UNIVERSIDAD POLITÉCNICA DE MADRID

FACULTAD DE INFORMÁTICA

Una Arquitectura Software Basada en Agentes y Recomendaciones Metodológicas para el Desarrollo de Entornos Virtuales de Entrenamiento con Tutoría Inteligente

Tesis Doctoral

Gonzalo Méndez Pozo

Licenciado en Informática

Madrid, Septiembre de 2008

Tesis Doctoral

Una Arquitectura Software Basada en Agentes y Recomendaciones Metodológicas para el Desarrollo de Entornos Virtuales de Entrenamiento con Tutoría Inteligente

Departamento de Lenguajes y Sistemas Informáticos
e Ingeniería de Software

presentada en la

FACULTAD DE INFORMÁTICA

de la

UNIVERSIDAD POLITÉCNICA DE MADRID

para la obtención del

GRADO DE DOCTOR EN INFORMÁTICA

Autor: **Gonzalo Méndez Pozo**
Licenciado en Informática por la
Universidad Politécnica de Madrid
Directora: **Angélica de Antonio Jiménez**
Doctora en Informática por la
Universidad Politécnica de Madrid

Madrid, Septiembre de 2008

Titulo:
Una Arquitectura Software Basada en Agentes
y Recomendaciones Metodológicas para el Desarrollo de
Entornos Virtuales de Entrenamiento con Tutoría Inteligente

Autor:
Gonzalo Méndez Pozo

Tribunal:

Presidente	:	Dña. Natalia Juristo Juzgado
Vocales	:	D. Pablo Gervás Gómez-Navarro
		D. Arcadio Reyes Lecuona
		D. Marcos Fernández Marín
Secretario	:	D. Ricardo Imbert Paredes
Suplentes	:	Dña. María Isabel Sánchez Segura
		D. Federico Peinado Gil

Acuerdan otorgar la calificación de

Madrid, 16 de Diciembre de 2008

Resumen

Una de las últimas tendencias en la aplicación de la informática a la educación se encuentra en la utilización de Entornos Virtuales tridimensionales en los que los estudiantes pueden actuar como si se encontrasen en un entorno real.

La combinación de estos entornos con aplicaciones que proporcionan tutoría inteligente para la práctica de actividades procedimentales recibe el nombre de Entorno Virtual de Entrenamiento con Tutoría Inteligente.

En muchos casos, la tutoría inteligente está proporcionada por un sistema software cuya arquitectura se asemeja a la de los Sistemas Inteligentes de Tutoría, aplicaciones de enseñanza cuyo origen se sitúa en los años 80 y cuyos principales objetivos eran dos: por un lado, se pretendía que la enseñanza se pudiese adaptar a las características de cada estudiante; por otro, se intentaba que el conocimiento del dominio de enseñanza fuese independiente de la estrategia pedagógica, lo que permitiría cambiar el domino de enseñanza sin modificar el resto del sistema.

Con el paso de los años y la incorporación de nuevas tecnologías, se ha hecho patente que el objetivo de reutilización perseguido por los Sistemas Inteligentes de Tutoría aún no ha sido alcanzado, y muchos de los nuevos sistemas que se construyen parten desde cero, puesto que no es posible reutilizar los elementos elaborados para sistemas previos.

En el presente trabajo se aborda el problema de elaborar una arquitectura software para Entornos Virtuales de Entrenamiento con Tutoría Inteligente que permita intercambiar los distintos elementos que la componen, así como reutilizarlos en distintos dominios de entrenamiento procedimental.

Para ello, se propone la utilización de agentes software como elemento de abstracción junto con técnicas específicas desarrolladas en el área de las arquitecturas software para obtener una arquitectura que permita el desarrollo de nuevos Entornos Virtuales de Entrenamiento con Tutoría Inteligente cuyos elementos sean fácilmente intercambiables y reutilizables.

Abstract

One of the new tendencies in the application of computer science to education is the utilization of three-dimensional virtual environments in which the students can act as if they were in the real one.

The combination of these virtual environments with software applications that provide intelligent tutoring to train procedural tasks are called Virtual Environments for Training with Intelligent Tutoring.

In many occasions, intelligent tutoring is provided by a software system whose architecture is similar to that of Intelligent Tutoring Systems. These systems are educational software whose origin dates back to the 80s, and whose objectives are twofold: on the one hand, teaching was intended to be adapted to the characteristics of every student; on the other hand, domain knowledge was designed to be independent from pedagogical strategies, which would ease the substitution of domain knowledge without affecting the rest of the elements that conform the system.

After some years of experience, and with the use of new technologies, the aim of reusing Intelligent Tutoring System's components has not been achieved, and many of the new systems that are developed nowadays are built from scratch, since it is not possible to reuse the elements designed to work in previous systems.

In the present work, we aim at the design of a software architecture for Virtual Environments for Training with Intelligent Tutoring that allows us to exchange its different constituent elements, as well as to reuse them in different procedural training domains.

With this objective in mind, we propose the use of software agents as an abstraction mechanism, together with specific techniques developed in the software architecture research area, in order to obtain a software architecture that allows de construction of new Virtual Environments for Training with Intelligent Tutoring whose elements are easily exchanged and reused.

Índice

I Planteamiento del Problema **1**

1. Introducción **3**
1.1. Un Poco de Historia 3
1.2. Problema a Resolver 4
1.3. Estructura del Documento 7

2. Estado de la Cuestión **9**
2.1. Introducción 9
2.2. Sistemas Inteligentes de Tutoría 10
2.3. Realidad Virtual y Entornos Virtuales 13
2.4. Agentes 17
2.4.1. ¿Qué es un agente? 17
2.4.2. El Estándar FIPA 20
2.4.3. Metodologías de Desarrollo con Agentes 21
2.5. Arquitecturas Software 23
2.6. Aplicación de la Realidad Virtual a la Enseñanza 28
2.7. Agentes Pedagógicos 31
2.8. Trabajos Realacionados 33
2.8.1. Agentes y Enseñanza 33
2.8.2. Entornos Virtuales de Enseñanza 56
2.8.3. Entornos Virtuales de Entrenamiento 68
2.9. Análisis del Trabajo Relacionado 92
2.10. Trabajo Previo 94
2.10.1. PRVIR 94
2.10.2. Integración de Steve y HeSPI 97
2.11. Reflexiones 99

II Propuesta de Solución 101

3. Planteamiento de la Solución 103
3.1. Introducción . . . 103
3.2. Sistemas Objeto de la Investigación . . . 103
3.2.1. Funciones de Tutoría . . . 107
3.2.2. Funciones de Experto . . . 107
3.2.3. Funciones de Modelado del Estudiante . . . 108
3.2.4. Funciones Relativas al Entorno Virtual . . . 108
3.2.5. Funciones de Simulación . . . 109
3.3. Sobre el Uso de Agentes . . . 109
3.4. Arquitectura Software . . . 112
3.4.1. ¿Por qué una Arquitectura? . . . 113
3.4.2. ¿Cómo se Evalúa la Arquitectura? . . . 114
3.4.3. ¿Cómo se Utiliza la Arquitectura? . . . 116
3.5. Objetivos del Trabajo . . . 117

4. Propuesta de Arquitectura Organizacional 121
4.1. Introducción . . . 121
4.2. Arquitectura de Alto Nivel . . . 122
4.3. Arquitectura Conceptual con Agentes . . . 125
4.3.1. Agentes de Primer Nivel . . . 126
4.3.2. Agentes de Segundo Nivel . . . 130
4.4. Análisis de la Arquitectura . . . 137
4.4.1. Efectos de la Planificación en la Arquitectura . . . 138
4.4.2. La Estructura Jerárquica . . . 139
4.5. Reflexiones . . . 140

5. Propuesta de Arquitectura Centrada en la Modificabilidad 143
5.1. Introducción . . . 143
5.2. Proceso de Diseño de la Arquitectura . . . 144
5.3. Atributos de Calidad . . . 146
5.3.1. Escenarios de Modificabilidad . . . 147
5.3.2. Escenarios de Eficiencia . . . 156
5.4. Documentación de la Arquitectura . . . 159
5.4.1. Descripción de la Documentación Necesaria . . . 159
5.4.2. Selección de la Documentación a Generar . . . 164
5.4.3. Plantilla de Documentación . . . 166
5.5. Diseño de la Arquitectura . . . 168
5.5.1. Vista de Descomposición . . . 168
5.5.2. Vista de Uso . . . 233

5.5.3. Vista Editor-Suscriptor 247
5.5.4. Vista *Peer-to-Peer* 274
5.5.5. Vista de Despliegue 288
5.6. Análisis de la Arquitectura 292
5.6.1. Escenarios de Modificabilidad 292
5.6.2. Escenarios de Eficiencia 300
5.7. Reflexiones 302

6. Recomendaciones Metodológicas y Criterios de Diseño 307
6.1. Introducción 307
6.2. Recomendaciones Metodológicas 308
6.2.1. Selección de Componentes de la Arquitectura 309
6.2.2. Realización de Modificaciones en el Sistema 313
6.2.3. Realización de Modificaciones en el SIT 314
6.3. Criterios para Realizar el Diseño de la Arquitectura 316
6.3.1. Diseño Top-Down 316
6.3.2. Estructura Jerárquica 318
6.3.3. Organización Peer-To-Peer 320
6.3.4. Edición-Suscripción 321
6.3.5. Relaciones de Uso 324
6.3.6. Sistemas Abiertos y Cerrados 325
6.4. Reflexiones 327

III Resultados y Aportaciones 329

7. Evaluación de la Arquitectura 331
7.1. Introducción 331
7.2. Selección del Método de Evaluación 332
7.3. Evaluación de la Arquitectura 333
7.3.1. Árbol de Pesos 334
7.3.2. Escenarios para la Evaluación 334
7.3.3. Análisis de la Arquitectura 338
7.4. Resultados de la Evaluación 354
7.5. Reflexiones 355

8. Resultados y Conclusiones 357
8.1. Introducción 357
8.2. Método de Trabajo 360
8.3. Aportaciones 363
8.3.1. Resultados Relativos a la Arquitectura 364

8.3.2. Resultados Relativos a las Recomendaciones 367
8.3.3. Resultados Relativos a la Implementación 368
8.4. Conclusiones 370

9. Líneas de Trabajo Futuro **371**
9.1. Introducción 371
9.2. Distintas Aproximaciones al Diseño de la Arquitectura 372
9.2.1. Arquitecturas Orientadas a Servicios 372
9.2.2. Distribución en un Grid 374
9.2.3. Diseño Alternativo de la Tutoría 375
9.2.4. Arquitecturas Software y Orientación a Agentes 376
9.3. Extensiones de la Arquitectura 377
9.3.1. Compañeros de Equipo Manejados por Agentes 377
9.3.2. Integración de una Arquitectura Cognitiva 378
9.3.3. Integración de Mecanismos de Percepción 379
9.3.4. Interacción mediante Lenguaje Natural 380
9.4. Reflexiones 382

IV Apéndices **383**

A. Sobre la Implementación de la Arquitectura **385**
A.1. Introducción 385
A.2. Implementación del Entorno Virtual 386
A.2.1. El Sistema Gestor de Entidades 386
A.2.2. El Simulador 388
A.2.3. Implementación de la Entidad 390
A.2.4. Implementación del Gestor de Entidades 392
A.2.5. Mensajes entre Subsistemas 395
A.2.6. Implementación del Simulador 396
A.2.7. Bibliotecas de Terceras Partes Utilizadas 398
A.3. Implementación de la Comunicación 400
A.3.1. El Protocolo de Comunicación 400
A.3.2. Diseño del Centro de Mensajes 403
A.3.3. Agentes de Comunicación 403
A.3.4. Diseño del Observador de Red en el EV 407
A.3.5. Envío de Mensajes 409
A.4. Diseño del Agente de Tutoría 414
A.4.1. Plan de Estudios 414
A.4.2. Plan de la Actividad 416
A.4.3. Seguimiento del Estudiante 418

V Bibliografía **421**

Bibliografía **423**

Índice de Figuras

2. Estado de la Cuestión **9**

Sistemas Inteligentes de Tutoría 10

2.1. Arquitectura clásica de un SIT 12

Agentes y Enseñanza 33

2.2. Adele explicando la importancia de un examen abdominal . . 34

2.3. Arquitectura monousuario de Adele 34

2.4. AutoTutor con Marco como tutor 36

2.5. Arquitectura de AutoTutor 37

2.6. PPP Persona y su arquitectura 39

2.7. Arquitectura de TEMAI 40

2.8. Arquitectura de Vincent 41

2.9. Arquitectura de Doris y su entorno 42

2.10. Arquitectura de Filip 43

2.11. Arquitectura de I-Help 47

2.12. Arquitectura de INES 49

2.13. Arquitectura del sistema de agentes de INES 50

2.14. Arquitectura de GIA 53

2.15. Arquitectura de Enxuto y de Jonas 54

2.16. Arquitectura externa e interna de ABITS 55

Entornos Virtuales de Enseñanza 56

2.17. Dos vistas del sistema GCW 57

2.18. Un avatar en el jardín de NICE 58

2.19. Conexiones a NICE a través de la red 59

2.20. Vista del Astronauta y Vista del Controlador 61

2.21. Interfaz de CVE-VM 62

2.22. Arquitectura de Red de CVE-VM 63

2.23. Herman the Bug en Design a Plant 64
2.24. Cosmo en Internet Advisor 65
2.25. Arquitectura de Cosmo 66
2.26. Arquitectura de WhizLow 67

Entornos Virtuales de Entrenamiento 68
2.27. STEVE dando instrucciones a un alumno 69
2.28. Arquitectura del Entorno Virtual de STEVE 70
2.29. Arquitectura de los Agentes de STEVE 71
2.30. Escenario principal del proyecto MRE 75
2.31. Arquitectura del proyecto MRE 77
2.32. Representación del guión del escenario 78
2.33. Entrenando una misión en TLCTS 80
2.34. Arquitectura de TLCTS 81
2.35. Arquitectura del cliente para PC de TLCTS 82
2.36. Jacob y las torres de Hanoi 84
2.37. Arquitectura de Jacob 84
2.38. Operación para apagar un incendio en Sécurévi 85
2.39. Arquitectura organizacional de MASCARET 87
2.40. Arquitectura de ADRI 88
2.41. Interfaz del estudiante de Lahystotrain 89
2.42. Arquitectura de Lahystotrain 90
2.43. Arquitectura de los agentes Tutor y Asistente 91
2.44. Arquitectura de los agentes de comportamiento 91

