anotados semánticamente.

En esta tesis doctoral, se parte del hecho de que, conceptualmente, agentes inteligentes y Servicios Web (semánticos) fueron concebidos con propósitos totalmente dispares y, como tal, se entiende (como hipótesis) que deben permanecer en dos niveles de abstracción diferentes. La principal idea que fundamenta la tecnología de agentes no es que los agentes inteligentes sean capaces de proporcionar servicios, sino que aquellos son concebidos como entidades autónomas que incorporan inteligencia y capacidades

cognitivas que les permitan mostrar un comportamiento pro-activo orientado a objetivos y establecer procesos de interacción, ya sean competitivos o cooperativos, con otras entidades para satisfacer sus objetivos de diseño. Por su parte, los Servicios Web supusieron una nueva evolución en el mundo de la computación distribuida, y tienen el único propósito de proveer funcionalidad a partir de componentes software accesibles globalmente. Estas diferencias conceptuales justifican la necesidad de disponer de ambas tecnologías en un entorno integrado y dan pistas sobre los beneficios que se pueden obtener si se aplican a tareas con distinto nivel de abstracción para el desarrollo de sistemas complejos.

En base a esta hipótesis de partida, en esta tesis se ha desarrollado un marco de trabajo que hace uso de las tecnologías de agentes y de Servicios Web Semánticos para la elaboración de aplicaciones que puedan tratar con el dinamismo de la Web, al tiempo que se pueden beneficiar de características como la autonomía, el aprendizaje y el razonamiento. Éste es el punto en que cobra relevancia la Ingeniería Ontológica. Las ontologías son los componentes que permiten que la comunicación entre agentes y Servicios Web, situados a distintos niveles de abstracción, se produzca de forma fluida y sin interpretaciones erróneas. La arquitectura del marco de trabajo desarrollado consta, fundamentalmente, de un entorno multi-agente, un conjunto de bases de conocimiento y diversas interfaces que permiten al sistema comunicarse, de forma efectiva, con las entidades externas identificadas, a saber, Servicios Web y proveedores de servicios, entidades (usuarios) consumidores de servicios, y desarrolladores. El entorno multi-agente está constituido por un total de siete agentes inteligentes cuya funcionalidad viene determinada por los roles que asumen, en cada momento, durante la ejecución del sistema. Los agentes propuestos son el '*Customer Agent*', que representa a los usuarios consumidores de servicios; el '*Discovery Agent*', que se encarga de buscar los servicios que cumplan unas determinadas condiciones; el '*Selection Agent*', responsable de ordenar la lista de servicios de acuerdo con algún criterio de utilidad; el '*Provider Agent*', que representa a las entidades proveedoras de servicios; el '*Service Agent*', que actúa como envoltura de los Servicios Web, confiriéndoles facultades como la negociación y la ejecución de tareas intensivas en conocimiento; el '*Broker Agent*', que se encarga de resolver los problemas de interoperabilidad que surjan entre los

restantes componentes; y, finalmente, el '*Framework Agent*', que es el responsable de monitorizar el estado del sistema y tomar las acciones necesarias para abordar los problemas que puedan aparecer.

La funcionalidad que muestra cada uno de estos agentes depende, en gran medida, de los roles que éstos asuman. Se distinguen trece tipos de roles categorizados en dos grupos principales, a saber, los encargados de llevar a cabo la interacción con los Servicios Web, y los responsables del control de la plataforma. Cada agente, que está asociado con un núcleo básico de roles que le asegura el mantenimiento de su identidad, puede asumir, en tiempo de ejecución y de forma dinámica, alguno de los otros roles identificados y, por tanto, modificar su comportamiento.

Otro de los componentes clave del marco de trabajo, lo constituyen las ontologías, las cuales operan como vocabularios universales, de forma que, tanto Servicios Web como agentes, comparten la misma interpretación de los términos contenidos del universo del discurso (esto es, del dominio). De esta forma, el sistema se beneficia de características tales como interoperabilidad y reusabilidad propias de la tecnología ontológica para alcanzar con éxito los objetivos establecidos. En el marco de trabajo propuesto, se distinguen cuatro bases de conocimiento que albergan distintas ontologías: (i) ontología de la aplicación y del dominio, que representan los conceptos que constituyen el dominio y la aplicación sobre los que se utiliza el sistema; (ii) ontología de conocimiento local de agentes, que contiene los elementos de conocimiento de que hacen uso cada uno de los agentes; (iii) ontología de negociación, que incorpora una representación de los protocolos y estrategias de negociación que maneje la aplicación; y (iv) ontología de Servicios Web Semánticos, que aloja la descripción semántica de las capacidades de los distintos servicios a los que tiene acceso la plataforma.

Por último, en lo que se refiere a la arquitectura del marco de trabajo, se pueden destacar las interfaces de que dispone el sistema para su interacción con las entidades externas al mismo, entre las cuales se encuentran: (i) los desarrolladores de aplicaciones, responsables de ajustar el marco de trabajo a las necesidades y requerimientos particulares de un dominio específico; (ii) los servicios, que proporcionan los niveles más básicos de funcionalidad de que hace uso el marco, y los proveedores de servicios, que, opcionalmente, pueden hacer uso de la plataforma y

beneficiarse de las ventajas que ello conlleva (p. ej. aplicación de sus directrices estratégicas en la provisión de los servicios que suministran); y (iii) los usuarios consumidores de servicios, que acceden al sistema para encontrar respuesta a una necesidad determinada.

El marco teórico desarrollado se ha evaluado en base a tres escenarios imaginarios en el dominio del Comercio Electrónico. Cada uno de estos escenarios plantea una serie de dificultades a las que se debe enfrentar la plataforma, mostrando, así, su utilidad en diversas situaciones. En particular, mediante estos escenarios de ejemplo, se comprueba el comportamiento del marco de trabajo en situaciones donde el descubrimiento, la selección y la invocación de servicios son fundamentales, así como en entornos que precisan la composición de varios servicios para satisfacer una necesidad determinada. Adicionalmente, se ha evaluado la capacidad del marco teórico definido para proporcionar acceso integrado a fuentes de datos heterogéneas, donde aparece la dificultad añadida de la ambigüedad semántica de los recursos accedidos. Para todas estas tareas, la última especialmente, es fundamental la aportación de las ontologías que permiten dirimir las dificultades de la heterogeneidad y la interoperabilidad.

Partiendo de la base teórica del marco de trabajo diseñado, se ha implementado una aplicación software que abarca la mayor parte de la funcionalidad concebida por parte del marco de trabajo propuesto. Esta aplicación, desarrollada en Java, se ha implementado sobre la plataforma multi-agente Jade, que proporciona los componentes básicos para la elaboración de SMAs conforme al estándar FIPA. Sobre esta plataforma, se han desarrollado los siete agentes identificados y se ha proporcionado una implementación básica de cada uno de los roles. La librería Jena, así como el repositorio Sesame, se han utilizado para el uso eficaz, la gestión y la explotación de las ontologías. Otras librerías de interés son KAText, que dota a los agentes de la capacidad de procesar texto en lenguaje natural, Axis2, que proporciona los medios básicos para la interacción con los Servicios Web, y un planificador del tipo STRIPS, que es utilizado en la fase de composición de servicios. Por otro lado, la interfaz del prototipo se ha desarrollado en forma de una aplicación Web. Para la implementación de la aplicación Web, se ha aplicado la tecnología Java en torno a JSP y los Servlets, y el contenedor de aplicaciones Tomcat. La comunicación entre la aplicación Web y el entorno multi-

agente se produce a través de la librería JadeGateway, que proporciona un mecanismo mediante el cual las acciones que realiza un usuario sobre la interfaz Web le llegan convenientemente al agente encargado de tratarlas.

Finalmente, el prototipo implementado fue testado empíricamente sobre tres escenarios de ejemplo. En cada uno de los experimentos realizados, la aplicación proporciona una funcionalidad determinada que viene determinada por los Servicios Web a los que tiene acceso. Dichos servicios, en unos casos han sido implementados sobre plataformas reales, dando acceso a información y funcionalidad suministrada por un proveedor determinado, y en otras ocasiones son meras simulaciones del mundo real. El primer experimento se ha planteado sobre el dominio de la Administración Digital (eGovernment). En este dominio, el principal problema con el que se enfrenta el prototipo es la complejidad de los trámites administrativos que, generalmente, involucran a diversas administraciones públicas. Por tanto, con el uso del sistema en este entorno, queda patente su utilidad para los procesos de descubrimiento y composición de Servicios Web. En el segundo experimento, el prototipo se ha aplicado en un entorno basado en Bioinformática. El mayor reto en el dominio de la Bioinformática es la cantidad y heterogeneidad de los datos existentes. Con este experimento, se demuestra la conveniencia del sistema para proporcionar acceso integrado a fuentes de datos heterogéneas. En el último experimento, se aplica el prototipo al dominio del Comercio Electrónico. El sistema, en este dominio, es el encargado de llevar a cabo, de forma exitosa, transacciones B2C y B2B. El problema principal de este tipo de entornos es la gran cantidad de servicios existentes que proporcionan una misma utilidad (producto o servicio). De este modo, el prototipo explota las capacidades de negociación y selección para determinar el servicio o servicios que mejor se ajustan a las pretensiones del usuario.

En conclusión, la principal diferencia entre la solución aportada a través de esta tesis doctoral y otras aplicaciones desarrolladas previamente, radica, fundamentalmente, en la asunción de que, habiendo sido concebidas con propósitos bien diferenciados, las tecnologías de agentes y de Servicios Web (semánticos) deben permanecer en dos niveles de abstracción diferentes. Partiendo de esta hipótesis, se ha diseñado un entorno de trabajo que, a diferencia de los desarrollados hasta el momento, es capaz de explotar

al máximo el potencial de sus tecnologías constituyentes (esto es, tecnología de agentes y de Servicios Web). Para esto, se propone una solución no intrusiva basada en el uso intensivo de ontologías como mecanismo de representación de conocimiento, que permite una comunicación fluida y eficaz entre las distintas entidades que componen el entorno. De esta forma, se consigue una complementariedad que, por un lado, conduce a la acotación de los problemas y dificultades de las tecnologías mencionadas, y, por otro, permite la explotación de los aspectos positivos y beneficiosos de cada una. Además, con este enfoque, se superan las limitaciones identificadas en las soluciones desarrolladas hasta el momento.

VII.2. Líneas Futuras

Por lo que respecta al trabajo futuro, se pueden señalar diferentes aspectos en los que el marco de trabajo y, sobre todo, el prototipo implementado se pueden mejorar. En los siguientes apartados, se enumeran las principales líneas a seguir a partir del trabajo desarrollado en esta tesis doctoral.

VII.2.1. Grid Computing

La computación GRID es una evolución de la Computación Distribuida, que se basa en el procesamiento de programas o partes de programas, de forma paralela, en uno o más nodos computacionales que se comunican entre sí mediante algún tipo de red de comunicación (Salt Cariols et al., 2007). El GRID da soporte a las organizaciones para que, mediante estas tecnologías, sean capaces de compartir recursos informáticos, generalmente distribuidos geográficamente y conectados por Internet, para hacer frente a diferentes necesidades.

Las tecnologías GRID intentan resolver problemas actuales de la Sociedad de la Información (Salt Cariols et al., 2007), tales como: (i) acceso rápido a bases de datos y almacenamiento; (ii) proporcionar su procesamiento y análisis utilizando potencia de cálculo distribuido; y (iii) utilización intensiva de la red.

Existe una estrecha relación entre la computación GRID y los Servicios Web. Así, los servicios GRID utilizan, como fundamento tecnológico, Servicios Web. Este vínculo ha llevado a algunos investigadores a plantear la posibilidad de, al igual que ocurrió en el

campo de los Servicios Web, asociar los servicios GRID con anotaciones semánticas de modo que entidades software automatizadas sean capaces de hacer un uso efectivo de estos servicios. En este contexto surgió S-OGSA (Corcho et al., 2006), una arquitectura de referencia para el GRID semántico que da soporte a la gestión explícita de semántica, y define los servicios de conocimiento asociados para permitir un amplio espectro de capacidades. El Grid Semántico, por tanto, se trata de una extensión del Grid en el que se exponen y manejan metadatos sobre los recursos, que se pueden compartir y gestionar por medio de protocolos GRID.

Debido a las similitudes entre los Servicios Web Semánticos y el GRID Semántico, es evidente pensar que la funcionalidad y/o aplicabilidad del marco de trabajo presentado puede ser similar en un entorno GRID, al compuesto por Servicios Web. Como trabajo futuro, se pretende investigar este extremo haciendo las modificaciones pertinentes en los componentes que constituyen el sistema implementado.

VII.2.2. Integración de datos

Como se destaca a lo largo del documento, si bien el sistema desarrollado en esta tesis doctoral es capaz de proporcionar acceso integrado a fuentes de datos heterogéneas, no incorpora los mecanismos necesarios para tratar los problemas relacionados con la integración de datos. La integración se puede definir como el proceso de combinación de datos que se encuentran almacenados en diferentes repositorios, de forma que se ofrezca al usuario una visión unificada de estos datos.