Trabajo Previo 94
2.45. Arquitectura de PRVIR 95
2.46. Interfaz de PRVIR 96
2.47. Interfaz de HeSPI 98
2.48. Integración de Steve y HeSPI 99

4. Propuesta de Arquitectura Organizacional **121**
4.1. Arquitectura clásica de un SIT 123
4.2. Arquitectura ampliada de un SIT 126
4.3. Arquitectura ampliada de un SIT con agentes 127
4.4. Arquitectura basada en agentes 130

5. Propuesta de Arquitectura Centrada en la Modificabilidad **143**
Escenarios de Calidad 146
5.1. Escenario de calidad genérico 147

Escenarios de Modificabilidad 147
5.2. Escenario general de modificabilidad en diseño 148
5.3. Escenario general de modificabilidad en ejecución 148
5.4. Escenario de sustitución del Entorno Virtual 149
5.5. Escenario de sustitución de la estrategia de tutoría 150
5.6. Escenario de sustitución del modelo del estudiante 151
5.7. Escenario de sustitución del planificador 152
5.8. Escenario de sustitución del planificador de rutas 153
5.9. Escenario de sustitución del simulador 154
5.10. Escenario de deshabilitación de la tutoría 155
5.11. Escenario de deshabilitación del cálculo de trayectorias 156

Escenarios de Eficiencia 156
5.12. Escenario general de eficiencia 157
5.13. Escenario de eficiencia ante una acción 158
5.14. Escenario de eficiencia en navegación 158

Vistas de Descomposición del Sistema 168
5.15. Descomposición del sistema 168
5.16. Interfaces de los módulos del sistema 170
5.17. Diagrama de contexto del sistema 171
5.18. Descomposición del EV 173
5.19. Interfaces de los módulos del EV 175
5.20. Diagrama de contexto del EV 176
5.21. Descomposición del Centro de Mensajes 178
5.22. Interfaces del Centro de Mensajes 179
5.23. Diagrama de contexto del Centro de Mensajes 180
5.24. Descomposición del Simulador 181
5.25. Interfaces del Simulador 182
5.26. Diagrama de contexto del Simulador 183
5.27. Descomposición del SIT 186
5.28. Interfaces del SIT 190
5.29. Diagrama de contexto del SIT 192

Vistas de Descomposición de los Agentes 194
5.30. Descomposición del Agente de Comunicación Global 195
5.31. Interfaces del Agente de Comunicación Global 196
5.32. Diagrama de contexto del Agente de Comunicación Global . . 197
5.33. Descomposición del Agente de Comunicación 199
5.34. Interfaces del Agente de Comunicación 200
5.35. Diagrama de contexto del Agente de Comunicación 201

5.36. Descomposición del Agente de Simulación 203
5.37. Interfaces del Agente de Simulación 204
5.38. Diagrama de contexto del Agente de Simulación 205
5.39. Descomposición del Agente Experto 207
5.40. Interfaces del Agente Experto 209
5.41. Diagrama de contexto del Agente Experto 210
5.42. Descomposición del Agente Mundo 211
5.43. Interfaces del Agente Mundo 213
5.44. Diagrama de contexto del Agente Mundo 214
5.45. Descomposición del Agente de Planificación 215
5.46. Interfaces del Agente de Planificación 217
5.47. Diagrama de contexto del Agente de Planificación 218
5.48. Descomposición del Agente de Trayectoria 219
5.49. Interfaces del Agente de Trayectoria 221
5.50. Diagrama de contexto del Agente de Trayectoria 222
5.51. Descomposición del Agente de Estudiante 223
5.52. Interfaces del Agente de Estudiante 225
5.53. Diagrama de contexto del Agente de Estudiante 227
5.54. Descomposición del Agente de Tutoría 228
5.55. Interfaces del Agente de Tutoría 230
5.56. Diagrama de contexto del Agente de Tutoría 231

Vistas de Uso 233
5.57. Módulos usados por el Sistema Inteligente de Tutoría 233
5.58. Módulos usados por el Agente de Tutoría 235
5.59. Módulos usados por el Agente de Estudiante 237
5.60. Módulos usados por el Agente de Trayectoria 239
5.61. Módulos usados por el Agente Mundo 241
5.62. Módulos usados por el Agente Experto 243
5.63. Módulos usados por el Agente de Simulación 245

Vistas Editor-Suscriptor 247
5.64. Inicio del Sistema 255

Vistas *Peer-to-Peer* 274
5.65. Tratamiento de una acción del estudiante 276
5.66. Tratamiento de un movimiento del estudiante 282
5.67. Tratamiento de una pregunta del estudiante 286

Vistas de Despliegue 288
5.68. Diagrama de despliegue de la aplicación 289

6. Recomendaciones Metodológicas y Criterios de Diseño 307
6.1. Vista de Uso de los agentes del SIT 310
6.2. Vista de Descomposición y Uso del SIT 311
6.3. Descomposición del sistema 314
6.4. Propagación de dependencias en la descomposición 319
6.5. Envío de mensajes entre agentes 319
6.6. Variación del patrón Editor-Suscriptor 322
6.7. Confirmación de recepción de mensajes 323

7. Evaluación de la Arquitectura 331
7.1. Árbol de características de ATAM 335

8. Resultados y Conclusiones 357
8.1. Herramienta de Autor 368
8.2. Clientes utilizados para la realización de pruebas 369

A. Sobre la Implementación de la Arquitectura 385
A.1. Diseño general del EV 386
A.2. Componentes del EV 387
A.3. Diseño del Simulador 389
A.4. Diagrama de estados del Simulador 390
A.5. Diseño de la Entidad 391
A.6. Diseño del Gestor de Entidades 393
A.7. Diseño de los mensajes 396
A.8. Diseño del Simulador 397
A.9. Diseño de los mensajes de red 401
A.10.Diseño del Centro de Mensajes 404
A.11.Diseño de la Tabla de Rutas 405
A.12.Diseño de los Agentes de Comunicación 406
A.13.Diseño del observador de red 407
A.14.Mensaje del EV al SIT 410
A.15.Mensaje del SIT al EV 411
A.16.Conexión de un EV al sistema 412
A.17.Mensaje entre dos clientes 413
A.18.Comportamientos del Agente de Tutoría 415
A.19.Estructura del plan de estudios 416
A.20.Estructura del plan de actividad 417
A.21.Estructura para el seguimiento del estudiante 419

Parte I

Planteamiento del Problema

Capítulo 1

Introducción

"Try not. Do, or do not. There is no try"
— Yoda

1.1. Un Poco de Historia

El trabajo que aquí comienza contiene los resultados de una investigación que se inició hace algunos años, aunque también es la historia de una evolución que tiene sus raíces en la época en que dejé de ser sólo un estudiante de informática y me propuse hacer algo más que ir a clase y aprobar exámenes. Fue cuando entré a colaborar en el Laboratorio Decoroso Crespo - o *el deco*, como lo llamamos todos - atraído por una convocatoria de alumnos para participar en proyectos de desarrollo de software educativo.

Aún no sabía nada de ingeniería de software, inteligencia artificial ni nada parecido, pero en seguida me vi inmerso en mi primer proyecto, *PHASE*, cuyo objetivo era desarrollar una herramienta de autor para construir software educativo. Ahí es donde empezaron a aparecer ante mis ojos los Sistemas Inteligentes de Tutoría, las primeras nociones de arquitecturas software, reutilización... Al principio todo era muy confuso. Después también, pero menos.

Luego llegó el verano y, como por arte de magia, PHASE dio paso a *Amusement* y a todo un mundo nuevo de formas tridimensionales y colores a los que llamaban Entornos Virtuales. Éstos traían consigo los avatares, las animaciones y los primeros agentes virtuales, equipados con un cargamento de rasgos de personalidad y estados de ánimo que los convertían en los compañeros ideales para jugar al mus o al escondite inglés.

Casi sin darme cuenta pasé otro año en su compañía, tras el cual se fue Amusement y llegó *PRVIR*. Con motivo de crear un curso sobre protección

radiológica, esos dos mundos aparentemente distintos y distantes, como son los Sistemas Inteligentes de Tutoría y los Entornos Virtuales, se combinaron en uno solo: los Entornos Virtuales de Entrenamiento, que han estado conmigo desde entonces.

En aquel momento no lo sabía, pero agazapados en algún escondrijo del laboratorio esperaban su oportunidad de hacer acto de presencia unos nuevos compañeros de viaje: los agentes software. Habían estado escondidos en un proyecto de fin de carrera durante algún tiempo, cogiendo polvo. Pero cuando el proyecto *MAEVIF* comenzó a gestarse hicieron su aparición estelar. Su misión desde aquel momento consistió en intentar hacernos la vida más fácil para que los Entornos Virtuales de Entrenamiento se convirtiesen en algo más sencillo de desarrollar y más ameno de utilizar.

No quiero dejar de mencionar a *Steve*, ese amigo aventajado al que conocí en una pausa en el camino y que, aunque no hace exactamente lo mismo que tú, tiene las ideas claras y te ayuda a encontrar lo que buscas, aunque no sepa del todo a dónde vas.

Y con todo este inventario de sofisticadas herramientas, curiosas experiencias y a veces extraños compañeros de viaje, se ha intentado realizar un trabajo que arroje algo más de luz en ese mundo en el que viven todos ellos: los Entornos Virtuales de Entrenamiento con Tutoría Inteligente.

1.2. Problema a Resolver

Los sistemas de Enseñanza Asistida por Ordenador son casi tan viejos como los primeros ordenadores, y desde sus orígenes han involucrado en su desarrollo a personas con muy diversos perfiles: técnicos, psicólogos, expertos en las materias a enseñar... Este es un hecho que ha enriquecido enormemente estos sistemas, pero que no está exento de problemas.

Uno de los integrantes de esta gran familia de aplicaciones son los Sistemas Inteligentes de Tutoría, cuyos requisitos van un poco más allá que los de una simple presentación de conceptos y ejercicios, adentrándose en el mundo de la Inteligencia Artificial. Así, a estos sistemas se les pide que se comporten de una manera más sofisticada, llamémosla *inteligente*, y que muestren la capacidad de adaptar la enseñanza a las acciones y características de los estudiantes, en lugar de avanzar de una manera lineal.

Esta característica, que hemos podido expresar en unas pocas líneas, presenta una complejidad bastante elevada y plantea numerosos retos que están aún por resolver de manera satisfactoria.

Uno de los problemas con el que nos hemos topado a lo largo de los años es la dificultad que conlleva reutilizar partes desarrolladas para un sistema en

otro sistema distinto, ya sea uno existente u otro de nueva creación. Si bien la idea que sustenta estas aplicaciones es precisamente la de poder intercambiar distintos elementos a partir de la separación en módulos del conocimiento a transmitir, la forma de enseñarlo y lo que se sabe del estudiante, lo cierto es que en la práctica esta reutilización no resulta sencilla.

La aparición de los Entornos Virtuales y su inclusión en el mundo de la enseñanza no ha hecho sino complicar una situación que ya era compleja sin ellos. El estudiante ya no interactúa a través de una interfaz gráfica, sino que el objetivo consiste en manipular desde su interior lo que aparece en esa interfaz. Así, a los elementos existentes se añade otro más con una complejidad inherente comparable a la que ya de por sí tenían los Sistemas Inteligentes de Tutoría. Y como no podía ser menos, los Entornos Virtuales han aportado al panorama un nuevo grupo de expertos en su desarrollo, como son diseñadores gráficos y desarrolladores técnicos especializados en estos sistemas.

El último elemento que se ha unido a este conjunto de tecnologías, y que también aporta su dosis de complejidad, son los agentes software. Provenientes del mundo de la inteligencia artificial, los agentes son una herramienta que ofrece grandes posibilidades y que presenta unas características a las que, hasta la fecha, no se ha sacado todo el partido que ofrecen.

En concreto, los agentes se están adoptando de una manera relativamente rápida porque, con ellos, los sistemas son más fácilmente reutilizables. Esto, como se verá a lo largo de este trabajo, es algo que muchos autores afirman, aunque pocos lo demuestran.

La situación actual, en resumen, incluye una amalgama de tecnologías variopintas, cada una de las cuales ha aportado su propio grupo de profesionales con sus particulares idiosincrasias, y donde la ingeniería de software ha hecho tímidos esfuerzos por introducirse, pero con un éxito relativo hasta la fecha.

Parte del limitado éxito de la ingeniería de software en este campo se debe a la escasa percepción de su necesidad. Habitualmente nos encontramos inmersos en un mundo de investigación en el que las pruebas y los prototipos raras veces van más allá de la puesta en práctica un nuevo modelo. Por lo tanto, el diseño no es tan importante, o al menos eso pensamos.

Además, no se diseña la arquitectura de los sistemas porque ya se dispone de la arquitectura estándar de los Sistemas Inteligentes de Tutoría. No se diseñan sistemas reutilizables porque la teoría de los Sistemas Inteligentes de Tutoría, por un lado, y de los agentes, por otro, afirman que los sistemas que se basen en ellas serán reutilizables.

La realidad resulta ligeramente distinta. La cantidad de tecnologías que han de utilizarse para construir un sistema de este tipo hace que sea prácticamente imposible que un sólo grupo de investigación pueda especializarse en todas ellas. Ello conlleva que, o bien alguno de los elementos desarrollados presenta una calidad inferior a la de los demás, o bien se necesita utilizar elementos desarrollados por terceras partes expertas en ese área.

Esta segunda opción puede ser la más adecuada, si bien no es la más sencilla de las dos. En muchos casos no es fácil encontrar, por ejemplo, un componente que realice la tutoría y que pueda utilizarse en el sistema que se está desarrollando, bien porque no exista o bien porque se ha construido considerando unas necesidades específicas que no se adecuan al contexto en el que ahora se quiere utilizar. En definitiva, el componente no se ha desarrollado pensando en su reutilización en otros sistemas.

En otras ocasiones sí se consigue reutilizar el elemento necesario, pero a costa de tener que fabricar un componente más o menos complejo que sirva de unión entre el sistema y el nuevo componente y que permita realizar la integración.

Existe además otro factor del que pocas veces se habla, que es el factor humano. Desarrollar software reutilizable, o modificable, es complejo, y no siempre sabemos cómo hacerlo. Ni siquiera existe una teoría sólida que nos ayude a diseñar software modificable, por lo que aún es una tarea que tiene mucho de artesanal. Si a ello unimos que parte de la gente que trabaja en este área no tiene una formación orientada al desarrollo de software, no podemos sino esperar que el grado de reusabilidad de las aplicaciones sea bastante impredecible.