Cuando, como ocurre en el contexto del marco de trabajo desarrollado en esta tesis, la integración conlleva el uso de ontologías, se habla de integración de datos basada en ontologías. Este tipo de integración supone el uso de ontologías para combinar, de forma efectiva, datos o información que se encuentra disponible en múltiples repositorios heterogéneos.

En general, se pueden distinguir distintos tipos de heterogeneidad entre los datos procedentes de múltiples fuentes[60], a saber, heterogeneidad sintáctica (esto es, diferencias en el formato de representación de los datos), heterogeneidad esquemática o estructural (difiere el modelo o estructura del almacenamiento de los datos),

[60] http://en.wikipedia.org/wiki/Ontology_based_data_integration

heterogeneidad semántica (o sea, diferencias en la interpretación del significado de los datos), y heterogeneidad del sistema (cuando se emplean distintos sistemas operativos). Las ontologías ayudan a tratar la heterogeneidad semántica, aunque la existencia de múltiples ontologías también puede suponer una fuente de problemas. En particular, una de las soluciones típicas para la integración de datos a partir de ontologías se basa en la utilización de varias ontologías, una por cada fuente de datos a modelar, que se utilizan en combinación para la integración. Sin embargo, cada ontología puede modelar los mismos datos de distinto modo, produciéndose nuevamente una situación indeseable. Las soluciones a este tipo de problemas se basan en técnicas de mezcla de ontologías (*merging*) y definición de reglas de mapeo (*mapping*) que definan las correspondencias entre unas y otras. Como trabajo futuro, se pretende incorporar las técnicas y mecanismos adecuados para resolver el problema de la integración de datos del modo más automatizado que sea posible.

VII.2.3. Composición de servicios

La composición de servicios ha sido estudiada en el campo de la Inteligencia Artificial durante numerosos años. Algunas soluciones para la composición requieren una especificación explícita del flujo de control entre los servicios básicos para generar servicios de valor añadido. Este es el caso del "Lenguaje de Ejecución de Procesos de Negocio" (*Business Process Execution Language*, BPEL), un lenguaje basado en XML diseñado para el control centralizado de la invocación de diferentes servicios Web, con cierta lógica de negocio añadida que ayudan a la programación en gran escala (OASIS WSBPEL TC, 2007). Otro ejemplo de este tipo de soluciones es YAWL (*Yet Another Workflow Language*), basado en redes de flujos de trabajo (como las Redes de Petri), que añade una semántica formal (van der Aalst & ter Hofstede, 2005).

Otra tendencia en la investigación acerca de la composición de servicios, es la aplicación de técnicas de planificación basadas tanto en demostradores de teoremas (p. ej., ConGolog (McIlraith & Son, 2002)), como en planificación jerárquica de tareas (p.ej., SHOP-2 (Wu et al., 2003)). La ventaja de este tipo de soluciones es que no se requiere un modelo del proceso predefinido, pero, en su contra, asumen que las descripciones de los servicios relevantes se encuentran cargadas desde un inicio y no se precisa realizar procesos de descubrimiento durante la composición.

Si bien el marco teórico propuesto en esta tesis doctoral no asume ningún modelo predefinido para el procesamiento de esta tarea de composición de servicios, dejando libertad al desarrollador para incorporar la que mejor considere, en el prototipo implementado se ha elaborado un mecanismo basado en la descomposición de los objetivos mediante un planificador tipo STRIPS y la posterior búsqueda de servicios para cada uno de los subobjetivos. Tal y como ha sido desarrollado, este mecanismo no permite la definición de flujos de trabajo complejos y se fundamenta en la ejecución secuencial de los servicios. Adicionalmente, se contempla la posibilidad de que, si un agente es incapaz de invocar un servicio por la falta de alguno de los parámetros de entrada, y el valor de ese parámetro se obtiene como resultado de algún otro servicio, entonces una vez el valor ha sido obtenido como consecuencia de la ejecución de este último servicio, se procede a la invocación del primero.

Sin embargo, esta solución carece de utilidad en entornos reales donde deben contemplarse flujos de trabajo más complejos. Por este motivo, se plantea como trabajo futuro la definición de flujos de trabajos mediante alguno de los lenguajes de flujos de trabajo existentes (como WS-BPEL, YAWL, Redes de Petri, etc.), que ofrecen un mayor poder de expresividad.

VII.2.4. Mejora del prototipo

En esta tesis doctoral, se citan varios tópicos que no han formado parte del núcleo de investigación de la misma, pero que son objeto de investigación por numerosos científicos en todo el mundo. Así, por ejemplo, existen investigadores dedicados exclusivamente a tratar el tema del descubrimiento en entornos de Servicios Web Semánticos, la selección de servicios y la definición de propiedades no funcionales, la composición de Servicios Web para generación de flujos de trabajo, la integración de datos, etc. Esta tesis, lejos de tratar de encontrar solución para cada uno de estos problemas, asume la existencia de soluciones para éstos, y permite la integración de las mismas en un marco de trabajo que, una vez combinadas de forma adecuada, se convierte en una poderosa aplicación capaz de adaptarse a los requisitos y condiciones de diversas aplicaciones en dominios dispares.

Asimismo, el marco teórico desarrollado plantea, por sí mismo, la combinación de una gran cantidad de funciones provenientes tanto del área de agentes (p.ej.,

comportamiento basado en objetivos, autonomía y proactividad, negociación, aprendizaje e inteligencia, etc.), como de los Servicios Web Semánticos (descubrimiento, composición, selección, invocación, monitorización, razonamiento sobre ontologías, etc.). Sin embargo, para simplificar el proceso de implementación del prototipo y disminuir el nivel de complejidad del mismo, y con el objetivo de ser capaces de ilustrar las propiedades básicas del entorno y las ventajas que éste aporta sobre las soluciones desarrolladas hasta el momento, se ha omitido, deliberadamente, la implementación de algunas de las características secundarias propuestas en el marco teórico. Como trabajo futuro, se podrían incorporar las siguientes características al prototipo actual:

- Definición de un perfil del usuario y administración del historial de interacciones del usuario con el sistema. Con estos elementos básicos, es posible desarrollar un sistema de recomendación que puede ser de mucha utilidad en algunos de los dominios en los que se plantea utilizar la herramienta. Un ejemplo típico proviene del dominio de Comercio Electrónico. El sistema puede presentar al usuario recomendaciones concernientes a posibles productos en los que puede estar interesado en base a los productos adquiridos por este mismo usuario en pasadas interacciones con el sistema.
- Utilización de las capacidades de razonamiento propias de Jena, o aquellas más sofisticadas que puedan aportar los razonadores sobre lógica descriptiva que se pueden utilizar a través de Jena (p.ej., Pellet, Racer, KAON2), que permita explotar todas las posibilidades asociadas a disponer de conocimiento representado en forma de ontologías. De esta forma, la eficacia de tareas como el descubrimiento, que en la actualidad se resuelven a partir de consultas SPARQL, se podrían mejorar considerablemente.
- Incorporación de mayor flexibilidad en la comunicación de los agentes y en el modo de proceder de los mismos. Sobre este enunciado general, existen varias consideraciones:
    - Disponer de un mecanismo de negociación dinámico. En el marco teórico, se contempla la existencia de una ontología que modela los componentes que constituyen un mecanismo de negociación, a saber, protocolos y

estrategias de negociación. La justificación para esto es que, en distintos dominios, son distintas las estrategias y protocolos de negociación que resultan óptimos. Mediante una ontología en que se definen varios mecanismos de negociación, se dota al sistema de la flexibilidad necesaria para determinar, en tiempo de ejecución, qué mecanismo de negociación utilizar ante un problema determinado. El prototipo, en cambio, sólo hace uso del protocolo de negociación "*Contract-Net*", que forma parte de las especificaciones de FIPA.

- Dar soporte a la negociación entre los agentes a gran escala. Actualmente, el prototipo sólo concibe procesos de negociación entre los agentes '*Selection Agent*' y '*Service Agent*' a la hora de llevar a cabo la tarea de selección de servicios. Pueden existir en el mundo real, sin embargo, entornos competitivos en los que se produzcan situaciones donde sea conveniente permitir, como se plantea en el marco teórico, la negociación entre todos los tipos de agente.
- Permitir la formación de coaliciones entre varios agentes de un mismo tipo. Esta estrategia, puede ser particularmente interesante en agentes '*Customer Agent*' que compartan objetivos similares y cuya cooperación podría revertir en la mejora de la experiencia del cliente.

- Mejora de la experiencia del usuario. En un futuro, y tras sucesivas mejoras que se correspondan con lo descrito en el marco teórico, se espera que el sistema incorpore nuevas técnicas para mejorar la experiencia de los usuarios que decidan registrarse. Como ejemplo, para el futuro queda pendiente la inclusión de un sistema de generación de texto a partir de ontologías. Este mecanismo, similar al ya descrito en sistemas como Ontopath[61] o GINO (Bernstein & Kaufmann, 2006), puede ser de interés para "usuarios no expertos" ya que aporta una guía útil en relación a los términos y expresiones a utilizar en el momento de generar una consulta. Asimismo, el desarrollo de una solución de esta índole supondría una mejora en la precisión y eficiencia con respecto al procesamiento de texto en lenguaje natural. En la entrada guiada por ontologías, el usuario está restringido a

[61] http://oogl.snu.ac.kr/desc/index.html

incluir conceptos y relaciones que formen parte de una o varias ontologías (en el caso de la aplicación en cuestión, se trataría de las ontologías de aplicación y del dominio). De esta forma, se excluye la posibilidad de introducir consultas incoherentes y se facilita al sistema el proceso de generación de objetivos.

CAPÍTULO VIII. RESUMEN EN INGLÉS / SUMMARY IN ENGLISH

The Intelligent Agents (Wooldridge, 2002) topic has been broadly studied over the last 30 years and it is currently being revisited due to its relation to the Semantic Web (Berners-Lee et al., 2001) and the potential benefits that can be reached from both approaches' (i.e., Intelligent Agents plus Semantic Web) integration. Before the emergence of the Semantic Web, agents had to face the problems derived from the lack of structure in the information published on the Web. Nowadays, the semantically annotated information of the Semantic Web can be automatically processed by agents and new powerful opportunities open up for both application developers and users.

On the other hand, Web Services (Booth et al., 2004) arose as the best solution for remote execution of functionality. This was partly due to properties such as operating system and programming language-independence, interoperability, ubiquity and the possibility to develop loosely-coupled systems. However, as the Web grows in both size and diversity, there is an increased need to automate aspects of Web Services such as discovery, execution, selection, composition and interoperation (Fensel & Bussler, 2002). Semantic Web Service technology, that is, the semantic annotation of services' capabilities, has been the solution proposed (Lara et al., 2003).

Both Intelligent Agent and Semantic Web Service technologies are able to reach remarkable achievements and in some cases have overlapping functionalities. Nevertheless, they have different problems that limit their functionality when applied separately. It is widely recognised that integrating these two technologies in a joint environment can overcome their problems while strengthening their advantages (Berners-Lee et al., 2001; Hendler, 2001; Buhler & Vidal, 2003; Paolucci & Sycara, 2003; OWL-S, 2004; Gómez et al., 2006). Current approaches on combining Intelligent Agent and Semantic Web Services exhibit shortcomings mainly due to their inability to completely benefit from the advantages of this combination. In fact, they restrain the possibilities by considering only some of the properties of either Intelligent Agents or Semantic Web Service technology.

The reasons explained in the previous paragraphs were the basic motivations for the accomplishment of this research work. The solution proposed in this thesis stems from a basic underlying hypothesis: Intelligent Agents and (Semantic) Web Services were conceived for quite different purposes, and so, they must lie on different abstraction layers. The main idea behind Agent Technology was not for Intelligent Agents to be able to provide services, but to act as autonomous entities that incorporate intelligence and cognitive capacities, which allow them to show a goal-oriented pro-active behaviour and to establish, either competitive or cooperative, interaction processes with other software entities in order to satisfy their design objectives. On the other hand, Web Services involved a further evolution in Distributed Computing and their only purpose is to provide worldwide-accessible functionality. These conceptual differences between Agent Technology and (Semantic) Web Services Technologies lead to the need to have both technologies working in an integrated environment and glimpse the advantages of this combination in the development of complex systems.

With all, the main objective of the work presented here is to obtain a software application, which, by using intelligent agents and ontologies, improves the Web Services consumption performance and increases the automation of Web Service management-related tasks, so that the Web dynamism is fully exploited. In other words, the ultimate goal is the development of a knowledge-supported, agent-based Semantic Web Service execution environment. In order to achieve this objective, the following methodology has been followed:

1. Analysis of the state of art in Ontological Engineering, Agent Technology, and Semantic Web Services.The study of Semantic Web, logic formalisms, logic reasoning and inference engines is also included in this analysis, given their connection with some of these technologies.
2. In-depth analysis of current methodologies for Multi-Agent Systems design and development.
3. In-depth analysis of current work on Intelligent Agents and Semantic Web Services integration.
4. Definition and formalisation of a knowledge-based, multi-agent framework for dynamic access and automated execution of Semantic Web Services. Such

framework has been conceived for being independent of both, the domain (the domain knowledge is represented by means of ontologies) and the specific application to be developed in that domain.