Teniendo en cuenta lo anteriormente expuesto, el interés principal para realizar este trabajo de tesis consiste en diseñar una arquitectura software, utilizando agentes, que se base en los usos y costumbres existentes en el campo de los Sistemas Inteligentes de Tutoría, pero que tenga en cuenta expresamente la necesidad de ser modificable en unos determinados aspectos que se detallarán más adelante. El objetivo de esta arquitectura se basa en conseguir que elementos de uso habitual en estos sistemas, como el Entorno Virtual, las estrategias de tutoría o los modelos de estudiante, resulten más fácilmente intercambiables de lo que lo son ahora.

El fin último que se persigue, aunque fuera del alcance de este trabajo, consiste en definir un estándar de facto que facilite el intercambio de los elementos que conforman los Entornos Virtuales de Entrenamiento con Tutoría Inteligente, labor que resulta sencilla sobre el papel pero que es poco frecuente en la realidad.

1.3. Estructura del Documento

El presente trabajo está estructurado en nueve capítulos, siendo el primero de ellos esta introducción.

En el **capítulo 2** se introducen las principales áreas de investigación relacionadas con este trabajo: realidad virtual, sistemas de enseñanza, agentes y arquitecturas software. También se realiza un estudio de algunos de los proyectos que han llevado a cabo distintos grupos de investigación dentro de estas áreas, prestando especial atención a aquellos proyectos que abarcan más de una de las áreas citadas, pues resulta de interés para analizar la forma en que han resuelto los problemas encontrados en su integración.

Con la información obtenida al analizar el estado de la cuestión, en el **capítulo 3** se identifican los principales problemas a abordar en el trabajo de tesis, proponiéndose el enfoque a utilizar para su resolución.

En el **capítulo 4** se describe la primera aproximación realizada a la solución del problema planteado, tras lo cual se realiza un análisis de la misma. El resultado de dicho análisis muestra que la solución propuesta adolece de una serie de defectos que no permiten resolver el problema de manera satisfactoria.

A partir de la experiencia obtenida, en el **capítulo 5** se presenta un enfoque distinto para resolver el problema planteado, donde se hace uso del trabajo realizado en los últimos años en el área de las arquitecturas software. Al final del capítulo se ha elaborado un análisis de la arquitectura diseñada donde se muestra que no presenta los problemas que se descubrieron en la anterior.

Con el fin de que otros grupos de investigación puedan utilizar la arquitectura propuesta, y para que además se pueda hacer evolucionar la misma, en el **capítulo 6** se exponen unas recomendaciones metodológicas que pueden ser utilizadas como guía para el diseño. Además, se proporcionan unos criterios de diseño cuyo objetivo es servir de ayuda en la creación de arquitecturas para sistemas similares al tratado en el presente trabajo.

En el **capítulo 7** se realiza una evaluación de la arquitectura diseñada en el capítulo 5 a fin de obtener una visión más completa del impacto de las decisiones de diseño sobre la arquitectura. Esto se traduce en una lista de puntos sensibles, junto con los factores que pueden afectarles y las implicaciones que pueden tener sobre la arquitectura.

El **capítulo 8** traza un breve recorrido por lo que ha supuesto desarrollar el trabajo de tesis y presenta las principales aportaciones que se han producido con su elaboración.

Para terminar, en el **capítulo 9** se presentan las que, a criterio del autor, constituyen las principales líneas de trabajo futuro que pueden seguirse desde el punto en el que se ha finalizado el trabajo.

Posteriormente, en el **apéndice A** se puede encontrar la descripción correspondiente a la implementación realizada sobre la arquitectura propuesta en el capítulo 5.

Al final del documento se encuentra la bibliografía referenciada a lo largo del trabajo de tesis.

Capítulo 2

Estado de la Cuestión

"Classic - something that everybody wants to have read and nobody wants to read"
— Mark Twain

2.1. Introducción

Los Entornos Virtuales de Entrenamiento son un tipo de aplicación que requiere combinar tecnologías pertenecientes a distintas áreas de investigación para su desarrollo. Como no dejan de ser una aplicación de la informática a la enseñanza, uno de sus componentes fundamentales son los Sistemas Inteligentes de Tutoría (SIT), que es el elemento que aporta la componente pedagógica al sistema.

Como segunda pieza se encuentran los Entornos Virtuales (EV), aplicaciones tridimensionales que reproducen el entorno donde tiene lugar el entrenamiento, y que es uno de los elementos diferenciadores respecto de otras aplicaciones de la informática a la enseñanza.

A la combinación de ambos tipos de aplicaciones es a lo que se le ha dado el nombre de **Entornos Virtuales de Entrenamiento con Tutoría Inteligente**, que son Entornos Virtuales en los que un usuario puede practicar actividades que tendrá que llevar a cabo en un entorno real, pero con la particularidad de que contará con la ayuda de un tutor inteligente para guiarle en el proceso de aprendizaje.

Una tercera pieza que puede unirse a este conjunto de componentes son los llamados agentes pedagógicos, que son una representación física del tutor dentro del EV. Esta representación física resulta de utilidad en muchas ocasiones, pues a la capacidad de ofrecer explicaciones y responder preguntas

se une la posibilidad de demostrar cómo se realiza una acción y la de guiar al usuario por el entorno.

Una de las herramientas que se van a utilizar en este trabajo para diseñar el sistema objeto de estudio se encuentra en el paradigma orientado a agentes, que aunque comparte nombre con los anteriores, se diferencia significativamente de ellos. Mientras que los anteriores agentes constituyen un fin en sí mismos, éstos son piezas que se utilizan para desarrollar software, como sucede, por ejemplo, con los objetos.

A lo largo de este capítulo se presenta una descripción de estas tecnologías, lo que ayudará a tener una visión que delimite el alcance de los sistemas estudiados en la sección de trabajo relacionado.

En primer lugar se describe la evolución que han experimentado los Sistemas Inteligentes de Tutoría, seguida de una breve historia de los sistemas de Realidad Virtual.

Posteriormente se exponen los conceptos del paradigma de la orientación a agentes que se han considerado más relevantes para este trabajo, para dar paso a un análisis del estado de la cuestión en el área de las arquitecturas software, que hasta el momento no ha tenido cabida en el desarrollo de Entornos Virtuales de Entrenamiento y cuya aplicación en este área constituye una de las principales aportaciones de esta tesis.

Finalmente, se describe la manera en que se ha llevado a cabo la aplicación de la Realidad Virtual a la enseñanza y a los agentes pedagógicos.

Una vez se cuenta con una visión de las tecnologías involucradas en el desarrollo de estos sistemas, se presenta una sección de trabajo relacionado en la que se describen los principales sistemas existentes en la literatura relacionados con el objeto de estudio, prestando especial atención a su arquitectura.

Esta sección da paso a un análisis de los sistemas descritos, identificando así las características comunes a todos ellos que son de interés para la elaboración de esta tesis.

Tras este análisis, se realiza una descripción de la experiencia previa del autor en el desarrollo de sistemas que combinan EVs y tutores inteligentes, pues constituye la motivación que ha dado pie a este trabajo.

2.2. Sistemas Inteligentes de Tutoría

La historia de los Sistemas Inteligentes de Tutoría comienza junto con la de la Inteligencia Artificial entre finales de los años 50 y principios de los 60, cuando investigadores como Turing, Minsky o Newell plantean que los ordenadores pueden llegar a *pensar* de manera similar a la de los humanos.

Este planteamiento encontró su caldo de cultivo en los años posteriores a la Segunda Guerra Mundial, cuando muchos de los soldados que volvieron del frente recibieron ayuda para realizar estudios universitarios. Esto requirió la preparación de muchos de ellos para alcanzar el nivel mínimo exigido para poder acceder a la universidad, lo que a su vez obligó a buscar maneras de mejorar la eficiencia en la enseñanza. Para ello, se pensó en el uso de ordenadores para satisfacer las necesidades de la educación.

En los años 60 se crearon algunas aplicaciones de enseñanza asistida por ordenador (CAI, del inglés *Computer Assisted Instruction*) de tipo generativo (Uhr, 1969). Estos programas creaban conjuntos de problemas diseñados para incrementar las habilidades de los estudiantes en determinados dominios. Básicamente, estas aplicaciones presentaban un problema, recibían la respuesta del estudiante y rellenaban una tabla con los resultados obtenidos. En términos pedagógicos, estos sistemas no establecían ninguna diferencia entre enseñanza y aprendizaje, y asumían que si se le presentaba una información al alumno, éste la aprendería.

Entre finales de los 60 y principios de los 70 los investigadores empezaron a considerar al estudiante como un factor más dentro del sistema (Suppes, 1967), y comenzaron a aparecer aplicaciones que presentaban información distinta al usuario en función de sus respuestas anteriores. Para ello, los programadores tenían que saber por adelantado las posibles respuestas de los estudiantes, para así poder decidir qué hacer en cada caso.

En el año 1970 se publica el que es considerado por muchos investigadores como el primer artículo sobre Sistemas Inteligentes de Tutoría (SIT) (Carbonell, 1970). En él se describe un sistema denominado SCHOLAR, un conjunto de programas basados en el diálogo del estudiante con la computadora, que utiliza una red semántica para representar el conocimiento y para organizar el diálogo con el estudiante, en lugar de utilizar ejercicios, respuestas y baterías de errores predefinidas.

En esta época, la influencia de Piaget comienza a hacerse notar, y la teoría del *constructivismo* se abre camino como base para los métodos de enseñanza/aprendizaje (Piaget, 1954).

A comienzos de los 80, Sleeman y Brown realizan una revisión del estado de la tecnología en el área de los Sistemas Inteligentes de Tutoría (Sleeman y Brown, 1982), en la que se utiliza por primera vez este término para distinguir los sistemas no deterministas de los sistemas anteriores, a los que siguen catalogando como CAI. Se asume ya de manera implícita que el mecanismo de aprendizaje es el de "aprender haciendo" (*learning by doing*).

Es también aquí donde por primera vez aparece el concepto de *modelo de estudiante* para describir la representación del alumno dentro del sistema, que clasifican en tres tipos: *superposición*, en la que el modelo

del estudiante es un subconjunto del conocimiento experto; *diferencial*, que modela el conocimiento del estudiante como subconjunto del experto y resalta las diferencias entre el conocimiento del estudiante y el del experto; y *perturbación*, que representa los desvíos del estudiante respecto al conocimiento del experto. Estos modelos contrastan con modelos previos, como el propuesto por (Brown y Burton, 1978), que identifica errores concretos que pueden cometer los alumnos. Este modelo se encuentra implementado en el sistema DEBUGGY (Burton, 1982).

A mediados de los 80 el constructivismo se ha convertido en la corriente dominante en la psicología educativa, y una nueva revisión del estado de la tecnología realizado por Wenger muestra lo mucho que han cambiado las cosas en los pocos años transcurridos desde la publicación del trabajo de Sleeman y Brown (Wenger, 1987). En este trabajo, Wenger presenta un modelo de Sistema Inteligente de Tutoría que consta de cuatro módulos: conocimiento pedagógico, conocimiento del dominio, modelo del estudiante e interfaz, que conforman la estructura que hoy en día se conoce como la arquitectura clásica de los Sistemas Inteligentes de Tutoría.

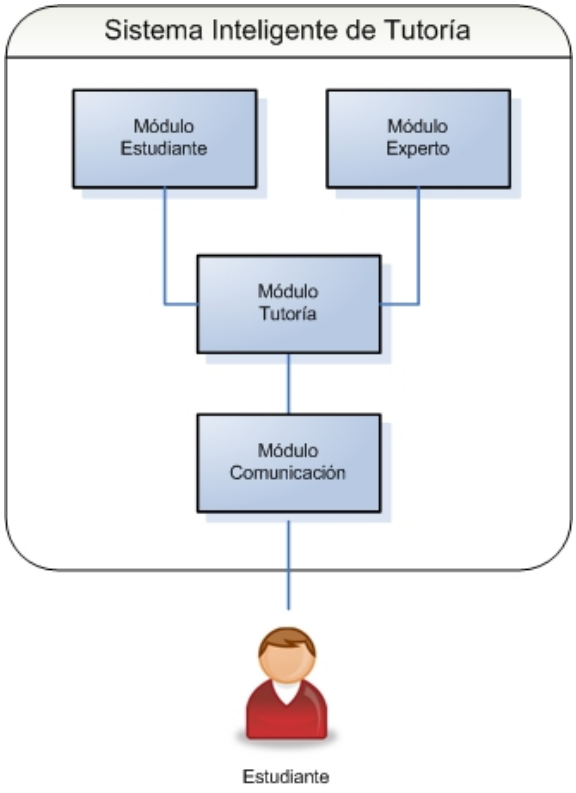

Figura 2.1: Arquitectura clásica de un SIT

El *conocimiento del dominio* es un modelo dinámico de la materia a enseñar, que contiene tanto la representación del conocimiento como la manera de razonar sobre él, lo que permite generar distintas soluciones a un problema en lugar de manejar una solución prediseñada.

El *modelo del estudiante* es la principal característica que permite distinguir los Sistemas Inteligentes de Tutoría de la Enseñanza Asistida por Ordenador, ya que permite adaptar la enseñanza a cada estudiante. Según Wenger, los modelos de estudiante deben recopilar datos acerca de sus acciones, utilizarlos para construir un modelo del conocimiento del estudiante, así como de su forma de aprendizaje, y hacer uso de estos modelos para diagnosticar qué sabe el estudiante y así poder decidir la mejor estrategia pedagógica con la que continuar la enseñanza.

En cuanto al *conocimiento pedagógico*, debe servir para realizar el diagnóstico pedagógico del alumno y para dirigir el proceso de enseñanza, para lo cual se requiere que observe el comportamiento del alumno y se adapte a él.

Finalmente, la *interfaz* permite la comunicación entre el alumno y el sistema, que según Wenger debe facilitar la eliminación de las ambigüedades en las respuestas de los estudiantes, ya que esto facilita el resto de tareas que debe llevar a cabo el SIT.

Desde el trabajo publicado por Wenger hasta la actualidad, los Sistemas Inteligentes de Tutoría han experimentado notables avances en todas sus áreas, relacionadas tanto con el aprendizaje como con la construcción de estos sistemas. Sin embargo, para el objetivo de esta tesis, el interés se centra en la utilización de los SITs junto con aplicaciones de realidad virtual para realizar entrenamientos procedimentales, motivo por el cual se cierra aquí esta breve introducción a los Sistemas Inteligentes de Tutoría. No obstante, en las siguientes secciones se continúa con el estudio de los Sistemas Inteligentes de Tutoría, aunque en un terreno más cercano al ámbito del presente trabajo, como es el desarrollo de Entornos Virtuales de Entrenamiento con Tutoría Inteligente.