5. Design and implementation of a software application to dynamically execute services from (human) users' abstract goals. A web interface for facilitating the use of the application from Web browsers has been provided. By adapting the framework, software developers can make it work in different domains and accomplish various tasks.
6. Test and validation of the software application. It is checked that the software application complies with the requirements that have been stated during the definition and formalisation stage. For this, the software application is applied in three different domains and the outcome evaluated.

The previous issues have been successfully accomplished, and the results obtained are presented in this thesis. Let us now describe how each objective was achieved.

STATE OF THE ART

This work's cornerstone foundation technologies are Ontologies, Multi-Agent Systems, and Semantic Web Services. The state of the art in these three research fields has been analyzed.

Agent Technology

The intelligent agents and multi-agent systems area has received ever-increasing attention by researchers over the last few years. This field emerged due to the promising benefits of having applications with a technology that allows systems to decide for themselves what they need to do in order to satisfy their design objectives. *Agents* are the computer systems in charge of carrying out this task. A common accepted definition of the term 'Agent' determines that an agent is a computer system situated in some environment and capable of autonomous action in this environment in order to meet its design objectives (Wooldridge, 2002). This author also highlights that an agent has to fulfil some properties in order to become intelligent: *reactivity* (i.e. the ability to perceive its environment and respond to changes in it in a timely fashion), *pro-activeness* (i.e. the ability to exhibit goal-directed behaviour by taking the initiative), and *social ability* (i.e. the ability to interact with other agents).

This is what Wooldridge and Jennings called the weak notion of agency (Wooldridge & Jennings, 1995). In fact, intelligent agents can present some other properties such as temporal continuity (i.e. an agent functions continuously and unceasingly), reasoning (i.e. decision-making mechanism, by which an agent decides to act on the basis of the information it receives, and in accordance with its own objectives to achieve its goals), rationality (i.e. an agent's mental property that attract it to maximize its achievement and to try to achieve its goals successfully), veracity (i.e. mental property that prevents an agent from knowingly communicating false information), mobility (i.e. the ability for a software agent to migrate from one machine to another), etc. (Elamy, 2005). In particular, one property that is often attached to the agent concept is that of *learning ability*, that is, the capacity to adapt or modify its behaviour by means of learning processes. In this work, Wooldridge's definition of intelligent agent has been adopted.

Agents can be useful as stand-alone entities that are delegated particular tasks on behalf of a user. However, in the majority of cases agents exist in environments that contain other agents, constituting multi-agent systems (MASs). A MAS can be seen as a system consisting of a group of agents that can potentially interact with each other (Vlassis, 2003). MASs present several advantages over isolated agents, such as reliability and robustness, modularity and scalability, adaptivity, concurrency and parallelism, and dynamism (Elamy, 2005).

When a group of individual agents form a MAS, the presence of a mechanism to coordinate such a group as well as a communication language becomes of key importance. Particularly, agents can either cooperate (if they have the same global objective) or compete (when they have different yet conflictive objectives). Along with this idea, Wooldridge distinguishes between *distributed problem solving systems* (constituted by agents explicitly designed to cooperatively achieve a given goal) and *open systems* (agents developed by different people to achieve possibly different objectives) (Wooldridge, 2002). For cooperative agents, the most typical cooperation mechanisms are: organizational structures, (centralized and distributed) multi-agent planning, contract nets and functionally exact cooperation. Meanwhile, competitive agents need a negotiation mechanism. Coalition formation, market mechanism, the bargaining theory, vote, auctions and task assignment between two agents are different negotiation mechanisms. Agent Communication Languages have been developed so that agents can carry out negotiation processes. Examples of such languages are KQML (Knowledge Query and Manipulation Language) and FIPA-ACL (Foundations for Intelligent Physical Agents – Agents Communication Language).

Agent-oriented programming is an emergent MAS-based programming paradigm. It seems appropriate for developing systems that operate in complex, dynamic, and unpredictable environments such as air traffic control, autonomous-spacecraft control, health care, and industrial-systems control. Along with this new programming paradigm and due to the increasing complexity of agent systems, it appeared what is called *Agent-Oriented Software Engineering* (AOSE) (Jennings & Wooldridge, 2000). The main purposes of AOSE are to create methodologies and tools that enable inexpensive development and maintenance of agent-based software. Several design methodologies

have been proposed to facilitate the development process of MASs. These methodologies are mainly based on traditional software methodologies that are extended to support this new programming paradigm. The most accepted ones are MAS-CommonKADS, ZEUS, GAIA, MaSE, and INGENIAS. For this work purposes, the INGENIAS methodology (Gomez-Sanz & Fuentes, 2002) has been applied because of its completeness and tool support. A broad study and comparison among these and other methodologies can be found in (Gomez-Sanz & Pavon, 2004).

Nowadays, efforts toward the standardization of agent technologies are being produced. Organizations such as FIPA[62] and OMG Agent PSIG[63] are leading this process. In particular, FIPA has become an IEEE Computer Society standards organization aimed at producing standards for the interoperation of heterogeneous software agents. FIPA has developed some specifications with a group of normative rules that permit an agent society to operate among themselves. This model identifies some necessary agent's roles for the platform and agent management: the AMS (Agent Management System) and the DF (Directory Facilitator), which should act as white and yellow pages respectively, and the MTS (Message Transport System), which manages the interoperability among agent platforms. There exist different FIPA compliant agent platform implementations. In this sense, FIPA-OS (FIPA-Open Source), JADE (Java Agent Development Environment) and ZEUS are the most popular. In this work, the JADE agent platform has been chosen.

Semantic Web Technologies

The World Wide Web (WWW), also called "the Web", was invented in 1989 by Tim Berners-Lee and his colleagues at CERN and it changed the way people gather and access information. Nowadays, the Web is an ever-growing huge data repository. As a consequence, a major bottleneck has emerged when trying to exploit the information represented in the Web, namely, how to find a specific piece of information we may be interested in. The Semantic Web was conceived with the purpose of solving this issue. It aims at adding semantics to the data published on the Web (i.e., establish the meaning

[62] http://www.fipa.org/

[63] http://agent.omg.org/

of the data), so that machines are able to process these data in a similar way a human can do (Berners-Lee et al., 2001). For this, ontologies are the backbone technology. Particularly, ontologies are expected to be used to provide structured vocabularies that describe the relationships between different terms, allowing computers (and humans) to interpret their meaning flexibly yet unambiguously (Horrocks et al., 2003).

A number of different ontology definitions can be found currently in literature. Some of them are in line with the philosopher Quine's interpretation (Quine, 1961), who said that everything that can be quantified exists. One of the best known definitions is Tom Gruber's (Gruber, 1993): "An ontology is an explicit specification of a conceptualization". This definition was later refined by Borst (Borst, 1997), stating that "an ontology is a formal specification of a shared conceptualisation". In this context, formal refers to the need of machine-understandable ontologies. This definition emphasises the need of agreement in carrying out a conceptualisation. On the other hand, shared refers to the type of knowledge contained in the ontologies, that is, consensual, non-private knowledge. Eventually, other researchers fused these previous two definitions and enunciated that "an ontology is a formal and explicit specification of a shared conceptualisation" (Studer et al., 1998). In this work, this ontology definition has been adopted.

Ontologies provide for a common vocabulary of an area and define –with different levels of formality- the meaning of the terms and the relations between them. Knowledge in ontologies is mainly formalized using five kinds of components: classes, relations, functions, axioms and instances (Gruber, 1993). Classes in the ontology are usually organized into taxonomies. Sometimes, the definition of ontologies has been diluted, in the sense that taxonomies are considered to be full ontologies (Studer et al., 1998).

Although ontologies have been used for many years in different research fields, they have become the de-facto standard knowledge representation technology after the emergence of the Semantic Web along with Semantic Web Services and the Semantic Grid. For all these new research branches, ontologies are the cornerstone technology. OWL (Web Ontology Language) (McGuinness & van Harmelen, 2004) is the *de facto*

Semantic Web standard ontology language. In this work, OWL has been applied as the knowledge representation language.

The semantic description of Web resources allows Web applications to be more powerful so that for example new smarter Web searches can be enacted. However, with this isolated approach the Web remains as a mere repository of information without the possibility to provide support for processing this information. This is the purpose of the Web Services technology. The idea behind Web Services is to extend the Web from a distributed source of information to a distributed source of functionality. Thus, Web Services connect computers and devices with each other using the Internet to exchange and combine data in new ways. A Web Service can be defined in a simple way as a service attached/linked to some location at the Internet that can be accessed through a standard protocol.

Web Services technology is based on a set of standard protocols such as UDDI (Universal Description, Discovery and Integration), SOAP (Simple Object Access Protocol), and WSDL (Web Services Description Language). UDDI provides a mechanism for clients to find Web Services by defining a standard way to publish and discover information about them (Clement et al., 2004). SOAP is a standard that defines XML-coded messages between two applications over the Internet protocols (Gudgin et al., 2006). Finally, WSDL provides a description of connection and communication with a particular Web Service by describing its functionality using an XML language (Booth & Liu, 2007). With all these components, everybody can use the service offered by any Web Service available on the Internet, and it is not necessary to take into account the programming language in which the service originally was defined. However, as the Web grows in size and diversity, there is an increased need to automate aspects of Web Services, including discovery, execution, selection, composition and interoperation. In fact, one of the key advantages of Web Services is that they make it possible for dynamic service composition using independent, reusable software components. The problem is that current technology around UDDI, WSDL, and SOAP provide limited support for all that (Fensel & Bussler, 2002).

The joint application of Semantic Web and Web Services in order to create intelligent Web Services is referred as Semantic Web Services (Lara et al., 2003). Semantic Web

Services are a step forward in the way to a Next Generation Web. They consist of describing Web Services with semantic content so that service discovery, composition and invocation can be done automatically by, for example, the use of intelligent agents able to process the semantic information provided. The W3C is currently examining various approaches with the purpose of reaching a standard for the Semantic Web Services technology: OWL-S, WSMO, SWSF, and WSDL-S, SAWSDL.

The first approach to be submitted to the W3C was OWL-S (OWL Web Ontology Language for Services), namely, an ontology for services that makes it possible for agents to discover, compose, invoke, and monitor services with a high degree of automation. It is composed of three main parts: the service profile for advertising and discovering services; the process model, which gives a detailed description of a service's operation; and the grounding, which provides details on how to interoperate with a service via messages (OWL-S, 2004).

WSMO submission comprises three different elements: Web Service Modeling Ontology (WSMO), Web Service Modeling Language (WSML), and Web Service Execution Environment (WSMX). WSMO provides a conceptual framework for semantically describing all relevant aspects of Web Services in order to facilitate the automation of discovering, combining and invoking electronic services over the Web (WSMO, 2005). It is based on the Web Service Modeling Framework (WSMF) (Fensel & Bussler, 2002), which consists of four main elements: ontologies, which provide for the terminology used by other elements; goals that define the problems that should be solved by Web Services; Web Services descriptions that define various (functional and behavioural) aspects for a Web Service; and mediators, which bypass interoperability problems. WSML provides a formal syntax and semantics for WSMO and is based on different logical formalisms (Description Logics, First-Order Logic and Logic Programming). Finally, WSMX defines a Semantic Web Services architecture and provides an initial implementation based on the WSMO conceptual model. It enables for discovery, selection, mediation, invocation and interoperation of Semantic Web Services. The goal is twofold: to provide a test-bed for WSMO and to demonstrate the viability of using WSMO as a model for achieving dynamic interoperation of Semantic Web Services.

The Semantic Web Services Framework (SWSF) is a proposal that comprises the Semantic Web Services Language (SWSL) and the Semantic Web Services Ontology (SWSO). Fundamentally, SWSL presents two variants, SWSL-FOL, which is based on First-Order Logic (FOL), and SWSL-Rules, which takes the "logic programming" paradigm as its starting point. SWSO is a conceptual model for Semantic Web Services. Two formal characterizations of that model have been realized: FLOWS (First-Order Logic Ontology for Web Services), which is derived from SWSL-FOL, and ROWS (Rules Ontology for Web Services), based on SWSL-Rules (SWSF, 2005).

One of the most recent submissions to the W3C is WSDL-S (Web Service Semantics). It defines a mechanism to associate semantic annotations with Web Services that are described using WSDL. The semantic information added by WSDL-S over WSDL includes definitions of precondition, input, output and effects on Web Service operations. Besides, the operation element can be annotated by a concept in a semantic model that provides a high level description of the operation. On the other hand, WSDL-S also presents a mechanism to add categorization information to services which may be used to narrow the range of candidate services in service discovery (WSDL-S, 2005).

Finally, SAWSDL follows the same approach as WSDL-S. It defines a set of extension attributes for WSDL and XML Schema, which allows for the semantic description of WSDL components. In SAWSDL, the semantic annotation is done, like in WSDL-S, by stating in WSDL references to external semantic models, such as ontologies. Therefore, SAWSDL does not set a particular language to represent the semantic models, but provides the means to associate WSDL documents to concepts from an external semantic model by using annotations.

Along with some of these W3C submissions, different tools have been implemented that bring together the major Semantic Web Services functionalities into an integrated framework. One example is WSMX (Web Services Execution Environment), an execution environment to perform dynamic discovery, selection, mediation, invocation and inter-operation of Semantic Web Services (WSMX, 2005). Another example is IRS (the Internet Reasoning Service), a Semantic Web Services framework which allows applications to semantically describe and execute Web Services (Motta et al. 2003;

Cabral et al., 2006). The IRS system supports the provision of semantic reasoning services within the context of the Semantic Web.

Both Intelligent Agent and Semantic Web Service technologies are able to reach remarkable achievements. However, they suffer from different shortcomings that straiten their functionality when applied separately. So, for example, in the case of agent technology its major flaw is the use of non-standard communication protocols. The use of proprietary protocols like IIOP or RMI prevents agents from being able to successfully communicate when different companies are involved. In those situations, companies' firewalls might be an impediment for the messages that go across companies' boundaries to reach their target. This eventually led to the emergence of several overlapping technologies such as Web Services that make use of standard technologies. Solutions to this problem imply the creation of gateways between every pair of companies aimed at interacting. As a consequence, no new dynamic links can be established between two entities that have not been set up for. Problems have also been reported on the effective exploitation of semantics in agents' communication mechanisms. In particular, none of the FIPA-compliant platforms (e.g. JADE, FIPA-OS, and ZEUS) provides for the proper support to handle the semantic dimension of the FIPA-ACL language (Louis & Martinez, 2005).

On the other hand, from its very conception, Semantic Web Service technology has been linked to agents. The semantic mark-up of services capabilities was intended for autonomous software entities (i.e., Intelligent Agents) to be able to understand and process them. Therefore, the necessity of using IA for this purpose seems quite straightforward. However, different non agent-based infrastructures for SWS operation have been developed (e.g. WSMX, IRS, and METEOR). The problem of these systems is that they are facing the very same problems and challenges (e.g. matchmaking, negotiation, trust, security, etc.) that have been subject to study in Artificial Intelligence for years. Then, they need to implement ad-hoc mechanisms that would be easily accessible through the utilization of agent technology.

RELATED WORK

It is widely recognised that the cooperative interaction between Intelligent Agents and Semantic Web Services can lead to the development of new more powerful applications. However, there is much discussion on how to actually achieve this integration. Blacoe & Portabella (2005) point out that in order to integrate what they called 'agent-based services' and 'web-based services' three main scenarios are possible: (1) Web Services provide the more basic level functionality and agents provide higher-level functions by using, combining and choreographing Web Services, so achieving added-value functions; (2) Communication in Web Services and agents becomes equivalent, so the distinction between them disappears ('*agents in web service wrappers*'); and (3) Both types remain separate creating a heterogeneous service space and interoperating through gateways and translation processes.

The integration of agents and (Semantic) Web Services has been addressed for different purposes successfully. The *Semantic Web FRED* project (SWF) combines agent technology, ontologies, and Semantic Web Services in order to develop a system for automated cooperation (Stollberg et al., 2004a; Stollberg et al, 2004b). In this system, software agents ('*Freds'*) perform tasks on behalf of their owners and interact among them if they have to. In order to resolve a task, agents make use of Services, computational resources that allow for automated resolution of tasks. The authors distinguish among three types of Services: plans (Java programs), processes (complex and nested services), and external Web Services (through WSDL). A major problem of the SWF is that it seems to be tightly bound to WSMO. In fact, the SWF is supposed to be a "WSMO Implementation". The framework we present here includes mechanisms to support all the current SWS approaches. It is even possible to dynamically incorporate support for new solutions.

Another example is the GODO (GOal Driven Orchestration) system (Gómez et al., 2004; Gómez et al., 2006), which is an agent located between users and the WSMX execution environment. When users want to send goals to WSMX, they have to write them down in WSML, a formal language that can be hard to understand to a common user (actually there exist some adapters that receive goals in other syntax, e.g. SOAP

messages, and transform them into WSML syntax). GODO is able to transform user requests in natural language to WSMX goals in WSML. With this purpose, it incorporates a language analyzer that determines the concepts, attributes, attribute values, and relationships within a sentence thus producing a lightweight ontology. Then, GODO uses the ontology to generate the goals to be executed and sends them to WSMX. GODO's main drawback is that it interacts with a Semantic Web Service infrastructure instead of the Semantic Web Services themselves. Due to this, few of the advantages of agents systems are exploited (e.g. agent-based negotiation cannot take place between parties and agents techniques for tasks such as discovery, composition and invocation have to be implemented ad-hoc within the infrastructure).

A further approach for Web Services and Intelligent Agents interoperation is presented in (Buhler and Vidal, 2003; Vidal et al., 2004; Buhler and Vidal, 2005). Here, the authors highlight the passive behaviour of Web Services and propose to wrap them in proactive agents. The idea is to use an Intelligent Agent acting on behalf of a Web Service within a workflow. When this Web Service is intended to be executed, its representative agent gets the flow control and attempts to improve the workflow process by using its 'intelligent' capabilities. Therefore, static workflows defined through Web Services become highly dynamic control flows thanks to the proactive and autonomous characteristics of agents. The authors also envision workflow-based MASs where service providers are agents themselves, thus acquiring the full proactive, autonomous, and selfish characteristics that are associated with agency. The problem of this approach is that semantically described Web Services are not considered at all. The authors claim that the state of the technology is still embryonic and so they just focus on BPEL workflows of Web Services. Therefore, neither the advantages of using Semantic Web Services are exploited nor the challenges faced.

The approaches developed so far present shortcomings mainly due to their inability to completely benefit from the advantages of this combination. In fact, they restrain the possibilities by considering only some of the properties of either Intelligent Agents or Semantic Web Service technology. From these considerations, this work initial hypothesis is that Intelligent Agents and (Semantic) Web Services were conceived for quite different purposes, and so they must lie on different abstraction layers. The main

idea behind Agent Technology was not for Intelligent Agents to be able to provide services, but to act as autonomous entities that incorporate intelligence and cognitive capacities, which allow them to show a goal-oriented, pro-active behaviour and to establish, either competitive or cooperative, interaction processes with other software entities in order to satisfy their design objectives. On the other hand, Web Services involved a further evolution in Distributed Computing and their only purpose is to provide worldwide-accessible functionality. These conceptual differences between Agent Technology and (Semantic) Web Services Technologies lead to the need to have both technologies working in an integrated environment and glimpse the advantages of this combination in the development of complex systems. Based on this working hypothesis, an integrated framework that combines agents and Semantic Web Services has been designed in order to develop software applications capable of dealing with the Web dynamism.

KNOWLEDGE TECHNOLOGIES-BASED SEMANTIC WEB SERVICE FRAMEWORK

As it was aforementioned, while (Semantic) Web Service infrastructures provide a high degree of interoperability across platforms and operating systems, they do not posses either enough degree of autonomy or ability to automatically adapt to changing situations. In these settings, agents can contribute to make systems more autonomous and dynamic, thus maximizing their perceived utility. In line with this, the framework proposed in this work comprises both approaches Intelligent Agents and Semantic Web Services working cooperatively in the same environment by means of ontologies as the facilitating technology.

Ontologies are the paramount technology of our approach as they operate as the 'glue' that binds the remainder components of the architecture. Firstly, ontologies function as universal vocabularies so that Web Services and agents share the same interpretation of the terms contained in the messages that they exchange. Secondly, ontologies are useful to semantically describe Web Services' capabilities and processes. This semantic description can then be automatically processed by software entities, so that Web Service discovery, composition, selection, execution and monitoring can be done without human intervention. Then, from the agents' perspective, each agent's local domain-related knowledge may be extracted from, or built upon, the application domain ontology. Moreover, inter-agent communication may be carried out by means of a common vocabulary based on an agreed ontology. Finally, in a similar way to what is presented in (Tamma et al., 2005), the negotiation processes between agents may take place in accordance with protocols represented in an ontology and by using the related strategies also stored in an ontology. It is at run-time, and depending on the problem they face, that agents with the desire to cooperate agree upon the protocol and strategy they are going to use.

The framework presented here is based on a multi-tier architecture that is composed of four different layers. The lower layer, namely, the *Business Logic Layer* provides the most specific operations. It comprises the internal and private business processes that constitute the companies' *engine*. Thus, Web Services are deployed that show off

particular elements of these business processes and make some functionality available publicly. Those public services along with the semantic description of their capabilities lay on the second layer, namely the *Semantic Web Services Layer*. Adding semantic annotations to Web Services capabilities enables software entities to automatically interact with those services in a dynamic way and without human intervention. In particular, new services can emerge and others may change their functionality or even disappear at run-time, but the system would keep on working and the changes would be reflected on the application instantly. These sophisticated software entities, namely Intelligent Agents, that interact with and take advantage of the offered services are located in the *Intelligent Agents Layer*. Intelligent Agents make use of the semantic annotation of services capabilities to automatically discover, compose, invoke and monitor Web Services. They are also able to exhibit and dynamically propagate the changing functionality provided in the lower layers. Finally, the *Application Layer* is responsible for organising (orchestrate and coordinate) agents to actually perform useful activities for users. In this way, depending on the agents available in the system and the way they inter-operate different user-tailored applications can be obtained.

The framework comprises the two middle layers of the four-tier infrastructure namely, the Intelligent Agents and the Semantic Web Services layers. As a result, the framework becomes independent from both the domain and the actual application in which it is to be applied. Therefore, the framework can be considered as a reference architecture for several business scenarios and complex environments such as eGovernment, eScience, eBusiness or Supply Chain Management.

The framework is composed of three main components: a set of intelligent agents that constitute a MAS, several ontology repositories, and three different interfaces for interacting with the external actors that have been identified.

In the platform proposed to run the system, there are seven types of agents. Agents are grouped in two main categories: agents that act on behalf of service owners and agents that act on behalf of service consumers. Those acting on behalf of service owners manage the access to services and ensure that the contracts are fulfilled. On the other side, the agents that act on behalf of service consumers have to locate services, agree on

contracts, and receive and present results. Next, a description of the seven types of agents is given:

- Broker Agent: it is responsible for solving the interoperability issues. Three different levels are considered: data mediation, process mediation and functional interoperability.
- Customer Agent: it acts as a user representative. First, (individual or collective) users indicate their preferences. Then, they specify the goal to be achieved. The goal is carried out and the results given back to the user. The intermediate process happens transparently to the user.
- Discovery Agent: it is in charge of searching in the Semantic Web Services repository for the service or set of services (i.e. composition) that satisfy the requisites established by the users.
- Framework Agent: it is responsible for monitoring and ensuring a correct functioning of the platform. This type of agent also controls and balances the workload.
- Provider Agent: it acts as a service provider representative. The entities set their preferences regarding service execution and these are taken into account during the negotiation process with the service consumers.
- Selection Agent: it is in charge of selecting the most appropriate (single or compound) service from the set of services found by the discoverer according to the users' preferences. A negotiation process with the different service representatives (i.e. Service Agent) is carried out for that purpose.
- Service Agent: it acts as a service representative. The service provider establishes a concrete set of preferences regarding a particular service and these are taken into account when negotiating with service consumers (in this case, the Selection Agent).

The framework proposed in this work has been designed so that there is not a fixed way tasks, which each agent carries out, should be implemented. In fact, it is at run-time (not compile-time) when it is determined which implementation is actually used. For this purpose, the '*role*' concept is introduced. Roles are encapsulations of dynamic

behaviour and properties that can be played by agents. The use of roles presents a number of benefits that can be summarized as follows: (1) roles are dynamic and flexible; (2) roles are responsibility-driven; (3) roles are context-sensitive (Zhao et al., 2004). Roles are also classified in two main groups: between roles dealing with service-related issues from those related to the framework management. Next, a description of the thirteen types of roles is put forward:

- Service-related roles
    - Broker role: it represents the functionality needed for solving all kind of interoperability problems (data, process and functional mediation).
    - Composer role: it allows the achievement of a goal by means of several composed services.
    - Invoker role: It invokes a Web Service once the operation to be executed and the parameters are known.
    - Matchmaker role: it finds the services whose semantic descriptions match the goal that was sent by the user.
    - Monitor role: it ensures that the contracts established for the execution of the services by both service owners and service consumers are fulfilled.
    - Ontology manager role: it includes functionality associated with the access and processing of ontologies.
    - Selector role: it provides the functionality necessary for the selection of a service from a list of services according to a set of preferences.
- Framework management roles
    - Negotiator role: it enacts the actual negotiation process between the parties establishing the basis for the system execution.
    - Platform manager role: it controls and balances the system workload.
    - Provider representative role: it interacts with service providers. At a high level of abstraction, it must be able to enforce the conditions present in the company's business strategy.
    - Service representative role: it acts on behalf of services, participating in negotiations and improving the services offered when possible.

- Consumer representative role: it interacts with service consumers by, firstly, determining their wishes and, then, returning the expected results.
- Global Monitor role: it monitors the events in the application, detects possible problems, and defines the actions to take in case of error.