2.3. Realidad Virtual y Entornos Virtuales

A pesar de que el concepto de Realidad Virtual (RV) parece de creación bastante moderna, lo cierto es que sus orígenes, al menos en el campo de la informática, datan de hace aproximadamente medio siglo.

Existe un consenso bastante unánime en otorgar a *Sensorama*, creado en 1956 por Morton Heilig, el título de primera aplicación de Realidad Virtual, que combinaba una mezcla de videoproyección, audio, vibraciones,

viento y olores, y estaba diseñado para hacer sentir al usuario que formaba parte de una película. Patentado en 1961, el sistema permitía al usuario experimentar diversas sensaciones mientras recorría uno de los cinco escenario posibles: un paseo en moto por Nueva York, un paseo en bicicleta, un paseo en un todoterreno por una duna, un vuelo en helicóptero sobre Century City o una danza del vientre. Puesto que los gráficos por ordenador aún estaban por crear, toda la experiencia se basaba en imágenes cinematográficas pregrabadas.

Heilig también patentó el que muchos consideran como el primer casco de Realidad Virtual en 1962, que daba la posibilidad de ver imágenes en tres dimensiones (3D), escuchar sonido en estéreo y percibir olores.

Además, propuso la idea de un teatro inmersivo que permitiría la proyección de imágenes en 3D, sonido estéreo y olores propagados por el sistema de ventilación, si bien esta idea no llegó a materializarse.

En 1961, Corneau y Bryan, empleados de Philco Corporation, construyeron el que parece ser el primer casco de RV de verdad. Este dispositivo permitía ver imágenes en movimiento y estaba dotado de un sensor magnético que determinaba la orientación de la cabeza del usuario. Estaba conectado a un sistema de vídeo controlado a distancia para visualizar situaciones peligrosas.

La utilización de gráficos a través del ordenador tuvo que esperar aún algún tiempo, y se debe al trabajo realizado en el MIT por Roberts y Sutherland. Roberts escribió el primer algoritmo para eliminar superficies oscuras y ocultas de una imagen, abriendo así el camino a la utilización de gráficos 3D. Por su parte, el trabajo de Sutherland consistió en el desarrollo de algoritmos que pudiesen realizar esta tarea de manera eficiente. Uno de los frutos de estos esfuerzos se encuentra en el desarrollo por Henri Gouraud, en el año 1971, de un algoritmo de iluminación que aún es muy utilizado hoy en día. Este algoritmo hace posible que una superficie formada por polígonos cobre el aspecto de una superficie suave y continua.

Mientras tanto, Sutherland también había desarrollado, en el año 1965, el conocido como *Ultimate Display* (Sutherland, 1965), un casco de RV dotado de visión estereoscópica y de un sistema mecánico de seguimiento. Este sistema ha sido calificado por la *National Academy of Sciences* como el elemento fundamental en este nuevo campo de investigación.

Durante esos años, la labor de Sutherland fue bastante prolífica, y la red de contactos que creó en el MIT, ARPA y la Universidad de Harvard le permitieron desarrollar numerosos proyectos financiados por entes tan dispares como la CIA, el Departamento de Defensa y los Laboratorios Bell, además de los anteriores.

De estos años data también la creación de la Realidad Aumentada (RA), una combinación de RV y escenarios reales, también de la mano de Sutherland. El propósito fue un intento de realizar una aplicación práctica aplicada a la medicina, relacionada com la representación del flujo sanguíneo en modelos de las válvulas cardíacas (Sutherland et al., 1971).

En esa época, Thomas Furness comenzó a trabajar en el primer simulador de una cabina de avión para entrenar a pilotos (Furness, 1986). El problema inicial consistía en la creciente complejidad de las cabinas de estos aparatos, por lo que Furness comenzó a buscar la forma de facilitar la interacción con los pilotos. La solución fue el desarrollo de una cabina que proporcionaba información 3D a los pilotos, quienes podían controlar el aparato a través de una representación virtual del terreno con campo de visión de 120º en horizontal. Encendieron este aparato por primera vez en septiembre de 1981, y ha constituido la base para el desarrollo de los sistemas de entrenamiento militar creados a partir de ese momento.

Los primeros sistemas de RV que funcionaban en estaciones de trabajo no aparecieron hasta 1984, cuando Michael McGreevy creó VIVED (*Virtual Visual Environment Display system*), las primeras estaciones de bajo coste dotadas de un campo de visión amplio, estéreo, con sensores de posición en el casco de RV, que constituyen aún una referencia para los sistemas de hoy en día.

De forma paralela a la construcción de sistemas visuales se producía también el desarrollo de los primeros guantes de RV. Uno de los primeros guantes documentados es *Sayre Glove*, desarrollado por Tom Defanti y Daniel Sandin en 1977. Este guante, basado en una idea de Richard Sayre, tenía en cada dedo un tubo flexible de fibra óptica con un emisor de luz en un extremo y un receptor en el otro. En función de la cantidad de luz que llegaba al receptor se podía calcular la flexión de los dedos.

Sin embargo, el primer guante en recibir reconocimiento fue patentado en 1983 por Gary Grimes, de los Laboratorios Bell. Este guante tenía sensores de flexión en los dedos, sensores táctiles en las yemas de los dedos y sensores de orientación y posición en la muñeca. El uso inicial que se le dio fue la interacción con el ordenador sin usar teclado, mediante la reproducción de las letras a través de la posición de la mano.

Posteriormente, Thomas Zimmerman fabricó el VPL Dataglove, que funcionaba con un mecanismo similar al del *Sayre Glove*. Su versión simplificada, el *Z-glove*, se convirtió en el primer guante de RV en ser comercializado.

A la vez que se desarrollaban estos sistemas, tenía lugar la creación de los primeros dispositivos hápticos, a cargo de Frederick Brooks, a partir de 1960. El objetivo de Brooks consistía en crear un dispositivo que sirviese para

entender las fuerzas que tienen lugar entre moléculas de distinta naturaleza. El desarrollo del proyecto GROPE-I empezó en el año 1967, y era un sistema con gráficos en dos dimensiones (2D) que representaba campos de fuerza continuos. Su evolución, el sistema GROPE II ampliaba al anterior incorporando 6 grados de libertad, con tres fuerzas y tres momentos. Este sistema se desarrolló en el año 1976, y las computadoras existentes en aquel momento sólo permitían realizar cálculos en tiempo real para modelos muy simples, por lo que el proyecto quedó parado durante casi diez años. Entonces, en 1986, se inició el desarrollo de GROPE III, que terminó en 1988, y que dio lugar a una serie de experimentos que culminaron en el año 1990, cuando se consiguió modificar en tiempo real la estructura de una molécula alrededor de 12 puntos de torsión (Frederick P. Brooks et al., 1990).

También en esos años, a principios de los 70, se desarrolló el dispositivo de posicionamiento magnético más utilizado durante muchos años, el Polhemus, que constaba de emisores transversales que inducían corrientes en un sensor receptor, que a su vez constaba de tres bobinas transversales. De esta manera, la posición y orientación del sensor podía ser calculada a partir de las 9 corrientes inducidas. Posteriormente, este dispositivo fue utilizado por George Lucas para la realización de sus películas de la *Guerra de las Galaxias*.

Estos dispositivos constituyen la base de los sistemas de Realidad Virtual que han aparecido hasta nuestros días. Con su desarrollo se llegó a los años 90, que han supuesto el despegue definitivo de las aplicaciones basadas en la Realidad Virtual, y con ellas, una ligera desvirtuación del término.

Actualmente, se entiende que un sistema de Realidad Virtual es una aplicación que proporciona una experiencia interactiva e inmersiva en un entorno tridimensional simulado (Gigante, 1993).

Como término más general se utiliza *Entorno Virtual* para definir un entorno tridimensional simulado con el que se puede interactuar por medio de un PC de escritorio, y es gracias a ellos por lo que estos sistemas han experimentado un crecimiento tan rápido desde mediados de los años 90.

A pesar del desarrollo sufrido, el alto coste de los dispositivos de Realidad Virtual hace que aún hoy no se encuentren al alcance de muchos grupos de investigación, y menos aún del público en general. Además, muchos de ellos dependen todavía de cables conectados al ordenador o de equipos que hay que llevar encima, por lo que su uso prolongado resulta bastante incómodo.

Sin embargo, los que sí han experimentado una expansión notable han sido los Entornos Virtuales, con aplicaciones que se pueden encontrar en áreas como: visualización de información científica (van Liere et al., 2002; Gherbi y Hérisson, 2001; Hérisson et al., 2004; Férey et al., 2004; Cugini et al., 2000), CAD (Toro et al., 2006), entretenimiento (Hirose, 2006; Zhu

et al., 2008; Ha y Woo, 2006; Sunar et al., 2006), Medicina (Heng et al., 2006; Park et al., 2005) o Planificación urbanística (Drettakis et al., 2007).

Estos constituyen sólo algunos ejemplos de entre la amplia gama de aplicaciones que se han construido en los últimos años. También se ha desarrollado un gran número de aplicaciones en el área de la educación, que por su relevancia para este trabajo se explica con mayor detalle en las próximas secciones.

2.4. Agentes

2.4.1. ¿Qué es un agente?

El término *agente* es una de esas acepciones que de forma inadvertida se ha ido adoptando en distintas áreas para denominar elementos con propiedades diferentes y que, por el momento, no ha logrado poner de acuerdo a los investigadores en cuanto a su significado. De hecho, probablemente el punto donde más coinciden unos y otros es en destacar que no existe una definición comúnmente aceptada del significado de este término.

Casi todos los autores coinciden en que son entidades software con autonomía, pero a partir de ahí cada autor añade características que considera imprescindibles para los agentes que no son compartidas por otros autores. Algunas de las definiciones más citadas se presentan a continuación:

(Russell y Norvig, 1995) *An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.*

(Maes, 1995) *Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed.*

(Smith et al., 1994) *Let us define an agent as a persistent software entity dedicated to a specific purpose. "Persistent" distinguishes agents from subroutines; agents have their own ideas about how to accomplish tasks, their own agendas. "Special purpose" distinguishes them from entire multifunction applications; agents are typically much smaller.*

(Hayes-Roth, 1995) *Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions.*

(Franklin y Graesser, 1996) *An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.*

(Garijo, 2002) *El concepto de "agente" caracteriza a una entidad software con una arquitectura robusta y adaptable que puede funcionar en distintos entornos o plataformas computacionales y es capaz de realizar de forma "inteligente" y autónoma distintos objetivos intercambiando información con otros agentes humanos o computacionales. Las características destacables del comportamiento de un agente son:*

- *Funcionamiento continuo y autónomo.*
- *Comunicación con el entorno y con otros agentes (posiblemente humanos), por medio de un lenguaje o formalismo de comunicación.*
- *Robustez.*
- *Adaptabilidad como capacidad de realizar objetivos y tareas en distintos dominios de forma incremental y flexible.*

Otros atributos importantes, aunque no todos los agentes los tienen, son los siguientes:

- *Razonamiento y aprendizaje. Ambos aspectos son necesarios para que el agente sea capaz de comportarse inteligentemente.*
- *Movilidad. Los agentes móviles son capaces de desplazarse entre los nodos de una red y ejecutarse en distintas plataformas.*

(Wooldridge y Jennings, 1995a) *A hardware or (more usually) software-based computer system that enjoys the following properties:*

- *autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;*
- *social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language;*
- *reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;*

- *pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.*

A pesar de que estas definiciones cuentan ya con algunos años de existencia, la situación no ha variado notablemente desde que se formularon, y cuando se realiza un trabajo que utiliza agentes, una de las tareas a realizar es adoptar una definición de agente, proporcionar una definición de lo que se considera que es un agente o dejar que el lector lo interprete según su criterio.

Teniendo en cuenta que la definición de agente es bastante nebulosa, lo mismo sucede con las propiedades que aportan al software. Para algunos autores, esto los convierten en unos elementos polivalentes dotados de capacidades bastante superiores a todo lo conocido hasta el momento y que son capaces de hacer funcionar una aplicación sin más que dejarlos que se ejecuten en una máquina (Wooldridge y Jennings, 1998). Si bien es cierto que los agentes nacieron en el área de la inteligencia artificial distribuida, también es cierto que siguen siendo los desarrolladores quienes les dotan de *inteligencia* y de la capacidad de trabajar en paralelo de forma coordinada.

Situándonos en un plano un poco menos extremo, al software desarrollado con agentes se le confiere, sin mayor justificación, la capacidad de mejorar propiedades como flexibilidad, modularidad o eficiencia (Odell, 2000; Ramdane-Cherif et al., 2005). Aunque los agentes, vistos como abstracciones que se utilizan para desarrollar software, pueden facilitar que las aplicaciones construidas con ellos posean estas propiedades, también es cierto, como en cualquier otro paradigma de desarrollo de software, que es necesario hacer un esfuerzo de diseño para que estas propiedades estén presentes.

Debido a esto, existe una notable confusión respecto a lo que aportan los agentes al desarrollo de software, por lo que, a los efectos de este trabajo, se intentarán tener en cuenta las afirmaciones de otros autores que ya se han enfrentado a estos problemas:

- Los agentes inteligentes son un noventa y nueve por ciento informática y un uno por ciento Inteligencia Artificial (Etzioni, 1996).

- Al desarrollar un sistema basado en agentes, el porcentaje del diseño que es específico de la orientación a agentes es comparativamente pequeño (Wooldridge y Jennings, 1998).

Consideraremos que un agente es una herramienta de abstracción para desarrollar software que nos permite situarnos un escalón por encima de objetos y componentes (Zambonelli et al., 2003) y que, utilizados de forma adecuada, facilitan la construcción de sistemas complejos.

2.4.2. El Estándar FIPA

Dos de los problemas que se deben resolver cuando aparece una nueva tecnología son la facilidad para interconectar un sistema con otros sistemas basados en ella y la posibilidad de extender la mencionada tecnología, ampliando la funcionalidad del sistema existente. Para conseguirlo es necesario disponer de estándares, aunque sean *de facto*, cuya utilización posibilite la consecución de estos objetivos.