At run-time each agent decides on what roles it *wants* to play depending on the goals the agent pursues. Nevertheless, some of the roles are mandatory for some agents, as they characterize the type of agent the agents belong to.

Regarding ontologies, four different kinds have been identified: application and domain ontology, agent local knowledge ontology, negotiation ontology, and Semantic Web Services ontologies. The application ontology contains the knowledge entities (i.e. concepts, attributes, relationships, and axioms) to model the application in which the framework is to be employed. The domain ontology, on the other hand, represents a conceptualization of the specific domain the framework is going to be applied in. This ontology supports the communication among the component in the framework without misinterpretations. The agent local knowledge ontology contains, for each agent, the knowledge about the environment it possesses. This ontology generally includes knowledge about the assigned tasks, and the mechanisms and resources available to achieve those tasks. The negotiation ontology comprises both negotiation protocols and negotiation strategies that constitute the negotiation mechanisms agents need to coordinate. Finally, the ontologies that contain the semantic description of Web Services are kept in the Semantic Web Services repository.

At last, three different interfaces have been included within the framework architecture. They aim at enabling the interaction with the actors that are external with respect to the framework: service consumers, service providers, and software developers. Software developers can, by means of their interface, customize the application by setting up the specific ontologies to be used. They also have to instantiate and configure the core agents necessary for the proper functioning of the system (customer, provider and service agents will be launched as needed at run-time). Once the application has been properly set up, both service consumers and service providers can register in the system and use it as a meeting point. Through their interface, service

providers can modify the list of services they provide and set the conditions under which a service they provide must be executed. Service consumers, on the other hand, can, by means of their interface, query the system and trigger the execution of one or several Web Services in order to fulfil a particular goal.

USE CASE SCENARIOS

The knowledge-based Semantic Web Service framework has been theoretically tested in three different use case scenarios. These three scenarios raise several challenges the framework must face. Each scenario is based on a simulated environment in which a customer needs to carry out a specific task and various services are available. The framework acts as a middleware tool by facilitating the customer to make use of the appropriate services. All these simulated use case scenarios are related to eCommerce transactions.

In the first scenario, the way the framework behaves when dealing with the basic Semantic Web Services exploitation and management tasks (i.e. discovery, selection and invocation) is analyzed. In this example, by making use of the agents that constitute the framework, it is evaluated how the system copes with all the functional requirements that the proposed scenario poses. In particular, the Discovery Agent, by making queries to the Semantic Web Services repositories, is able to find the services that match with the goal. The Selection Agent, on the other hand, is responsible for sorting the list of services. To do that, the agent assesses the utility each service based on both, users' preferences and the services' proposals (which have been obtained during the negotiation process among the Selection Agent and Service Agents). Service Agents are in charge of dealing with the services they represent, invoke the appropriate methods, and retrieve the results. Finally, the Customer Agent controls the interaction with the user, query processing and result presentation.

In the second scenario, the framework faces the difficulties of service composition. Several agents must deal with the new requirements this task arises. First of all, the Discovery Agent, if no services have been found that match the initial goal, is responsible for decomposing the goal into various sub-goals. Then, the Discovery Agent searches for services that match each sub-goal. The Selection Agent, on the other side, takes into account the goal decomposition and sorts out each subgoal's list of services. Finally, the Customer Agent is in charge of sending the invocation request to all the appropriate Service Agents.

In the last scenario, the capability of the system to provide access to heterogeneous data sources has been analysed. In this scenario, each provider uses a different ontology to describe the service. In order to handle the heterogeneity, different agents are affected. When querying the service descriptions repository, the Discovery Agent must make use of the ontology mappings to be able to find the appropriate services. Similarly, the Service Agents, which represent the services that should be executed, employ the mapping rules to determine the input parameters each service expects. Finally, the Customer Agent receives all the responses and processes them by using the mapping rules. Once processed, the Customer Agent presents the integrated outcome to the final user.

A SEMANTIC WEB SERVICES EXECUTION ENVIRONMENT

Based on the above described framework, a Semantic Web Services execution environment has been implemented. This tool comprises a Multi-Agent System that complies with the requirements imposed by the knowledge-based Semantic Web Service framework, and a Web application that constitutes the interface with the three external actors identified: software developers, service providers and service consumers.

The seven agents envisioned in the framework were implemented on top of the Jade Multi-Agent Platform[64]. Jade uses the concept of 'behaviour' to specify the functionality provided by an agent. In order to fully accomplish what the proposed framework states, a set of roles has been attached to the behaviours. Behaviours, therefore, only control the inter-agent communication. Roles, on the other hand, provide the actual implementation of all the functionality supported by the system. Then, when an agent's behaviour receives a message, it delegates to the role that is in charge of managing it, the control flow. Next, the list of implemented behaviours is given and the association with roles established:

- Broker behaviour
    - Broker role
    - Ontology manager role
- Consumer representative behaviour
    - Consumer representative role
    - Ontology manager role
- Global monitor behaviour
    - Global monitor role
- Invoker behaviour
    - Invoker role
    - Ontology manager role
- Matchmaker behaviour
    - Matchmaker role

[64] http://jade.tilab.com/

    - Composer role
    - Ontology manager role
- Monitor Behaviour
    - Monitor role
- Negotiation Responder Behaviour
    - Negotiator role
    - Ontology manager role
- Platform manager behaviour
    - Platform manager role
- Provider representative behaviour
    - Provider representative role
- Selector behaviour
    - Selector role
    - Negotiator role
    - Ontology manager role
- Service representative behaviour
    - Service representative role

Despite that roles are attached to behaviours in design-time, the number and type of roles an agent plays may vary in run-time, thus modifying the behaviours the agent shows. However, as it was aforementioned, there are roles that are tightly coupled with some kinds of agents and set the basic behaviours an agent must show.

Aiming at adding flexibility and modularity to the system, each role is defined through the interface it complies with. Then, software developers are free to include their own implementations for these roles as a plug-in to the system. Nevertheless, a basic implementation for each role has been developed. The core tasks roles are responsible for include user query processing, service discovery and composition, service selection, and service invocation.

For user query processing, KAText (Ruiz-Sánchez et al., 2003; Valencia-García et al., 2004), a natural language processing tool, has been applied. It translates a sentence in natural language into a lightweight ontology (i.e. an ontology that contains the concepts

that are present in the sentence and the relationships among them). The resultant ontology conveys the user goal represented according the internal goal model. This goal model, expressed in OWL, contains both the expected outcome and the data the user inputs.

Service discovery and composition tasks are closely related. Actually, composition takes place when discovery has failed in finding services for a proposed goal. The implemented software application follows the following flow of control: (1) services that match the goal are retrieved; (2) if no service has been found, then the goal is decomposed into several sub-goal; (3) each sub-goal is then separately matched against the services; (4) if at least a service has been found for each sub-goal, the process ends; otherwise, those sub-goals that have no matching services are sent back to step '2'. Matchmaking is performed through SPARQL queries over the service descriptions repository. On the other hand, a STRIPS-like planner[65] has been applied for composition.

The selection technique applies a decision support system approach based on utility. In particular, during the selection stage, the system calculates a utility value for each service that has been discovered in the previous phase. For this, a negotiation process takes place between the Selection Agent and the Service agents that represent the services under question. During the negotiation process, the service representatives state the conditions under which their services are to be executed. Based on those conditions, ant taking into account the user's preferences, the Selection Agent assesses each service and sorts the list of services accordingly.

The invocation task is carried out by the Service Agent. This agent accesses the service's semantic description, and retrieves the expected input parameters. Once the parameters have been set, the agent executes the service and gets the output. Then, it sends back the results to the user through the Customer Agent.

The interfaces with the external actors have been implemented by using JSP and Servlet technologies. This Web application provides software developers the means for (1) add/delete ontology repositories, (2) add/delete ontologies, (3) incorporate new roles implementations, and (4) instantiate agents. Service providers can, through the Web

[65] http://www.dcs.shef.ac.uk/~pdg/com1080/java/strips/

application, (1) register in the system and modify their personal data, and (2) add/delete/update services they provide. Finally, service consumers have the means for (1) register in the system and modify their personal data, (2) issue a query to the system, (3) select a set of services and ask for their execution, and (4) examine the outcome of the services execution. Service consumers can also interact with the application through a Web Service, which exposes the system functionality so that it can be accessed by other software entities.

PROTOTYPE VALIDATION

The implemented prototype has been empirically validated by means of three disparate experiments. Through these experiments, the capability of the system to face some of the problems that arise in the explored domains is checked. Each experiment is based on an example scenario in which the prototype has been employed to solve a concrete domain-dependent task. Three different domains have been considered: eGovernment, Bioinformatics, and eCommerce.

In the first experiment, the prototype has been successfully applied to the eGovernment domain. The proposed scenario consists of a citizen willing to change his/her residence and the two concerned public administrations (source and target), which provide services by means of Web Services. In these settings, the system shows its capability to, first of all, discover and compose several services that, jointly, can reach the desired goal. Then, the effectiveness of the invocation mechanism provided by the system is also proved.

The second experiment poses some additional challenges. Bioinformatics is the domain of interest in this experiment. A major issue in this domain, as in any e-Science field, is the vast amount of data available. Furthermore, the information is commonly stored in heterogeneous data sources. Therefore, the prototype needs to handle that heterogeneity and, at the same time, properly manage the available services. In this case, the scenario consists of several Web Services that provide access to different biomedical Web resources. The domain is bounded to ‘oncogenes’, a modified gene, or a set of nucleotides that codes for a protein and is believed to cause cancer. This experiment demonstrates the ability of the system to provide integrated access to heterogeneous data sources.

In the last experiment, the prototype has been applied in the eCommerce domain in order to provide support for B2C and B2B transactions. In these settings, the most common difficulty is related to the amount of providers in the same environment that provide the same or similar products/services. In such situations, the hardest challenge the system must face is the selection of the provider whose proposal is the best according to the user’s preferences. Therefore, negotiation is the key in these kinds of

environments. The proposed scenario in this experiment comprises three computer hardware providers, which have set some basic conditions to the services they provide. A user willing to buy a hard disk triggers the system machinery. The system, when trying to decide which service to execute, carries out a negotiation process. During the negotiation, each service sets the conditions under which they are to be executed based on the preferences previously established by the providers. The system, then, assesses each service proposal taking into account the user's preferences. Thus, each service is given a utility value, and the list of available services is sorted accordingly. The user is presented with the sorted list, selects the preferred one and requests its execution. As a result, in this experiment, besides negotiation and selection mechanisms, the system is also evaluated for discovery, invocation and monitoring tasks.

CONCLUSIONS AND FUTURE WORK

Agent technology promises to enable cost effective distributed systems that are powerful and flexible. However, several problems arose that have prevented Multi-Agent Systems from being applicable to real-world settings. A major flaw of the technology is the use of non-standard proprietary protocols what makes it difficult for agents that have not been designed to work together to interoperate. Web Service technology is the evolution of other solutions in the distributed programming field such as RMI, CORBA or DCOM. Web Services are based on three open Internet standards, namely, UDDI, SOAP, and HTTP. The utilization of standard protocols enables Web Services to constitute loosely-coupled distributed systems. In fact, a key advantage of this technology is that it makes possible dynamic service composition using independent, reusable software components. Semantic Web Services emerged to facilitate the automation of service discovery, composition, monitoring, and invocation. They consist of describing Web Services' capabilities with semantic content so that autonomous software entities can interact with the services without human intervention.

Although it is still subject to great discussion in both the Artificial Intelligence and the Distributed Computing areas, it seems quite straightforward that joining together Agent and Semantic Web Service technologies can lead to the development of new more powerful applications. Actually, a large number of research projects are being carried out nowadays by researchers worldwide with the purpose of integrating these two technologies. However, the existing approaches suffer from several shortcomings caused mainly by their inability to completely benefit from the advantages of this combination. In order to fully exploit the applicability of a combined infrastructure, it is important to notice that agents and Web Services were conceived with different purposes, and so, their strengths and weaknesses are different.

In line with this, the initial hypothesis assumed in the design and development of the framework was that Intelligent Agents and (Semantic) Web Services were conceived for quite different purposes, and so, they must lie on different abstraction layers. The main idea behind Agent Technology was not for Intelligent Agents to be able to provide services, but to act as autonomous entities that incorporate intelligence and cognitive

capacities, which allow them to show a goal-oriented pro-active behaviour and to establish, either competitive or cooperative, interaction processes with other software entities in order to satisfy their design objectives. On the other hand, Web Services involved a further evolution in Distributed Computing and their only purpose is to provide worldwide-accessible functionality. These conceptual differences between Agent Technology and (Semantic) Web Services Technologies lead to the need to have both technologies working in an integrated environment and glimpse the advantages of this combination in the development of complex systems.

Consequently, the objective of this thesis was to develop a knowledge-based Semantic Web Services framework that successfully integrates Intelligent Agents and Semantic Web Services technologies. For achieving this combination, the framework takes an ontology-centred approach. Ontologies are the facilitating technology that enables a seamlessly communication between agents and services. Ontologies are present at almost every stage of the interaction process. Ontologies are used not only to represent different kinds of knowledge (domain and application knowledge), but also to formally exhibit the services' capabilities. The former is employed by agents to communicate to each other without misunderstandings. The later enables agents to automatically discover, compose and invoke the available services.

The framework presented here is based on a multi-tier infrastructure, where Intelligent Agents and Semantic Web Services are located on two different layers of the cake. Web Services, in the lower layer, make some parts of the companies' private business processes public. Then agents, in the upper layer, provide higher-level functions by using, combining and choreographing those services. At last, software developers can use the framework to build user-tailored applications. For this, they only have to properly orchestrate the agents in the upper layer so that they perform some specific activity.

The proposed framework consists of: (1) seven different agent types (Broker Agent, Customer Agent, Discovery Agent, Selection Agent, Provider Agent, Framework Agent and Service Agent); (2) four ontology repositories (domain and application ontologies repository, agents' local knowledge ontologies repository, negotiation protocol and strategies ontology repository, and Semantic Web Services ontologies repository) that

can be either local or remote; and (3) three interfaces (service consumers interface, service providers interface, and customization interface) aimed at enabling the interaction with the three external actors (service consumers, service providers and software developers respectively). A number of roles have also been conceived that comprise all the system functionality. The roles an agent instance chooses to play will eventually determine the behaviour of the agent at run-time. An agent can dynamically change the roles it aims to play depending on the goal it has to achieve at any moment. It is not the purpose of this work to procure new innovative solutions for Semantic Web Services related tasks (i.e. matchmaking, composition, mediation, and so on) but to provide an infrastructure capable of maximizing the outcome of the combination of Intelligent Agent and Semantic Web Services technologies.

The knowledge-based Semantic Web Service framework has been theoretically tested in three different use case scenarios in the eCommerce domain. These three scenarios raise several challenges the framework must face. Each scenario is based on a simulated environment in which a customer needs to carry out a specific task and various services are available. The framework acts as a middleware tool by facilitating the customer to make use of the appropriate services. By means of these use cases, the framework has proved to be useful for efficiently managing semantically annotated Web Services.

Based on the designed framework, a Semantic Web Services execution environment has been implemented. This tool comprises a Multi-Agent System that complies with the requirements imposed by the knowledge-based Semantic Web Service framework, and a Web application that constitutes the interface with the three external actors identified: software developers, service providers and service consumers. The Jade platform has been used for the development of the multi-agent system, and the Jena library was chosen for the management of ontologies within the system. Java technologies around JSP and Servlets have been employed for implementing the Web application.

This prototype has been empirically validated by means of three experiments. Each experiment is based on an example scenario in which the prototype is employed in a particular domain to solve a concrete task. Three different domains have been considered: eGovernment, Bioinformatics, and eCommerce. Each domain imposes

different challenges the application has been able to face successfully. The critical issue in the eGovernment domain is the ability to compose several services in order to achieve a high-level goal. On the other hand, the application is able to deal with a vast amount of heterogeneous data sources in Bioinformatics. Finally, eCommerce environments involve many providers supplying the same products/services, and so, the system should be able to properly choose the best provider.

Regarding future work, different issues may be pointed out. First, the inclusion of GRID services within the framework will be analysed. As it happened in the Web Services field, GRID computing is evolving in line with the Semantic Web with the purpose of reaching a Semantic GRID. This new trend involves the annotation of GRID components with semantic content so that software entities can automatically manage GRID services. As opposite to Web Services, GRID services are, in general, stateful. Therefore, in order for the framework to provide support for GRID services, several changes should be made to the components that constitute the framework.

Another possible improvement to the framework is concerned with service composition. Currently, the composition technique that is applied only permits sequential list of services, whereas it would be desirable to consider other control structures such as parallelism, choice, loops, etc. A proper way to do this would be to use one of the existing workflow languages for Web Services (e.g. WS-BPEL, YAWL, etc.).

In addition, some improvements can be made in the software prototype to take advantage of all the benefits the framework procures. So, for example, by taking into account the user profile and the log of the user interactions with the system, it would be possible to develop a recommender system that would improve the user experience. Besides, by fully exploiting the reasoning capabilities of Jena and the associated reasoner engines (e.g. Pellet, Racer, etc.), it would be possible to provide a better discovery solution. Finally, the flexibility of the prototype can be further extended by, for example, supporting coalition formation among the agents in the platform.

REFERENCIAS

Abela, C. (2003). Semantic Web Services Composition. In Proc. of the Computer Science Annual Workshop (CSAW'03), pp. 1-9.

Acton, D. (2004). A Brief History of Distributed Computing. Diapositivas de clase. Consultado el 2 de Abril de 2007, http://www.ugrad.cs.ubc.ca/~cs416/X/Notes/01%20-%20Sep%2010%20-%20History/02.history.html

Alesso, H.P. & Smith, C.F. (2005). Developing Semantic Web Services. A K Peters, Ltd.

Álvarez Espinar, M. (2005). Introducción a la Web Semántica. V Workshop REBIUN. Consultado el 1 de Enero de 2007, http://www.w3c.es/Presentaciones/2005/1018-WebSemanticaREBIUN-MA

Alvarez Sabucedo, L., García-Sánchez, F., Anido Rifón, L. & Martínez-Béjar, R. (2006). Plataforma basada en anotación y procesamiento semántico para eGovernment. II Jornadas Cientifico-Técnicas en Servicios Web (JSWEB'2006), Santiago de Compostela, España.

Amor, M., Fuentes, L. & Pinto, M. (2003). Interoperabilidad entre Plataformas de Agentes FIPA: Una Aproximación Basada en Componentes. 3er Congreso Iberoamericano de Telemática (CITA'2003). Montevideo, Uruguay.

Ana Mas (2005). Agentes Software y Sistemas Multi-Agente: Conceptos, Arquitecturas y Aplicaciones. Pearson Educación, S.A. Madrid, 2005.

Antoniou, G. & van Harmelen, F. (2004). A Semantic Web Primer. The MIT Press.

Austin, J. L. (1962). How To Do Things With Words. Oxford University Press, Oxford.

AWSI WG (2005). IEEE Foundations for Intelligent Physical Agents Standards Committee (FIPA SC). Agents and Web Services Interoperability Working Group. Charter Proposal. Consultado el 24 de Abril de 2007, http://www.fipa.org/subgroups/AWSI-WG-docs/AWSI_WG_Charter.pdf

Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., & Patel-Schneider, P. F. (Eds.) (2003). The Description Logic Handbook. Cambridge University Press.

Beam, C. & Segev, A. (1997). Automated Negotiations: A Survey of the State of the Art. Wirtschaftsinformatik 39(3), pp. 263-268.

Bellifemine, F., Caire, G., Poggi, A. & Rimassa, G. (2003). JADE - A White Paper. "EXP in search of innovation" Journal, 3(3), pp. 6-19.

Bellifemine, F., Caire, G., Trucco, T. & Rimassa, G. (2006). JADE Programmer's Guide. Consultado el 23 de Enero de 2007, http://jade.tilab.com/doc/programmersguide.pdf

Berners-Lee, T. (2000). Semantic Web - XML2000. Consultado el 28 de Diciembre de 2006, http://www.w3.org/2000/Talks/1206-xml2k-tbl/

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. Scientific American, Mayo 2001, pp. 34-43.

Bernstein, A. & Kaufmann, E. (2006). GINO - A guided Input Natural Language Ontology Editor. 5th International Semantic Web Conference (ISWC 2006). Lecture Notes in Computer Science (4273), pp. 144-157.

Blacoe, I. & Portabella, D. (2005). Guidelines for the integration of agent-based services and web-based services. Deliverable D2.4.4 (WP2.4), Knowledge Web project.

Borst, W.N (1997). "Construction of Engineering Ontologies for Knowledge Sharing and Reuse.". CTIT Ph.D-thesis series No.97-14. University of Twente. Enschede, The Netherlands.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. & Orchard, D. (2004). Web Services Architecture. W3C Working Group Note. Consultado el 2 de Abril de 2007, http://www.w3.org/TR/ws-arch

Booth, D. & Liu, C. K. (Eds.) (2007). Web Services Description Language (WSDL) Version 2.0. Part 0: Primer. W3C Working Draft 26 March 2007. Consultado el 27 de Marzo de 2007, http://www.w3.org/TR/wsdl20-primer/

Brickely, D., & McBride, B. (Eds.) (2004). RDF Vocabulary Description Language 1.0: RDF Schema. Consultado el 28 de Diciembre de 2006, http://www.w3.org/TR/rdf-schema/

Buhler, P. A. & Vidal, J. M. (2003). Semantic Web Services as Agent Behaviors. In Agentcities: Challenges in Open Agent Environments, LNCS/LNAI, B. Burg, J. Dale, et al., Eds. Berlin: Springer-Verlag, 2003.

Buhler, P. A. & Vidal, J. M. (2005). Towards Adaptive Workflow Enactment Using Multiagent Systems. Information Technology and Management Journal, 6(1), 61-87.

Cabral, L., Domingue, J., Galizia, S., Gugliotta, A., Tanasescu, V., Pedrinaci, C. & Norton, B. (2006). IRS-III: A Broker for Semantic Web Services based Applications. In Proc. of the 5th International Semantic Web Conference (ISWC 2006).

Caire, G. (2003). JADE Tutorial. JADE Programming for Beginners. Consultado el 19 de Enero de 2007, http://jade.tilab.com/doc/JADEProgramming-Tutorial-for-beginners.pdf

Castells, P. (2003). La Web Semántica. C. Bravo, M.A. Redondo (Eds.), Sistemas Interactivos y Colaborativos en la Web. Ediciones de la Universidad de Castilla - La Mancha, ISBN: 84-8427-352-0, pp.195-212. Consultado el 13 de Marzo de 2007, http://www.ii.uam.es/~castells/publications/castells-uclm03.pdf

Chinnici, R., Moreau, J.J., Ryman, A. & Weerawarana, S. (Eds.) (2007). Web Services Description Language (WSDL) Version 2.0. Part 1: Core Language. W3C Working Draft 26 March 2007. Consultado el 27 de Marzo de 2007, http://www.w3.org/TR/wsdl20/

Cimpian, E. & Mocan, A. (2005). Process Mediation in WSMX. WSMX Working Draft 08 July 2005. Consultado el 5 de Junio de 2007, http://www.wsmo.org/TR/d13/d13.7/v0.1/

Clement, L., Hately, A., von Riegen, C., & Rogers, T. (Eds.) (2004). UDDI Version 3.0.2. UDDI Spec Technical Committee Draft. Consultado el 28 de Marzo de 2007, http://uddi.org/pubs/uddi_v3.htm

Collis, J. C. & Ndumu, D. T. (1999). The Role Modelling Guide. Informe, Applied Research and Technology, BT Labs.

Corcho, O., Alper, P., Kotsiopoulos, I., Missier, P., Bechhofer, S. & Goble, C. A. (2006). An overview of S-OGSA: A Reference Semantic Grid Architecture. Journal of Web Semantics, 4(3), pp. 102-115.

Cost, R. S., Finin, T., Joshi, A., Peng, Y., Nicholas, C., Chen, H., Kagal, L., Perich, F., Zou, Y., Tolia, S. & Soboroff, I. (2002). ITTALKS: A Case Study in the Semantic Web and DAML. In Proc. of the International Semantic Web Working Symposium, July 2002.

Crubezy, M., Motta, E., Lu, W. & Musen, M. (2003). Configuring Online Problem-Solving Resources with the Internet Reasoning Service. IEEE Intelligent Systems 18(2), pp. 34-42.

de Bruijn, J. (2006). Logics for the Semantic Web. DERI Technical Report 2006-06-03. Consultado el 2 de Enero de 2007, http://www.deri.at/fileadmin/documents/deri-tr-2006-06-03.pdf

DeLoach, S. (2001). Analysis and Design using MaSE and agentTool. In Proc. of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS).

Domingue, J., Gutierrez, L., Cabral, L., Rowlatt, M., Davies, R. & Galizia, S. (2004). WP 9: Case Study eGovernment. D9.3 e-Government Ontology. DIP Deliverable. Consultado el 28 de Junio de 2007, http://dip.semanticweb.org/documents/D9-3-improved-eGovernment.pdf

Durinck, S., Moreau, Y., Kasprzyk, A., Davis, S., De Moor, B., Brazma, A. & Huber, W. (2005). BioMart and Bioconductor: a powerful link between biological databases and microarray data analysis. Bioinformatics, 21(16), pp. 3439-3440.

Elamy, A. H. (2005). Perspectives in Agent-Based Technology. AgentLink News 18, pp. 19-22.

Elenius, D., Denker, G., Martin, D., Gilham, F., Khouri, J., Sadaati, S. & Senanayake, R. (2005). The OWL-S editor-a development tool for semantic web services. In Proc. of the 2nd European Semantic Web Conf (ESWC2005), 3532 (2005), pp. 78–92.

Etzold, T., Harris, H. & Beaulah, S. (2003). SRS: An Integration Platform for Databanks and Analysis Tools in Bioinformatics. Managing Scientific Data.

Fatima, S. S., Wooldridge, M. & Jennings, N. R. (2004). An agenda-based framework for multi-issue negotiation. Artificial Intelligence, 152(2004), pp. 1-45.