En el área de los sistemas multiagente, quien más ha trabajado en esta dirección ha sido FIPA (*Foundation for Intelligent Physical Agents*), organización integrada dentro de IEEE que se ha encargado de elaborar una serie de especificaciones que deben cumplir las plataformas de gestión de sistemas multiagente. Existen algunas plataformas que se han desarrollado a partir de estas especificaciones, como JADE y FIPA-OS, y algunas otras tienden a adoptarlas para conseguir que los usuarios sigan desarrollando sus aplicaciones con ellas.

Son dos los principios que han guiado la elaboración de las especificaciones FIPA. Por un lado, conseguir que los sistemas sean completamente abiertos, de manera que sistemas heterogéneos puedan interactuar como sociedades de agentes. Por otro lado, sólo se han definido las interfaces de los distintos elementos, para que cada equipo de desarrollo pueda tomar las decisiones de diseño que crea convenientes, siempre y cuando se siga cumpliendo con la especificación.

El núcleo de la especificación lo constituye la plataforma de agentes, la cual proporciona toda la infraestructura y servicios necesarios para poder crear un sistema multiagente dentro de ella.

El estándar FIPA para la arquitectura de plataformas multiagente (IEEE-FIPA, 2002a) define los servicios que deben proporcionar las plataformas de agentes: un sistema de gestión de agentes (*Agent Management System*), un sistema encargado del transporte de mensajes entre agentes (*Internal Platform Message Transport*), un servicio de directorio o páginas amarillas (*Directory Facilitator*) y un canal de comunicación para los agentes (*Agent Communication Channel*).

Además, para la comunicación que tiene lugar entre los agentes se ha definido el lenguaje estándar FIPA ACL (*Agent Communication Language*) (IEEE-FIPA, 2002b), derivado del lenguaje KQML (*Knowledge Query and Manipulation Languaje*) (Finin et al., 1993; Labrou y Finin, 1997), que define la estructura que deben tener los mensajes que se envíen entre agentes. Un mensaje ACL contiene una serie de elementos de mensaje, como la identidad del emisor y del receptor, algunos de los cuales son de uso opcional. Además, se permite el uso de elementos no definidos en el estándar. El único elemento

que es de inclusión obligatoria es la *performativa* o acto comunicativo, que describe la intención del mensaje (p.e. realizar una pregunta, solicitar un servicio o proporcionar información), y en función del acto comunicativo variarán el resto de elementos del mensaje.

2.4.3. Metodologías de Desarrollo con Agentes

Existen ya un buen número de publicaciones que abordan temas como qué son los agentes y los sistemas multiagente, cómo desarrollarlos y cómo implementarlos (Weiss, 1999; Mas, 2005; Bellifemine et al., 2007).

También, como no podía ser de otra manera, han surgido diversas metodologías de desarrollo orientadas a agentes que en mayor o menor medida ayudan en la construcción de estos sistemas. Una de las principales diferencias entre ellas se sitúa en los distintos puntos de vista utilizados para diseñar el sistema.

Algunas de ellas, como *Tropos* (Giorgini et al., 2004; Bresciani et al., 2004), se basan en la utilización de UML para modelar los sistemas basados en agentes. Esta metodología define cuatro fases para realizar el diseño del sistema: análisis de requisitos preliminar, análisis de requisitos final, diseño arquitectónico y diseño detallado. En la tercera fase, de diseño arquitectónico, proponen la utilización de estilos arquitectónicos para definir la organización y, consecuentemente, el diseño de la arquitectura.

Una de las primeras metodologías orientadas a agentes que se desarrollaron es AAII/BDI (Kinny et al., 1996), que considera dos puntos de vista, uno externo y otro interno, para proponer el desarrollo de sistemas basados en agentes. Con el punto de vista externo se identifican los agentes existentes o *modelo de agentes* y las relaciones entre ellos o *modelo de interacciones*. El punto de vista interno está basado en el modelo BDI para desarrollar agentes (Haddadi y Sundermeyer, 1996).

Existen más metodologías que consideran distintos puntos de vista para desarrollar el sistema, como por ejemplo la conocida como Ingeniería de las Vocales (Demazeau et al., 2001), que considera cinco puntos de vista: Agente, Entorno, Interacciones, Organización y Usuario (esta última de más reciente incorporación). Para desarrollar cada uno de estos aspectos se pueden utilizar distintas técnicas, y el objetivo es desarrollar librerías que permitan solucionar cada uno de dichos aspectos. Además, dependiendo del sistema que se vaya a desarrollar, se propone considerar las vocales en distinto orden, dando más importancia a las vocales correspondientes a los aspectos primordiales del sistema. Si lo más importante es la interacción entre agentes, se comenzará modelando la organización. Si, por contra, se

comienza modelando a los agentes, la organización emergerá como resultado de la interacción entre los agentes individuales.

También en la metodología MAS-CommonKADS (Iglesias et al., 1998), derivada de la metodología CommonKADS para desarrollo de sistemas de gestión del conocimiento (Schreiber et al., 1999), se considera la utilización de distintos puntos de vista. Esta es la primera metodología que incorpora la notación de la orientación a objetos, en concreto de la metodología OMT, para estructurar el sistema.

MESSAGE es una metodología que propone la integración de características que aparecen en otras metodologías y resultados de investigación (Caire et al., 2001), y considera cinco puntos de vista similares a los de la ingeniería de las vocales. La novedad radica en el uso de metamodelos para describir los aspectos relacionados con los puntos de vista.

Una extensión de los metamodelos de esta metodología ha dado lugar a INGENIAS (Pavón y Gómez-Sanz, 2003), que incorpora además una serie de herramientas construidas sobre esos metamodelos (Gómez-Sanz y Pavón, 2006). INGENIAS considera también cinco puntos de vista para desarrollar el sistema, pero sustituye la U de las vocales (el usuario) por un punto de vista de *tareas y objetivos*.

Hay otras metodologías, como Gaia (Zambonelli et al., 2003), que en lugar de considerar el sistema desde distintos puntos de vista se centran en los roles que desempeñan los agentes y sus interacciones. Los roles están definidos por varias propiedades, y el análisis en esta metodología consiste en identificar posibles roles junto con sus propiedades. A partir de ellos, la fase de diseño produce un *modelo de agentes*, un *modelo de servicios* y un *modelo de conocidos*. A esta metodología se le achaca que los modelos producidos tienen un nivel de detalle demasiado alto, y que aún se debe profundizar en el diseño para poder implementar el sistema. A pesar de ello, es una de las metodologías más citadas y utilizadas.

En la misma línea que Gaia se desarrolla MaSE (DeLoach et al., 2001), que a diferencia de Gaia sí aborda todo el ciclo de desarrollo hasta la construcción del sistema. Sin embargo, MaSE está más cercana a la orientación a objetos, y considera a los agentes como una especialización de los objetos. Varias fases de la metodología se corresponden con fases de metodologías orientadas a objetos, como un análisis de casos de uso y la consiguiente elaboración de diagramas de secuencia. Un aspecto al que MaSE presta atención son las conversaciones entre agentes, que se definen como protocolos de coordinación, para lo cual es necesario haber definido antes los roles que pueden adoptar los agentes. Al igual que otras metodologías, MaSE ha dado lugar a la elaboración de una herramienta, agentTool, que ha posibilitado una expansión bastante rápida de esta metodología.

La metodología ZEUS propone el desarrollo del sistema en cuatro fases (Collis et al., 1998): el análisis del dominio, el diseño de los agentes (que se divide en tres subfases), la realización de los agentes y el soporte en tiempo de ejecución. ZEUS cuenta con bastante difusión debido, en parte, a que proporciona una herramienta que soporta las fases finales del desarrollo del sistema. Hasta la aparición de MaSE, Zeus constituía la herramienta de desarrollo de referencia, pero la mayor completitud de MaSE como metodología ha desplazado a ZEUS a un segundo lugar.

2.5. Arquitecturas Software

El campo de las arquitecturas software, como área de investigación, es relativamente nuevo. En (Shaw y Clements, 2006b; Shaw y Clements, 2006a) se hace un repaso de la historia de esta disciplina, y aunque la primera mención de arquitectura software data del año 1969, cuando Ian P. Sharp utilizó el término en una conferencia de la OTAN (Randell y Buxton, 1970), no es hasta mediados de los años 90 cuando de verdad ha empezado a despegar, considerando los expertos en el área que es aproximadamente a partir del año 2000 cuando ha alcanzado su mayor apogeo (Shaw y Clements, 2006a; Kruchten et al., 2006).

Se define **arquitectura software** como la estructura del sistema, que comprende elementos software, las propiedades de esos elementos visibles externamente y las relaciones entre ellos. La arquitectura software conforma el esqueleto de cualquier sistema software, y es la principal responsable de los atributos de calidad del sistema. Una arquitectura adecuada, correctamente diseñada, documentada y evaluada, constituye la base para que un proyecto finalice con éxito (Bass et al., 2003).

Esta definición implica que la arquitectura de un sistema define componentes y la interacción existente entre ellos, pero no detalles internos a esos componentes, que se puede considerar que no pertenecen a la arquitectura de la aplicación. Además, la arquitectura del sistema se puede ver desde un número no determinado de perspectivas, todas ellas válidas siempre que valgan para algún fin: análisis, comunicación o comprensión.

Se define **estilo arquitectónico** como la abstracción de distintas arquitecturas software (Klein et al., 1999). Así, las conocidas como arquitectura cliente-servidor o arquitectura en tres capas serían, en realidad, estilos arquitectónicos.

Uno de los grupos de investigación más fuertes en este área se encuentra en el *Software Engineering Institute* (SEI) de la Universidad Carnegie

Mellon, que genera gran parte de la producción científica relacionada con las arquitecturas software.

Dentro de este grupo se han desarrollado diversos métodos para diseñar, documentar y evaluar arquitecturas software, como pueden ser el *Attribute-Driven Design* (ADD), *Attribute-Based Architectural Styles* (ABAS), *Quality Attribute Workshop* (QAW), *Active Reviews for Intermediate Designs* (ARID), *Architecture Tradeoff Analysis Method* (ATAM), *Cost-Benefit Analysis Method* (CBAM) y *Views and Beyond* (V&B). Resultan especialmente interesantes porque no son técnicas independientes, sino que se proporcionan formas de combinar unas con otras (Nord et al., 2004).

ADD es un método de diseño arquitectónico dirigido por los atributos de calidad que se quiere que posea el sistema, más que por la funcionalidad de la aplicación, que queda en un segundo nivel (Bass et al., 2001). Está basado en la utilización de dos elementos:

- Escenarios Generales: son descripciones generales de situaciones que afectan a cada atributo de calidad por separado. Para cada atributo de calidad se han identificado una serie de escenarios que son válidos para todos los sistemas, aunque eso no quiere decir que sean aplicables en todos los casos. De esta forma, es necesario filtrar los escenarios aplicables y los no aplicables. Posteriormente, es necesario particularizar el escenario para cada sistema en cuestión.

- Primitivas de Atributo: son un conjunto de componentes y conectores que conforman un elemento mínimo que sirve para mejorar un atributo de calidad. Por ejemplo, una *cache* sirve para mejorar el rendimiento de un sistema. Aunque cada primitiva está orientada a mejorar un atributo, puede tener efectos secundarios sobre otros atributos. Por tanto, para cada primitiva de atributo es necesario identificar qué escenarios satisface y cuáles son sus posibles efectos secundarios.

Las entradas para el método ADD son los requisitos de calidad del sistema expresados en forma de escenarios generales, y la salida es una arquitectura conceptual (Hofmeister et al., 2000), la cual ayuda a alcanzar los atributos de calidad deseados y proporciona la estructura necesaria para dotar al sistema de la funcionalidad requerida.

ABAS (Klein y Kazman, 1999), por su parte, es a la arquitectura lo que los patrones de diseño a los objetos, es decir, son soluciones basadas en estilos arquitectónicos que han surgido de la experiencia de resolver problemas que se presentan de manera frecuente. ABAS utiliza los atributos de calidad para definir un marco de aplicación de un estilo arquitectónico concreto,

proporcionando un razonamiento, cuantitativo o cualitativo, que fundamente la utilización de un estilo arquitectónico en un diseño determinado.

ATAM (Kazman et al., 2000) es la evolución de un método anterior llamado SAAM (*Software Architecture Analysis Method*) (Kazman et al., 1994), para el análisis de arquitecturas software basado en la utilización de escenarios, en el que la evaluación de una arquitectura no se realiza para identificar de manera precisa el comportamiento de un atributo de calidad, lo cual no es posible en fases tempranas del diseño por falta de información, sino para descubrir qué decisiones de diseño afectan de una u otra forma a los atributos de calidad. Esto se hace a través de la identificación de:

- Riesgos: decisiones aplazadas o decisiones cuyo efecto no se alcanza a valorar.
- Puntos sensibles: partes de la arquitectura que pueden tener mucha influencia en algún atributo de calidad.
- Puntos de compromiso: partes de la arquitectura cuya modificación significa mejorar algún atributo de calidad a costa de empeorar otro. Es lo que sucede, en algunos casos, con la modificabilidad y el rendimiento.

El objetivo es poder tomar decisiones razonadas acerca del diseño para, en sucesivos análisis, dedicar más esfuerzo a completar esas partes de la arquitectura.

ARID (Clements, 2000) es un método de análisis de arquitecturas resultado de combinar *Active Design Reviews* (ADR, (Parnas y Weiss, 1985)) y ATAM. Resulta más ligero que ATAM y es útil para ser aplicado en etapas más tempranas del diseño arquitectónico e, incluso, sobre partes del sistema en lugar de sobre el sistema completo. La evaluación se basa en detallar el funcionamiento del sistema en una serie de escenarios predefinidos, a través de lo cual se pueden identificar posibles puntos problemáticos de la arquitectura. Su utilidad no se encuentra en la sustitución de ATAM, sino que más bien prepara el camino en sistemas en los que, por su complejidad, puedan ser necesarias revisiones de la arquitectura en etapas intermedias del diseño.

QAW (Barbacci et al., 2003) es un método creado para complementar ATAM. Su utilidad reside en la identificación de los atributos de calidad que dirigen el proceso de diseño de la arquitectura, por lo que su aplicabilidad es previa al diseño arquitectónico. Más concretamente, lo que pretende definir este método es el significado de los atributos de calidad en el contexto del sistema en desarrollo, la manera de descubrir, caracterizar y priorizar los atributos de calidad, y la forma de utilizar esta información. El resultado de este proceso es una lista de factores que van a dirigir el diseño de la

arquitectura y una lista de escenarios que servirán para evaluar los atributos de calidad.