Fensel, D. & Bussler, C. (2002). The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications, 1(2), pp. 113-137.

Fernández-Breis, J. T., Martínez-Béjar, R., Valencia-García, R., Vivancos-Vicente, P. J., García-Sánchez, F. (2004). Towards Cooperative Frameworks for Modeling and Integrating Biological Processes Knowledge. IEEE Transactions on NanoBioscience, 3(3), pp. 164-171.

Fikes, R. E. & Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2, pp. 189-208.

Finin, T., Labrou, Y. & Mayfield J. (1995). KQML as an Agent Communication Language. In Proc. of the 3rd International Conference on Information and Knowledge Management (CIKM'94).

FIPA (2002a). FIPA Abstract Architecture Specification. FIPA Standard Specification 00001. Consultado el 11 de Enero de 2007, http://www.fipa.org/specs/fipa00001/SC00001L.pdf

FIPA (2002b). FIPA ACL Message Structure Specification. FIPA Standard Specification 00061. Consultado el 7 de Enero de 2007, http://www.fipa.org/specs/fipa00061/SC00061G.pdf

FIPA (2004). FIPA Agent Management Specification. FIPA Standard Specification 00023. Consultado el 11 de Enero de 2007, http://www.fipa.org/specs/fipa00023/SC00023K.pdf

FIPA-OS (2001). FIPA-OS Developers Guide. Consultado el 24 de Enero de 2007, http://fipa-os.sourceforge.net/docs/Developers_Guide.pdf

Flanagan, D. (1999). Java en pocas palabras. Ed. McGraw-Hill.

García-Sánchez, F., Valencia-García, R. & Martínez-Béjar, R. (2005). An integrated approach for developing e-commerce applications. Expert Systems with Applications, 28(2), pp. 223-235.

Genesereth, M.R. & Fikes, R.E. (1992). Knowledge Interchange Format, Version 3.0 Reference Manual. Technical report logic-92-1, Computer Science Department, Stanford University.

Gibbins, N., Harris, S. & Shadbolt, N. (2003). Agent-based Semantic Web Services. In Proc. of the 12th International World Wide Web Conference (WWW2003), pp. 710-717.

Gómez, J. M., Rico-Almodóvar, M., García-Sánchez, F., Liu, Y., Terra, M. (2007). BIRD: Biomedical Information Integration and Discovery with Semantic Web Services. In Proc. of the 2nd International Work-Conference on the Interplay between Natural and Artificial Computation. Lecture Notes in Computer Science (4528), pp. 561-570.

Gómez, J. M., Rico-Almodóvar, M., García-Sánchez, F., Martínez-Béjar, R. & Bussler, C. (2004). GODO: Goal-driven Orchestration for Semantic Web Services. WSMO Implementation Workshop, September 2004.

Gómez, J. M., Rico-Almodóvar, M., García-Sánchez, F., Toma, I. & Han, S. (2006). GODO: Goal Oriented Discovery for Semantic Web Services. Discovery on the WWW Workshop (SDISCO'06), Beijin, China.

Gómez-Sanz, J. J. & Fuentes, R. (2002). The INGENIAS Methodology. Fourth Iberoamerican Workshop on Multi-Agent Systems Iberagents 2002.

Gómez-Sanz, J. J. & Pavón, J. (2004). Methodologies for Developing Multi-Agent Systems. Journal of Universal Computer Science, 10(4), pp. 359-374.

Greenwood, D., Buhler, P. A. & Reitbauer, A. (2005). Web Service Discovery and Composition using the Web Service Integration Gateway. IEEE International Conference on e-Technology, e-Commerce, and e-Services (EEE 2005), pp. 789-790.

Grimm, S., Motik, B. & Preist, C. (2006). Matching Semantic Service Descriptions with Local Closed-World Reasoning. In Proc. of the 3rd European Semantic Web Conference, LNCS 4011, pp. 575-589.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. Knowledge Acquisition, 5, pp. 199-220.

Gruber, T. R. (2004). Interview to Tom Gruber. AIS SIGSEMIS Bulletin (October 2004), 1(3), p. 4.

Guarino, N. (1998). Formal Ontology and Information Systems. En N. Guarino (ed), In Proc. of the 1st International Conference on Formal Ontology in Information Systems (FOIS-98), Trento, Italia, 6-8 Junio, 1998. Amsterdam, IOS Press, pp. 3-15.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J-J., Nielsen, H. F., Karmarkar, A., & Lafon, Y. (2006). SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C Recommendation. Consultado el 9 de Mayo de 2007, http://www.w3.org/TR/soap12-part1/

Guijarro, L. (2004). Analysis of the Interoperability Frameworks in e-Government Initiatives. Lecture Notes in Computer Science 3183, pp. 36-39.

Haas, L. M., Schwarz, P. M., Kodali, P., Kotlar, E., Rice, J. E. & Swope, W. C. (2001). DiscoveryLink: A system for integrated access to life sciences data sources. IBM Systems Journal, 40(2), pp. 489-511.

He, M., Jennings, N. R. & Leung, H-F. (2003). On Agent-Mediated Electronic Commerce. IEEE Transactions on Knowledge and Data Engineering, 15(4), pp. 985-1003.

Hendler, J. (2001). Agents and the Semantic Web. IEEE Intelligent Systems, 16(2), pp. 30-37.

Horrocks, I., Patel-Schneider, P. & Van Harmelen, F. (2003). From SHIQ and RDF to OWL: The making of a Web Ontology Language. Journal of Web Semantics: Science, Services and Agents on the World Wide Web, 1 (2003), pp. 7-26.

Huhns, M. N. (2002). Agents as Web Services. Internet Computing, 6(4), pp. 93-95.

Huhns, M. N. & Stephens, L. M. (1999). Multiagent Systems and Societies of Agents. En Multiagent Syhstems: A Modern Approach to Distributed Artificial Intelligence, Gerhard Weiss (Eds.), MIT Press, Cambridge, USA, pp. 80-120.

Iglesias, C. (1998). Definición de una metodología para el desarrollo de Sistemas Multi-Agente. Tesis doctoral. Departamento de ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid.

Jennings, N.R. (2001). An Agent-Based Approach for building Complex Software Systems. Communications of the ACM, 44(4), pp. 35-41.

Jennings, N.R., Faratin, P., Lomuscio, A.R., Parsons, S., Sierra, C., & Wooldridge, M. (2001). Automated Negotiation: Prospects, Methods and Challenges. International Journal of Group Decision and Negotiation, 10(2), pp. 199-215.

Kalfoglou, Y. & Schorlemmer, M. (2003). Ontology mapping: the state of the art. The Knowledge Engineering Review, 18(1), pp. 1-31.

Kang, B. (1996). Multiple classification ripple down rules. PhD Thesis, University of New South Wales.

Kelemen, V. (2006). Simple Example for using the JadeGateway class. JADE Tutorial, 25 October 2006. Consultado el 14 de Junio de 2007, http://jade.tilab.com/doc/tutorials/JadeGateway.pdf

Keller, U., Lara, R. & Polleres, A. (Eds.) (2004). WSMO Web Service Discovery. WSML Working Draft 12 November 2004. Consultado el 30 de Mayo de 2007, http://www.wsmo.org/2004/d5/d5.1/v0.1/

Kifer, M., Lausen, G. & Wu, J. (1995). Logical foundations of object-oriented and frame-based languages. Journal of the ACM, 42(4), pp. 741-843.

Klyne, G. & Carroll, J.J. (Eds.) (2004). Resource Description Framework (RDF): Concepts and Abstract Sysntax. Consultado el 28 de diciembre de 2006, http://www.w3.org/TR/rdf-concepts/

Kopecky, J. (Ed.) (2005). Relationship of WSMO to other relevant technologies. W3C Member Submission 3 June 2005. Consultado el 1 de Mayo de 2007, http://www.w3.org/Submission/WSMO-related/

Kuter, U., Sirin, E., Parsia, B., Nau, D. S. & Hendler, J. (2005). Information gathering during planning for Web Service composition. Journal of Web Semantics, 3(2-3), pp. 183-205.

Laboratorio de Agentes Software Inteligentes. Universidad de Carnegie Mellon. (2001). Multi-Agent Systems. Consultado el 5 de Enero de 2007, http://www.cs.cmu.edu/~softagents/multi.html

Lara, R., Lausen, H., Arroyo, S., De Bruijn, J. & Fensel, D. (2003). Semantic Web Services: description requirements and current technologies. International Workshop on Electronic Commerce, Agents, and Semantic Web Services, In conjunction with the Fifth International Conference on Electronic Commerce (ICEC 2003).

Lloyd, J.W. (1987). Foundations of logic programming (2nd Edition). Springer-Verlag.

Lomuscio, A.R., Wooldridge, M. & Jennings, N.R. (2001). A classification scheme for negotiation in electronic commerce. Agent-Mediated Electronic Commerce: A European AgentLink Perspective (Eds. F. Dignum and C. Sierra), Springer Verlag Lecture Notes in Artificial Intelligence, 1991, pp. 19-33.

Louis, V. & Martinez, T. (2005). The JADE Semantic Agent: Towards Agent Communication Oriented Middleware. AgentLink News 18, pp. 16-18.

Luck, M. & McBurney, P. (2004). Retos de la Tecnología de Agentes en el horizonte del año 2010. Novática nº 170, pp. 6-10.

Luke, S., Spector, L., Rager, D. & Hendler, J. (1997). Ontology-based Web Agents. In Proc. of the 1st International Conference on Autonomous Agents, pp. 59-66.

Martín-Sánchez, F., Iakovidis, I., Nørager, S., Maojo, V., de Groen, P. C., Van der Lei, J., Jones, T., Abraham-Fuchs, K., Apweiler, R. & Babic, A. (2004). Synergy between medical informatics and bioinformatics: facilitating genomic medicine for future health care. Journal of Biomedical Informatics, 37(1), pp. 30-42.

Masuoka, R., Labrou, Y., Song, Z., Parsia, B. & Hendler, J. (2005). On Building Task Computing. AgentLink News 18, pp. 23-25.

Masouka, R., Parsia, B. & Labrou, Y. (2003). Task Computing - The Semantic Web meets Pervasive Computing -. In D. Fensel et al. (Eds.), "The Semantic Web - ISWC 2003", the 2nd International Semantic Web Conference (ISWC 2003), Sanibel Island, Fl, USA, October 2003, LNCS 2870, pp. 866-881.

McGuinness, D.L. & van Harmelen, F. (Eds.) (2004). OWL Web Ontology Language: Overview. Consultado el 1 de Enero de 2007, http://www.w3.org/TR/owl-features/

McIlraith, S., Son, T. C., & Zeng, H. (2001). Semantic Web Services. IEEE Intelligent Systems, 16(2), pp. 46-53.

McIlraith, S. & Son, T. C. (2002). Adapting Golog for Composition of Semantic Web Services. In Proc. of the 8th International Conference on Principles and Knowledge Representation and Reasoning (KR 2002), pp. 482-496.

Mitra, N. & Lafon, Y. (Eds.) (2007). SOAP Version 1.2 Part 0: Primer (Second Edition). W3C Recommendation. Consultado el 9 de Mayo de 2007, http://www.w3.org/TR/soap12-part0/

Motta, E., Domingue, J., Cabral, L. & Gaspari, M. (2003). IRS-II: A Framework and Infrastructure for Semantic Web Services. In Proc. of the 2nd International Semantic Web Conference (ISWC 2003), LNCS 2870.

Narayanan, V. & Jennings, N.R. (2005). An adaptive bilateral negotiation model for e-commerce settings. In Proc. of the 7th IEEE International Conference on E-Commerce Technology (CEC 2005), pp. 34-41.

Noy, N. F. & Musen, M. A. (2000). Prompt: Algorithm and tool for automated ontology merging and alignment. In Proc. of the 17th National Conference on Artificial Intelligence (AAAI2000), pp. 450-455.

Nwana, H.S., Ndumu, D.T., Lee, L.C. & Collis, J.C. (1999). ZEUS: A Toolkit for Building Distributed Multi-Agent Systems. Applied Artificial Intelligence, 13(1-2), pp. 129-185.

OASIS WSBPEL TC (2007). Web Services Business Process Execution Language Version 2.0. OASIS Standard, 11 April 2007. Consultado el 29 de Junio de 2007, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

O'Sullivan, J., Edmond, D. & ter Hofstede, A. H. (2005). Formal description of non-functional service properties. Technical report, Queensland University of Technology, Brisbane. Consultado el 31 de Mayo de 2007, http://www.bpm.fit.qut.edu.au/about/docs/non-functional.jsp

OWL (2004). OWL Web Ontology Language Overview. W3C Recommendation 10 February 2004. Consultado el 9 de Mayo de 2007, http://www.w3.org/TR/owl-features/

OWL-S (2004). OWL-S: Semantic Markup for Web Services. W3C Member Submission 22 November 2004. Consultado el 27 de Abril de 2007, http://www.w3.org/Submission/OWL-S/

Paolucci, M., Ankolear, A., Srinivasan, N. & Sycara, K. (2003a). The DAML-S Virtual Machine. In Proc. of the 2nd International Semantic Web Conference (ISWC 2003), pp. 290-205.