CBAM (Kazman et al., 2002) es un método para evaluar los beneficios, costes y riesgos de las diferentes decisiones que se toman para diseñar la arquitectura software del sistema. Al igual que QAW, también está pensado para su integración con ATAM, ya que los resultados producidos por la evaluación de la arquitectura, especialmente las estrategias arquitectónicas a seguir, se utilizan como entrada en CBAM para tomar decisiones basadas en criterios económicos.

V&B (Clements et al., 2002a) es la propuesta realizada en el SEI para documentar la arquitectura software de un sistema. De acuerdo con la definición de arquitectura como la estructura o estructuras del sistema, V&B propone la definición de una serie de vistas relevantes de la arquitectura software del sistema, documentando cada una de ellas, así como las características que afecten a más de una o a todas en general. El número y el tipo de las vistas de un sistema no está determinado a priori, aunque en general se pueden agrupar en vistas de módulos, vistas de componentes y conectores, vistas de localización y combinaciones de ellas. Para documentar las vistas de una manera sistemática y homogénea han creado unas plantillas que contienen la estructura de la información que debe aportarse sobre cada vista.

En la misma línea se mueve el estándar IEEE 1471-2000 (IEEE, 2000), que define un marco conceptual para describir arquitecturas y la información que debe incluir una documentación que cumpla el estándar. También utilizan el concepto de vista como una representación del sistema desde el punto de vista de un conjunto de intereses.

En (Clements, 2005) se realiza una comparación entre las dos formas de documentar una arquitectura, y se demuestra que utilizando V&B es posible satisfacer los requisitos impuestos por el estándar IEEE 1471-2000.

Se están iniciando trabajos dirigidos a la medida cuantitativa de los atributos de calidad de una arquitectura software (Chastek y Ferguson, 2006). Aunque todavía están en un estado muy preliminar, ya se pone de manifiesto la dificultad de tomar estas medidas, debido a las interacciones entre los atributos de calidad, a las interacciones de la arquitectura con otros elementos del proyecto, y a la utilización de vistas para documentar la arquitectura.

Otro factor que está impulsando la utilización de arquitecturas software es la adopción del Proceso Unificado de desarrollo de software. En (Kroll y Kruchten, 2003) se describe el proceso unificado como *dirigido por casos de uso y centrado en la arquitectura*, es decir, que utiliza la arquitectura software del sistema como elemento central del desarrollo para coordinar a todos los miembros del equipo. También manejan la idea del uso de vistas

para describir la arquitectura del sistema (Kruchten, 1995). Sin embargo, existen diferencias importantes respecto al trabajo realizado en el SEI.

La primera y quizá más importante es la menor claridad de los conceptos que se manejan, ya que definen repetidamente el concepto de arquitectura como *el conjunto de elementos arquitectónicamente significativos*, definición recursiva que viene a aportar poca luz al panorama de las arquitecturas software.

En segundo lugar, se alejan bastante de la importancia que pueden tener los atributos de calidad del sistema, y se llega a proponer que, cuando se está en dificultades para cumplir los plazos de un proyecto, las alternativas pasan por recortar funcionalidades menores de la aplicación o reducir la calidad de la misma. Esta cuestión no se plantearía de considerar los atributos de calidad como parte de la arquitectura, pues llegados al punto de ver que no se cumple con lo planificado, lo más probable es que los atributos de calidad ya estuviesen incluidos en el planteamiento del sistema desde etapas tempranas del desarrollo.

En tercer lugar, las evaluaciones de la arquitectura se plantean siempre sobre *arquitecturas ejecutables*, es decir, sobre prototipos del sistema o de la parte del sistema que se quiere evaluar. Esto implica, por un lado, la necesidad de implementar un prototipo para poder evaluar la arquitectura, y por otro, un gasto extra de recursos para evaluar decisiones que pueden no ser válidas, por lo que una evaluación previa evitaría el desarrollo de algunos de los prototipos y, por tanto, el gasto de recursos.

Finalmente, el modelo de las 4+1 vistas de la arquitectura marca la necesidad de documentar, para cada sistema en desarrollo, las siguientes vistas:

- Vista lógica: existe para cualquier sistema y muestra su estructura.
- Vista de proceso: válida en sistemas distribuidos y concurrentes para mostrar el paralelismo entre distintas entidades.
- Vista de implementación: organización de los elementos de implementación, como código fuente, ejecutables y librerías.
- Vista de despliegue: muestra dónde se sitúa cada componente en tiempo de ejecución y cómo se comunican entre ellos.
- Vista de casos de uso: contiene los requisitos más significativos, tanto funcionales como no funcionales, que tienen más impacto en la arquitectura. Sirve para unir y coordinar las cuatro vistas anteriores.

Como se puede ver, en este caso sí se definen las vistas que tienen que existir, limitándolas a 5, y obligando a la coordinación a través de la vista de casos de uso. Esto convierte esta manera de documentar en poco adecuada para sistemas que no sean fácilmente descritos a través de casos de uso, si bien es cierto que, de ser éste el caso, es poco probable que se utilice el Proceso Unificado como método de desarrollo.

Tanta es la diferencia entre las dos visiones, y tantas lagunas parece tener el Proceso Unificado, que ya se han realizado esfuerzos para incluir las técnicas desarrolladas en el SEI dentro de éste (Kazman et al., 2004).

2.6. Aplicación de la Realidad Virtual a la Enseñanza

Las aplicaciones basadas en Realidad Virtual, tal como hoy las conocemos, se vienen aplicando a la enseñanza desde que empezaron a desarrollarse los primeros sistemas de RV en los años 60. Sin embargo, su número se ha incrementado notablemente desde principios de los años 90, cuando los ordenadores de uso convencional han comenzado a soportar con mayor facilidad las aplicaciones con interfaces 3D.

Muchos educadores e investigadores piensan que el uso de la Realidad Virtual en la enseñanza puede aportar grandes beneficios a los estudiantes, y prueba de ello es la gran cantidad de aplicaciones de este tipo que se han desarrollado hasta la fecha.

Una cuestión que es importante tener en cuenta es cuándo se debe y cuándo no se debe hacer uso de la Realidad Virtual en un software educativo. En (Pantelidis, 1996) se propone un conjunto de criterios para decidir acerca de la aplicación de la Realidad Virtual en la enseñanza y el entrenamiento. Según sus autores, la RV se puede utilizar cuando:

- Se pueda emplear una simulación.
- La enseñanza o el entrenamiento en el mundo real pueda ser peligrosa, imposible o inconveniente.
- Pudieran producirse errores significativos por parte del alumno en el mundo real, errores que pudieran ser devastadores o desmoralizadores para el alumno, perjudiciales para el ambiente, causantes de averías en equipos o costosos.
- El modelo del entorno enseñe o entrene tan bien como la situación real.

- La interacción con el modelo sea igual o más motivadora que la interacción con la situación real.
- Se desee lograr experiencias compartidas por un grupo.
- Sea necesario hacer perceptible lo imperceptible.
- La realización de una clase atractiva requiera viajes, dinero o logística.
- Se desee enseñar tareas que involucren destrezas manuales o movimientos físicos.

Por el contrario, la Realidad Virtual no debe usarse cuando:

- La interacción con los humanos reales es necesaria.
- El entorno virtual pudiera ser física o emocionalmente dañino.
- El entorno virtual pudiera causar que el usuario confunda el modelo con la realidad.

Existe una gran diversidad de aplicaciones de los Entornos Virtuales a la enseñanza, y lo que se ha denominado como Entornos Virtuales de Enseñanza incluye aplicaciones con características dispares que, en muchos casos, lo único que tienen en común es el uso de Entornos Virtuales.

En primer lugar, se puede realizar una primera distinción en cuanto a su propósito, lo que da lugar a los Entornos Virtuales para Educación y los Entornos Virtuales de Entrenamiento.

Los primeros suelen ser sistemas desarrollados para apoyar el aprendizaje de niños en edad escolar, por lo que suelen estar enfocados hacia la adquisición de conocimientos. Además, suelen estar planteados para que puedan soportar varios usuarios, de manera que el aprendizaje se realice de forma colaborativa, y en muchos casos no incluyen un tutor inteligente, aunque sí contemplan la presencia de tutores humanos (Johnson et al., 1998a; Johnson et al., 1999b).

Los segundos suelen estar más enfocados al desarrollo de habilidades de tipo procedimental, y los principales usuarios son adultos que deben adquirir habilidades en su entorno profesional, por lo que es habitual que estos sistemas se encuentren ambientados en entornos industriales (Lozano et al., 2003), médicos (Lucas et al., 2008) y militares (Johnson y Valente, 2008).

También se puede realizar una clasificación de estos sistemas en función del tipo de tutoría que incorporan, y es aquí donde surgen las mayores diferencias. Se pueden encontrar sistemas que no incorporan ningún tipo

de tutoría, en los que los EVs se usan simplemente como herramienta de prácticas. Existen sistemas de este estilo en entrenamiento médico (Viciana-Abad y Reyes-Lecuona, 2005), militar (Loftin et al., 2004) e industrial (Hosseini et al., 2002), pero también en Entornos Virtuales educativos para niños (Popovici et al., 2005) y adultos (Asay-Davis et al., 2000).

En el caso de los sistemas médicos y militares, se da el caso de que, aunque no existan tutores inteligentes, sí puede hacer acto de presencia algún tipo de agente más o menos inteligente que hace las funciones de compañero de aprendizaje (Swartout et al., 2006; Chou et al., 2003; Muller-Wittig et al., 2001).

Por el contrario, en los EVs educativos suele ser más habitual que se incorpore algún mecanismo para que un profesor esté presente en el EV mientras los niños se encuentran en él.

Si se da el caso de que incorporen algún tipo de tutoría, se pueden encontrar sistemas con tutores más o menos cableados, cuya tutoría es poco flexible, y sistemas que incorporan tutores inteligentes que se adaptan a las características y acciones de los estudiantes.

En este último caso se pueden realizar dos distinciones, en función de la presencia o no de un tutor personalizado dentro del EV y del tipo de tutoría que realiza.

Si incluyen un tutor con presencia física en el entorno, entonces se habla de agentes pedagógicos, sobre los cuales se dan más detalles en la próxima sección.

Respecto al tipo de tutoría, en general los sistemas se pueden clasificar en dos tipos: los que se basan en el análisis de las acciones que realiza el usuario (Buche et al., 2004), que requieren de unas capacidades de planificación y adaptación complejas, y los que se basan en el diálogo con el tutor (Graesser et al., 2008), que conllevan un esfuerzo notable de modelado de las posibles situaciones que se dan en un diálogo, y que todavía implican ciertas limitaciones en cuanto al establecimiento de turnos para hablar. Existen sistemas que intentan aplicar ambos tipos de tutoría, pero aún son escasos y están poco desarrollados (Rickel et al., 2002a).

En base a las distintas características que se han descrito, los sistemas que resultan de mayor interés para el desarrollo de esta tesis son los Entornos Virtuales de Entrenamiento que incorporen tutores inteligentes, no necesariamente presentes en el EV, y que se basen en el análisis de las acciones que realizan los usuarios dentro del EV.

A pesar de ello, en la sección de trabajo relacionado se analizan sistemas de todos los tipos aquí mencionados, ya que todos ellos presentan características que puede ser interesante incluir en la arquitectura que se pretende diseñar con la elaboración de la presente tesis doctoral.

2.7. Agentes Pedagógicos

Los agentes pedagógicos son el resultado de la combinación de dos áreas de investigación como son los *agentes de interfaz animados* (Cassell et al., 2000; Hayes-Roth y Doyle, 1998; Ball et al., 1997; André, 1997) y los Sistemas Inteligentes de Tutoría (Wenger, 1987; Sleeman y Brown, 1982). La primera de las dos áreas proporciona una metáfora de interacción persona-ordenador a través del diálogo con un agente animado, mientras que la segunda se ocupa de la creación de tutores inteligentes (ver sección 2.2).

Aunque su utilización no se encuadra únicamente dentro de las aplicaciones de la RV a la enseñanza, ya que existen ejemplos de su uso en aplicaciones web (Shaw et al., 1999b), muchos de los sistemas que utilizan estos agentes lo hacen por medio de la utilización de Entornos Virtuales. La primera referencia sobre su uso data del año 1990 (Chan y Baskin, 1990), aunque su aplicación se ha extendido desde 1997, año en el que el número de publicaciones en este área comenzó a incrementarse notablemente (Lester et al., 1997a; Lester et al., 1997b; Rickel y Johnson, 1997; Alem et al., 1997; Baranauskas et al., 1997).

Los agentes pedagógicos pueden desempeñar distintos papeles dentro de una aplicación de enseñanza, y no únicamente el de tutores. Así, se han utilizado como compañeros de equipo (Chan, 1996; Rickel y Johnson, 2002; Chou et al., 2003), como generadores de estrategias pedagógicas (Person et al., 2001) o como ayudantes de instructores y compañeros (Marsella y Johnson, 1997; Rickel y Johnson, 1999).

Uno de los aspectos destacables de estos agentes es el llamado *efecto persona*, identificado en (Lester et al., 1997c), donde se muestra cómo los estudiantes mejoran su aprendizaje cuando cuentan con la ayuda de un agente pedagógico de aspecto humano con presencia física dentro del Entorno Virtual. Posteriormente, otros investigadores han identificado este efecto en estudiantes que utilizan aplicaciones con agentes pedagógicos (Baylor, 2001; Moreno et al., 2001), si bien otros investigadores atribuyen los beneficios a otros factores, como la estrategia pedagógica, más que al uso de agentes pedagógicos (Choi y Clark, 2006).

Uno de los aspectos que han centrado más la atención en éste área ha sido la posibilidad de dotar a los agentes pedagógicos de distintas capacidades que se asemejen a las humanas (Swartout et al., 2006):

- Emociones: algunos autores argumentan que la capacidad de mostrar emociones facilita la relación con el estudiante y le motiva para continuar con el aprendizaje (Elliott et al., 1999). Los agentes pueden

mostrar emociones a través de expresiones faciales, movimientos, gestos y tonos de voz.

- Comportamiento no verbal: esta característica le da al agente la posibilidad de ofrecer respuestas al estudiante sin necesidad de interrumpirle, como en el caso de asentir con la cabeza para indicar que se comprende lo que el estudiante dice. También se puede utilizar para indicarle al alumno la opinión del agente sobre una acción que ha realizado, como una negación con la cabeza cuando se ha realizado una acción incorrecta (Rickel y Johnson, 1999).

- Conversación no verbal: cuando dos personas mantienen una conversación, resulta habitual que realicen gestos que remarquen lo que se está diciendo con palabras, y el tono de voz también cambia el significado de una frase. Esta capacidad proporciona a los agentes la posibilidad de darle distinto significado o de enfatizar lo que le dicen al estudiante (Cassell, 2000).

- Conversación con lenguaje natural: aparte de las interacciones no verbales, también es deseable que los agentes pedagógicos puedan comunicarse con los estudiantes con una comunicación oral flexible que se adapte a la situación, en forma de explicaciones, preguntas o respuestas (Graesser et al., 2005b).

- Percepción: para que los agentes puedan presentar un comportamiento similar al humano, una de las características necesarias es que perciban el entorno de forma similar a como lo haría un humano (Kim et al., 2005; Herrero et al., 2005). De esta forma, deben poder ver las cosas que están en su campo de visión, pero no las que están ocultas. Igualmente, pueden captar sonidos distinguiendo su lugar de procedencia, lo que les permite, por ejemplo, girarse hacia un estudiante cuando oyen que éste les está hablando.

Además de estas características, los agentes pedagógicos aportan una serie de beneficios que los hacen útiles dentro de los sistemas de enseñanza. Por una parte, ofrecen la posibilidad de utilizarlos para demostrar la forma de llevar a cabo una determinada tarea dentro del Entorno Virtual (Rickel y Johnson, 1999). Por otra, se pueden utilizar movimientos, miradas y gestos para dirigir la atención del estudiante a un determinado punto o para guiarle dentro del EV (Noma y Badler, 1997; Lester et al., 1997b; Cassell et al., 2007).

Todas estas características hacen que los agentes pedagógicos presenten dos ventajas sobre otro tipo de sistemas: incrementan las formas de

comunicarse con un estudiante y aumentan la capacidad de la aplicación para motivar a los estudiantes (Johnson et al., 2000).

Sin embargo, la cantidad y complejidad de las características requeridas hacen que existan numerosas cuestiones abiertas a la espera de ser resueltas de manera satisfactoria, como la percepción del entorno y la interacción con él, la generación de comportamientos complejos y creíbles, el mantenimiento de un diálogo elaborado con el estudiante o el manejo eficiente de todos estos aspectos en su conjunto (Gratch et al., 2002).

2.8. Trabajos Relacionados[1]

2.8.1. Agentes y Enseñanza

2.8.1.1. Adele

Adele (*Agent for Distance Education - Light Edition*) es un agente pedagógico utilizado para enseñar a realizar diagnósticos y tratamientos médicos (Shaw et al., 1999b; Shaw et al., 1999a). Adele se ha desarrollado en el ISI (*Information Sciences Institute - University of Southern California*) y se basa parcialmente en el desarrollo de Steve (ver sección 2.8.3.1), aunque, a diferencia de éste, Adele se ha desarrollado para que funcione en un navegador web.

En el ejemplo que se ve en la Figura 2.2, el estudiante puede realizar diversas acciones sobre el paciente, como realizar preguntas sobre el historial médico, realizar exámenes físicos, ordenar la realización de pruebas y, finalmente realizar el diagnóstico.

Adele es capaz de monitorizar las acciones de los estudiantes y, en función de ellas, puede resaltar algún aspecto del caso, sugerir las acciones correctas a realizar o proporcionar pistas y material de apoyo relevante. Dado que el estado del paciente varía a lo largo del tiempo, Adele puede alertar al estudiante si aprecia que el estado del paciente no es estable y el estudiante no lo está tratando adecuadamente.

Puesto que es una aplicación basada en web, la arquitectura de Adele es, en primera instancia, de tipo cliente/servidor, aunque gran parte de la lógica de la aplicación permanece en el cliente. Esto hace que, a diferencia de muchos otros sistemas en los que la lógica de la aplicación reside en el servidor (Brusilovsky et al., 1996), la ejecución de Adele sea más dinámica,

[1]En esta sección se incluyen imágenes originales relacionadas con los trabajos descritos. Estas imágenes son propiedad de sus autores y se reproducen por motivos meramente informativos, para dotar a las descripciones de mayor claridad

Figura 2.2: Adele explicando la importancia de un examen abdominal

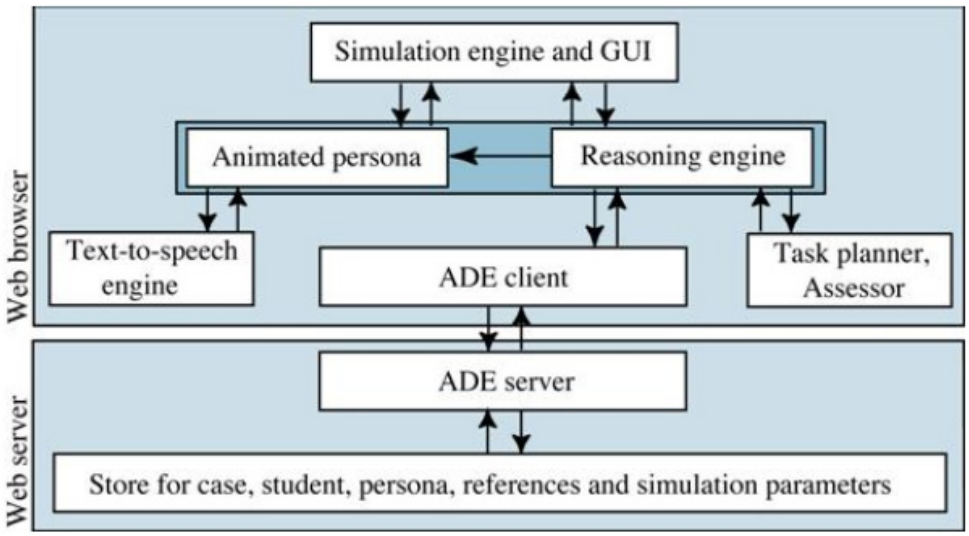

Figura 2.3: Arquitectura monousuario de Adele

al eliminarse los periodos de latencia en la comunicación con el servidor. Básicamente, el sistema se compone de cuatro módulos, como son el agente pedagógico, la simulación, el módulo de comunicación cliente/servidor y el almacén central del servidor. El agente pedagógico, a su vez, se descompone en un módulo de control de Adele y un motor de razonamiento. El almacén del servidor se utiliza para guardar los progresos de los alumnos y, en caso de utilizar entrenamiento multiusuario, para sincronizar las acciones de los distintos estudiantes.

El motor de razonamiento toma todas las decisiones de tutoría, basadas en un modelo del estudiante, un plan de la tarea a resolver, un estado inicial y el estado del agente, que se actualiza con las acciones del estudiante.

Inicialmente, el plan de la tarea a resolver se realizaba a través de un planificador jerárquico no lineal, pero sucesivas evaluaciones de la aplicación indicaron que había estudiantes que consideraban al sistema demasiado rígido, por lo que, para solucionarlo, se tomaron dos medidas. En primer lugar, se dotó a Adele de tres modos de ejecución: uno en el que no permitía desviarse del plan, otro en el que sí lo permitía y proporcionaba pistas y explicaciones el estudiante si lo solicitaba, y un tercero en el que sólo monitorizaba las acciones del estudiante, pero no intervenía. Posteriormente se sustituyó el planificador por una red bayesiana. A cada nodo de la red se le asigna una pista y una explicación, y la relación causa-efecto entre los nodos de la red se transfiere también a las pistas, de manera que los estudiantes perciben que las pistas proporcionadas conducen de una manera más natural por el camino correcto.

2.8.1.2. AutoTutor

AutoTutor es un sistema de enseñanza basado en el diálogo que simula a un tutor humano y que ha sido desarrollado en el Instituto de Sistemas Inteligentes de la Universidad de Memphis (Wiemer-Hastings et al., 1998; Graesser et al., 2001). Sus autores lo definen como un complejo sistema cognitivo por dos motivos: incorpora un modelo de conocimiento similar al humano y un proceso cognitivo, y utiliza estos mecanismos psicológicos para promover el aprendizaje (Graesser et al., 2005a; Graesser et al., 2005b).

AutoTutor basa gran parte de su capacidad de enseñanza en el diálogo con los estudiantes. Así, las preguntas planteadas por el tutor requieren respuestas que ocupan entre tres y siete frases. Según los autores, las respuestas iniciales suelen contener entre una palabra y dos frases, lo que sirve como base a AutoTutor para iniciar el diálogo con el estudiante para elaborar la respuesta.

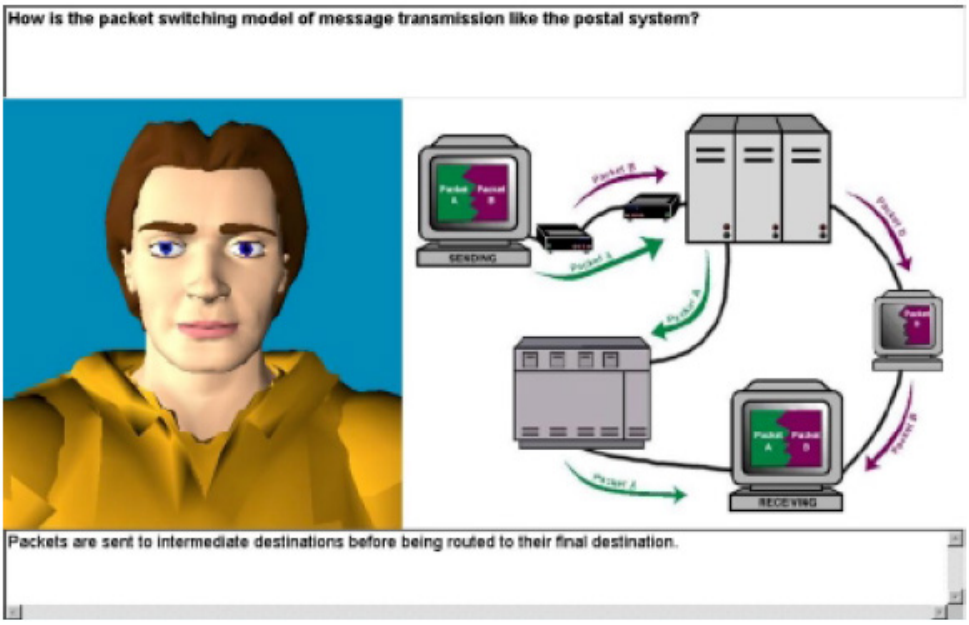

Figura 2.4: AutoTutor con Marco como tutor

El diálogo con el estudiante se extiende a lo largo de entre 50 y 200 turnos, por lo que entre las capacidades de AutoTutor se encuentra un sistema de diálogo basado en turnos y un método de procesamiento de lenguaje natural bastante elaborado.

Uno de los puntos fuertes de AutoTutor es que ha sido extensa y formalmente probado, con experimentos que abarcan alrededor de 1000 sujetos de estudio y entre dos y tres mil horas de conversación registradas.

Aunque las conversaciones entre el sistema y el alumno se realizan de manera escrita, existe un prototipo que incorpora reconocimiento de voz. Además, posee un sistema de síntesis de voz que apoya las explicaciones escritas, además de un tutor en forma de una cabeza y la parte superior del tronco con capacidades de mostrar distintas expresiones faciales, además de realizar algunos gestos y señalar.

Existe también una versión de AutoTutor que interactúa con entornos 3D, si bien la capacidad de interacción por parte del alumno no es muy elevada, y básicamente sirve para que el sistema muestre el comportamiento de un problema determinado cuando el estudiante ajusta distintos parámetros. Por tanto, se puede considerar más como una interfaz que como un entorno de aprendizaje, y los autores sostienen que no existen pruebas empíricas de los beneficios proporcionados por las simulaciones interactivas.

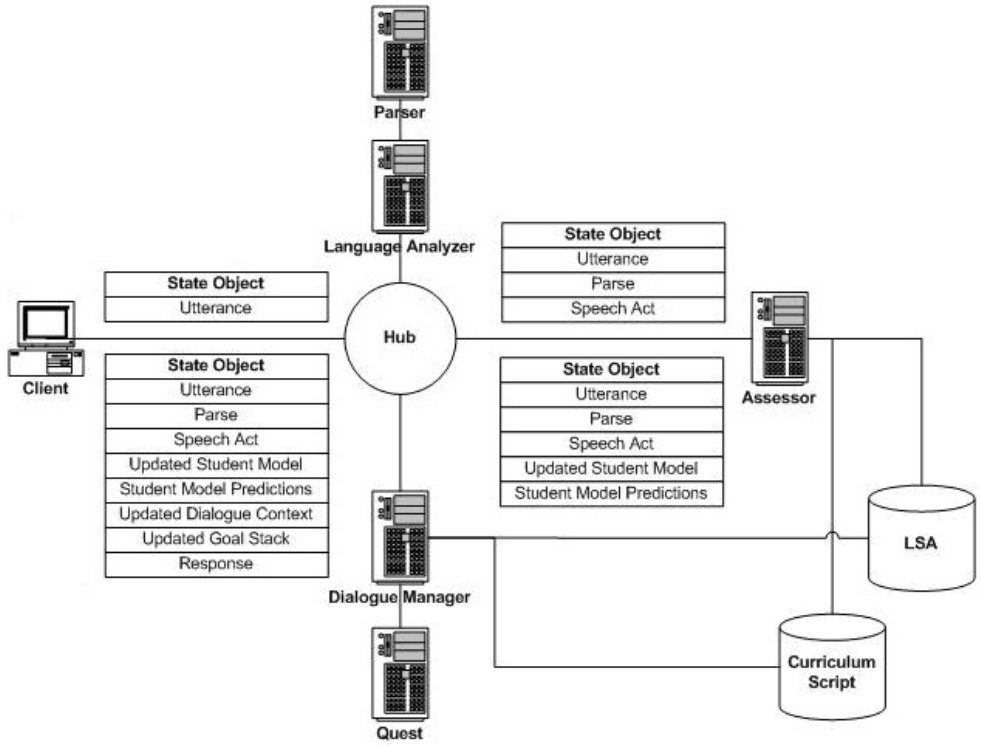

Figura 2.5: Arquitectura de AutoTutor

AutoTutor constituye posiblemente el sistema que cuenta con un mayor y más serio soporte de teorías cognitivas y del aprendizaje. Entre ellas se encuentran teorías de aprendizaje constructivista basado en explicaciones, respuestas adaptadas al conocimiento del estudiante y resultados empíricos acerca del uso de patrones de discurso.