Paolucci, M., Kawamura, T., Payne, T. & Sycara, K. (2002). Semantic Matching of Web Services Capabilities. In Proc. of the 1st International Semantic Web Conference (ISWC 2002), LNCS 2342, pp. 333-347.

Paolucci, M., Srinivasan, N., Sycara, K. & Nishimura, T. (2003b). Toward a Semantic Choreography of Web Services: From WSDL to DAML-S. In Proc. of the 1st International Conference on Web Services (ICWS'03), pp. 22-26.

Paolucci, M. & Sycara, K. (2003). Autonomous Semantic Web Services. IEEE Internet Computing, 7(5), pp. 34-41.

Pavón, J. & Gómez-Sanz, J.J. (2003) Agent Oriented Software Engineering with INGENIAS. In Proc. of the 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2003), pp.394-403.

Payne, T. R., Singh, R. & Sycara, K. (2002a). Calendar Agents on the Semantic Web. IEEE Intelligent Systems 17(3), pp. 84-86.

Payne, T. R., Singh, R. & Sycara, K. (2002b) Browsing Schedules - An Agent-based approach to navigating the Semantic Web. In Proc. of International Semantic Web Conference (ISWC), pp. 469-473.

Polleres, A. & Lara, R. (Eds.) (2005). A Conceptual Comparison between WSMO and OWL-S. WSMO Working Group Working Draft. Consultado el 1 de Mayo de 2007, http://www.wsmo.org/2004/d4/d4.1/v0.1/

Quine, W.V. (1961). From a Logical Point of View, Nine Logico-Philosophical Essays. Cambridge, Massachusetts: Harvard University Press.

Rahwan, I., Ramchurn, S. D., Jennings, N. R., McBurney, P., Parsons, S. & Sonenberg, L. (2004). Argumentation-based negotiation. The knowledge Engineering Review, 18(4), pp. 343-375.

Ruiz-Sánchez, J. M., Valencia-García, R., Fernández-Breis, J. T., Martínez-Béjar, R. & Compton, P. (2003). An approach for incremental knowledge acquisition from text. Expert Systems With Applications, 25, pp. 77-86.

Russell, S. & Norvig, P. (2004). Inteligencia Artificial. Un Enfoque Moderno. Segunda edición. Ed. Pearson Prentice Hall.

Salt Cairols, J. F., González de la Hoz, S. & Fernández Casani, A. (2007). Introducción a las Tecnologías GRID. Principales logros del Proyecto EGEE. XVII Escuela de Verano de Informática - Computación GRID: del middleware a las aplicaciones, Albacete, España.

SAWSDL (2007). Semantic Annotations for WSDL and XML Schema. W3C Working Draft 10 April 2007. J. Farrell & H. Lausen (Eds.). Consultado el 4 de Mayo de 2007, http://www.w3.org/TR/sawsdl/

Scicluna, J., Abela, C. & Montebello, M. (2004). Visual Modelling of OWL-S Services. IADIS International Conference WWW/Internet.

Searle, J. R. (1969). Speech Acts: an Essay in the Philosophy of Language. Cambridge University Press, Cambridge.

Shafiq, O., Ali, A., Suguri, H. & Ahmad, F. (2005). AgentWeb Gateway for dynamic and seamless integration of FIPA Multi Agent System and W3C Web Service System. Analysis of existing AWSI related implementations. FIPA AWSI WG. Consultado el 24 de Abril de 2007, http://www.fipa.org/subgroups/AWSI-WG-docs/Omair_AWSI_Req_Doc.doc

Shafiq, O., Suguri, H., Ali, A. & Fensel, D. (2006). A first step towards enabling Interoperability between Software Agents and Semantic Web Services: Multi Agent Systems adapting Web Services Standards. IBIS - Interoperability in Business Information Systems, 2(2), pp. 97-117.

Sheshagiri, M., desJardins, M. & Finin, T. (2003). A Planner for Composing Services Described in DAML-S. International Conference on Automated Planning & Scheduling, ICAPS 2003.

Sirin, E., Parsia, B. & Hendler, J. (2004). Filtering and Selecting Semantic Web Services with Iteractive Composition Techniques. IEEE Intelligent Systems, 19(4), pp. 42-49.

Song, Z., Labrou, Y. & Masuoka, R. (2004). Dynamic Service Discovery and Management in Task Computing. MobiQuitous 2004, August 22-26, 2004, Boston, USA, pp. 310-318.

Steve, G., Gangemi, A. & Pisanelli, D. M. (1997). Integrating Medical terminologies with ONIONS Methodology. ITBM-CNR Technical Report ITBM-RIM-3/1997. Consultado el 27 de Diciembre de 2006, http://www.loa-cnr.it/Papers/onions97.pdf

Stollberg, M. & Haller, A. (2005). Semantic Web Services Tutorial. 3rd International Conference on Web Services (ICWS 2005), Orlando, Florida.

Stollberg, M., Herzog, R. & Zugmann, P. (2004a). SWF Framework. SWF Project Deliverable. Consultado el 16 de Abril de 2007, http://swf.deri.at/papers/SWF-D1-SWFFramework-final.pdf

Stollberg, M., Herzog, R. & Zugmann, P. (2004b). Semantic Web Fred. Poster at the 3rd International Semantic Web Conference (ISCW2004), Hiroshima, Japan.

Stollberg, M., Roman, D., Toma, I., Keller, U, Herzog, R., Zugmann, P. & Fensel, D. (2005). Semantic Web Fred - Automated Goal Resolution on the Semantic Web. In Proc. of the 38th Hawaii International Conference on System Sciences (HICSS-38 2005), Big Island, HI, USA.

Stollberg, M., Cimpian, E., Mocan, A. & Fensel, D. (2006). A Semantic Web Mediation Architecture. Semantic Web and Beyond Computing for Human Experience. Canadian Semantic Web, 2, Springer, pp. 3-22.

Studer, R., Benjamins, R. & Fensel, D. (1998) Knowledge Engineering: Principles and Methods. Data and Knowledge Engineering, 25(1-2), pp. 161-197.

SWSF (2005). Semantic Web Services Framework (SWSF) Overview. W3C Member Submission 9 September 2005. Consultado el 27 de Abril de 2007, http://www.w3.org/Submission/SWSF/

SWSF-SWSL (2005). Semantic Web Services Language (SWSL). W3C Member Submission 9 September 2005. Consultado el 27 de Abril de 2007, http://www.w3.org/Submission/SWSF-SWSL/

SWSF-SWSO (2005). Semantic Web Services Ontology (SWSO). W3C Member Submission 9 September 2005. Consultado el 27 de Abril de 2007, http://www.w3.org/Submission/SWSF-SWSO/

Sycara, K. (1998). Multi-Agent Systems. AI Magazine, 19(2), pp. 79-92.

Tamma, V., Blacoe, I., Smith, B. L. & Wooldridge, M. (2004). SERSE: Searching for Semantic Web Content. In Proc. of the 16th European Conference on Artificial Intelligence (ECAI 2004), pp. 63-67.

Tamma, V., Phelps, S., Dickinson, I. & Wooldridge, M. (2005). Ontologies for supporting negotiation in e-commerce. Engineering Applications of Artificial Intelligence, 18(2005), pp. 223-236.

The Gene Ontology Consortium (2000). Gene Ontology: tool for the unification of biology. Nature Genetics 25, pp. 25-29.

The USHER Project (2002a). e-Business Adviser Handbook. Section - 2.2 - Business to Consumer (B2C). IST Project USHER, USHER Handbook, Version 2. Consultado el 25 de Junio de 2007, http://www.usherproject.org.uk/support/hb/HBs22v2.pdf

The USHER Project (2002b). e-Business Adviser Handbook. Section - 2.3 - Business to Business (B2B). IST Project USHER, USHER Handbook, Version 2. Consultado el 25 de Junio de 2007, http://www.usherproject.org.uk/support/hb/HBs23v2.pdf

Toma, I. & Foxvog, D. (2006). Non-Functional Properties in Web Services. WSMO Working Draft 25 October 2006. Consultado el 31 de Mayo de 2007, http://www.wsmo.org/TR/d28/d28.4/v0.1/

Tran, Q. N. (2005). MOBMAS - A Methodology for Ontology-Based Multi-Agent Systems Development. Tesis doctoral. University of New South Wales.

Tveit, A. (2000). A survey of Agent-Oriented Software Engineering. Informe, Norwegian University of Science and Technology, Mayo, 2002. Consultado el 9 de Enero de 2007, http://citeseer.ist.psu.edu/tveit01survey.html

Valencia-García, R. (2005). Un Entorno para la Extracción Incremental de Conocimiento desde Texto en Lenguaje Natural. Tesis Doctoral, Universidad de Murcia. Consultado el 3 de Julio de 2007, http://www.tdr.cesca.es/TESIS_UM/AVAILABLE/TDR-1123105-140512//rvalencia.pdf

Valencia-García, R., Ruiz-Sánchez, J. M., Vivancos-Vicente, P. J., Fernández-Breis, J. T. & Martínez-Béjar, R. (2004). An incremental approach for discovering medical knowledge from texts. Expert Systems With Applications, 26(3), pp. 291-299.

van der Aalst, W. M. P. & ter Hofstede, A. H. M. (2005). YAWL: Yet Another Workflow Language. Information Systems, 30(4), pp. 245-275.

van Heijst, G. Schreiber, A.T. & Wielinga, B.J. (1997). Using explicit ontologies in KBS development. International Journal of Human-Computer Studies, 46(2), pp. 183-292.

Verma, K., Gomadam, K., Sheth, A. P., Miller, J. A. & Wu, Z. (2005). The METEOR-S Approach for Configuring and Executing Dynamic Web Processes. Technical Report. June, 2005.

Vidal, J. M., Buhler, P. A. & Stahl, C. (2004). Multiagent Systems with Workflows. IEEE Internet Computing, 8(1), pp. 76-82.

Vlassis, N. (2003). A Concise Introduction to Multiagent Systems and Distributed AI. Consultado el 5 de Enero de 2007, http://carol.science.uva.nl/~vlassis/cimasdai/

W3C Oficina Española (2006). Estándares: Fundamentos de la Web. I Jornadas Tecnologías Web y Software Libre. A Coruña, 12 de Julio de 2006. Consultado el 13 de Marzo de 2007, http://www.w3c.es/Presentaciones/2006/0712-estandaresGPUL-MA/

Walton, C. D. (2005). Uniting Agents and Web Services. AgentLink News 18, pp. 26-28.

Wiederhold, G. (2003). Interoperation, Mediation, and Ontologies. In Proc. of the International Symphosium on Fifth Generation Computer Systems (FGCS94), Workshop on heterogeneous Cooperative Knowledge-Bases, Vol. W3, pp. 33-48.

Wilkinson, M., Schoof, H., Ernst, R. & Haase, D. (2005). BioMOBY Successfully Integrates Distributed Heterogeneous Bioinformatics Web Services. The PlaNet Exemplar Case. Plant Physology 138, pp. 5-17.

Williams, S. K., Battle S. A. & Esplugas-Cuadrado, J. (2005). Protocol Mediation for Adaptation in Semantic Web Services. HP Technical Reports (HPL-2005-78). Consultado el 5 de Junio de 2007, http://www.hpl.hp.com/techreports/2005/HPL-2005-78.html

Wooldridge, M. (2002). An Introduction to MultiAgent Systems. Ed. John Wiley & Sons Ltd.

Wooldridge, M. & Jennings, N. R. (1995). Intelligent Agents: theory and practice. The Knowledge Engineering Review, 10(2), pp. 115-152.

Wooldridge, M., Jennings, N. R. & Kinny, D. (2000). The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems, 3(3), pp. 285-312.

WSDL-S (2005). Web Service Semantics - WSDL-S. W3C Member Submission 7 November 2005. Consultado el 27 de Abril de 2007, http://www.w3.org/Submission/WSDL-S/

WSML (2005). Web Service Modeling Language (WSML). W3C Member Submission 3 June 2005. Consultado el 9 de Mayo de 2007, http://www.w3.org/Submission/WSML/

WSMO (2005). Web Service Modeling Ontology (WSMO). W3C Member Submission 3 June 2005. Consultado el 27 de Abril de 2007, http://www.w3.org/Submission/WSMO/

WSMX (2005). Web Service Execution Environment (WSMX). W3C Member Submission 3 June 2005. Consultado el 9 de Mayo de 2007, http://www.w3.org/Submission/WSMX/

Wu, D., Parsia, B., Sirin, E., Hendler, J. & Nau, D. (2003). Automating DAML-S web services composition using SHOP2. In Proc. of 2nd International Semantic Web Conference (ISWC2003), 2003.

Zambonelli, F., Jennings, N.R. & Wooldridge, M. (2003). Developing Multiagent Systems: The Gaia Methodology. ACM Transactions on Software Engineering Methodology, 12(3), pp. 317-370.

Zhao, L., Mehandjiev, N. & Macaulay, L. (2004). Agent Roles and Patterns for Supporting Dynamic Behavior of Web Services Applications. 3rd International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'04), New York, USA.