El sistema tiene una arquitectura cliente/servidor con comunicación asíncrona, similar a un sistema web de tipo cliente ligero. Cuando un cliente se comunica con el servidor, le envía un *objeto estado* que contiene toda la información relativa al estado de la sesión: historia, acción actual, etc., de manera que el servidor es independiente del número de clientes que estén conectados a él. Aunque este enfoque posibilita que el servidor consuma menos recursos, especialmente memoria, también hace que la posibilidad de trabajar con un grupo de estudiantes sea más compleja, porque cada cliente debería tener información relativa a todos los demás clientes.

Dentro del servidor existe un *hub* central que es el único que conoce al resto de módulos. Cuando recibe el objeto estado, lo envía a cada uno de los módulos en un orden prefijado, de forma que cada cual realiza las acciones

que sean su responsabilidad. De esta forma, los módulos sólo necesitan conocer la estructura del objeto estado, y cualquier módulo que proporcione la información necesaria en el objeto estado puede trabajar con él, lo que hace que añadir o quitar módulos sea razonablemente sencillo.

Puesto que AutoTutor es un sistema de enseñanza basado en el diálogo, el elemento principal de su arquitectura lo constituye el gestor de diálogos (*Dialogue Manager*). Aunque el resto de los módulos del servidor procesen de alguna forma la información contenida en el objeto estado, el gestor de diálogos es el encargado de tomar las principales decisiones, especialmente la realización de planes acerca de cómo continuar la conversación con el alumno, así como los gestos que deben realizarse.

A pesar de la aparente sofisticación del sistema, los autores son conscientes de que AutoTutor es útil especialmente para estudiantes con pocos conocimientos y bajo nivel de aprendizaje. Esto es debido a dos motivos: en primer lugar, este tipo de estudiante es poco propenso a llevar la iniciativa del diálogo, lo cual facilita el diseño del gestor de diálogos; en segundo lugar, un estudiante con un nivel alto es más propenso a encontrar inconsistencias en el funcionamiento del sistema, lo que a su vez hace más probable que encuentre menos estimulante el aprendizaje con este sistema.

2.8.1.3. PPP Persona

Personalized Plan-Based Presenter Persona (PPP Persona) es un proyecto desarrollado en el Centro de Investigación en Inteligencia Artificial de Alemania (DFKI) (André et al., 1996; André et al., 1999). Aunque no es un agente pedagógico propiamente dicho, su propósito principal consiste en desarrollar un agente capaz de realizar presentaciones en un entorno web, lo cual le confiere la capacidad de proporcionar explicaciones a un estudiante. Tanto es así que, en la práctica, es habitual que las publicaciones del área de los agentes pedagógicos lo incluyan en el grupo de agentes pedagógicos existentes (p.e. (Johnson et al., 2000)).

Aunque existen otros agentes diseñados para realizar presentaciones (Noma y Badler, 1997; Magnenat-Thalmann y Kalra, 1995) ninguno de ellos parece haber sido utilizado con fines pedagógicos, al contrario de lo que sucede con PPP Persona (André et al., 1998), el cual ha sido también extendido para posibilitar que las presentaciones las realice un grupo de agentes (André et al., 2000).

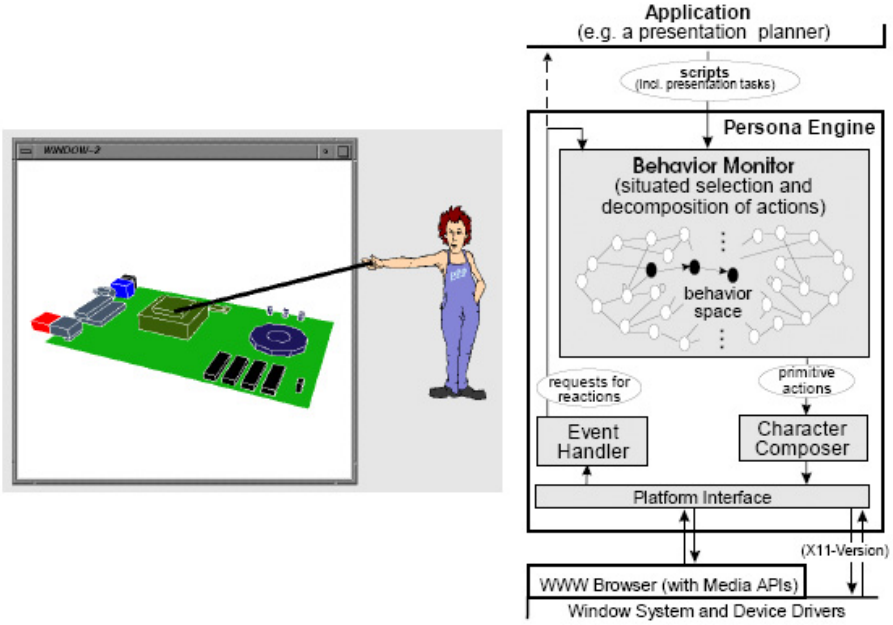

Figura 2.6: PPP Persona y su arquitectura

2.8.1.4. Vincent

Vincent es un agente pedagógico utilizado en un entorno llamado TEMAI, un sistema de entrenamiento basado en web, que se utiliza para dar formación a trabajadores de una fábrica de calzado para que aprendan a controlar las líneas de producción (Paiva y Machado, 2002; Paiva y Machado, 1998).

TEMAI consta de cuatro módulos: una base de datos de material de aprendizaje, utilizada para proporcionar explicaciones y ejercicios, un conjunto de entornos de aprendizaje, un modelo probabilístico del estudiante y Vincent, el agente pedagógico.

El conocimiento que debe ser transmitido al alumno está estructurado en forma de árbol, dividido en capítulos, y el objetivo es que se alcance un nivel mínimo en cada uno de ellos. Cada uno de estos capítulos contiene páginas de teoría, de demostración y ejercicios, y es durante la realización de los ejercicios cuando Vincent monitoriza al estudiante.

Los entornos de aprendizaje son utilizados para plantear los ejercicios a los estudiantes, y cada entorno sirve para presentar ejercicios de un mismo tipo.

El modelo del estudiante está representado con la misma estructura que el conocimiento experto, de manera que cada nodo representa la probabilidad

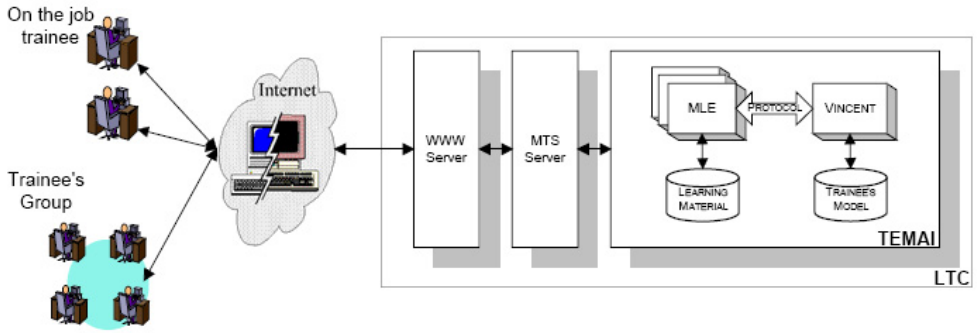

Figura 2.7: Arquitectura de TEMAI

de que el estudiante conozca un concepto. Esta probabilidad se utiliza para decidir la estrategia a seguir, y se actualiza cada vez que el usuario accede a un concepto o resuelve un ejercicio.

El encargado de mantener el modelo del estudiante actualizado es Vincent. Aunque, por lo que se ha mencionado, los módulos de TEMAI corresponden a los de un SIT, en el que Vincent sería el Tutor, la estructura no se ajusta a la de un SIT tradicional. Vincent obtiene el conocimiento de lo que debe enseñar de los entornos de aprendizaje, los cuales, a su vez, lo obtienen de la base de datos de conocimiento experto. Según los autores, este mecanismo permite que Vincent sea independiente del contexto en el que se ejecute, lo que aumenta su reutilización. Por contra, según sus explicaciones, el resto del sistema se hace dependiente de la existencia de Vincent.

Vincent puede percibir lo que sucede en los entornos de aprendizaje y actuar de acuerdo con ello. De esta manera, debe poseer un mecanismo de percepción, otro de razonamiento y otro de acción.

El módulo de acción envía peticiones e información a los entornos de aprendizaje, además de comunicarse directamente con el estudiante. Por su parte, el módulo sensor interpreta los mensajes procedentes de los entornos de aprendizaje, lo que le permite saber cuándo un estudiante inicia y acaba un ejercicio o cuándo pide ayuda.

El comportamiento de Vincent está dividido en cognitivo y físico. El cognitivo está controlado por un módulo que se encarga de las tareas pedagógicas, tareas de gestión del conocimiento experto (especialmente del modelo del estudiante) y tareas de diagnóstico. El comportamiento físico está controlado por otro módulo, que cambia la postura y las frases que dice Vincent en función de la salida del módulo cognitivo.

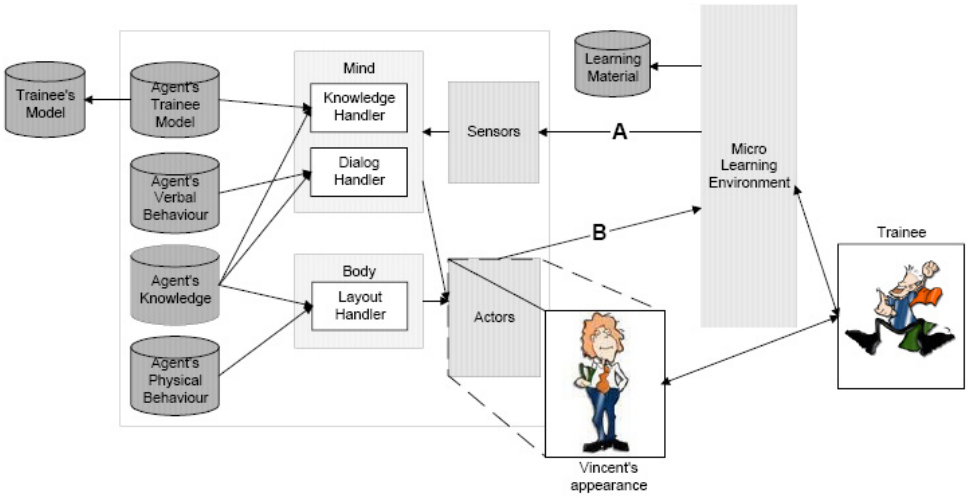

Figura 2.8: Arquitectura de Vincent

2.8.1.5. Doris

Doris es un agente pedagógico que actúa en un entorno web y cuyo objetivo es investigar los mecanismos, herramientas y métodos que se pueden utilizar para proporcionar una educación que saque partido a las características de la web (dos Santos et al., 2002).

Los estudiantes interactúan con el SIT a través de una interfaz web, y Doris monitoriza esta interacción y envía la información recogida a un sistema de razonamiento basado en casos.

Doris tiene la capacidad de percibir el entorno en el que habita y modificarlo, autonomía, capacidad de interactuar con el estudiante, capacidad de adaptarse a cambios en el entorno (i.e. estrategia de tutoría), movilidad (entre la máquina del estudiante y el servidor), y conocimiento acerca de los estudiantes con los que trabaja.

Para dar soporte a todas estas características, la arquitectura de Doris consta de tres módulos más una base de conocimiento.

El módulo de percepción tiene la responsabilidad de monitorizar la interacción del estudiante con el sistema, obteniendo información acerca de: inicio y finalización de la sesión, páginas visitadas, tiempo empleado en cada una de ellas, opiniones sobre el material empleado, preferencia por elementos a ser utilizados en futuras clases y preguntas realizadas.

El módulo cognitivo está encargado de realizar inferencias en la base de conocimiento, a partir de lo percibido por el módulo anterior, con el fin de determinar las acciones a realizar: mensajes que hay que enviar a los

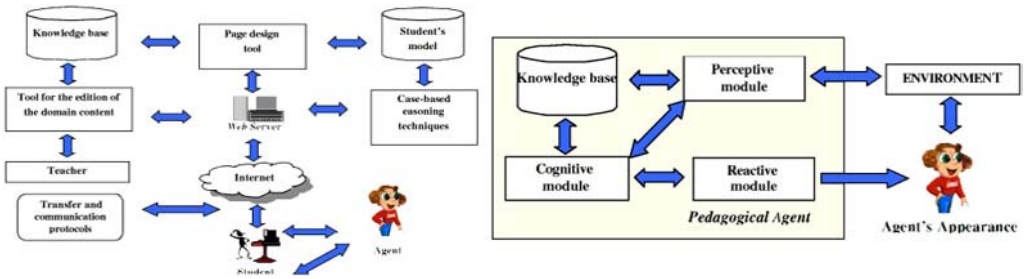

Figura 2.9: Arquitectura de Doris y su entorno

estudiantes, preguntas que hay que hacerle, respuestas a sus preguntas o animaciones que tiene que mostrar el agente.

El módulo reactivo es el responsable de ejecutar las acciones seleccionadas por el módulo cognitivo.

La base de conocimiento contiene animaciones y sonidos de Doris, información acerca de las clases, páginas visitadas, respuestas a las preguntas de los estudiantes, mensajes (i.e preguntas, sugerencias), preguntas realizadas por los estudiantes y preferencias de los estudiantes.

Doris presenta dos tipos de comportamiento: cognitivo y reactivo. El cognitivo es el encargado de enviar mensajes al estudiante, y de almacenar información acerca de él en la base de conocimiento. El reactivo, a su vez, es el encargado de las acciones que manipulan la apariencia de Doris, seleccionando una apariencia en función de la situación actual.

2.8.1.6. FILIP

FILIP es una plataforma multiagente para el desarrollo de sistemas de aprendizaje inteligentes basados en simulaciones (que no en EVs), orientado a la adquisición y desarrollo de habilidades por parte de distinto tipo de operadores (Zhang et al., 1998; Alem et al., 1997).

La arquitectura de FILIP se compone de siete agentes: cinco de ellos se corresponden con los habituales de un SIT, mientras que los otros dos, a tenor de las explicaciones de los autores, no parecen ser más que dos módulos de comunicación.

El agente de estudiante tiene la responsabilidad de recopilar las acciones del estudiante, relacionarlas con el conocimiento experto y construir un modelo de estudiante.

El agente de desarrollo de habilidades aconseja al instructor acerca de las habilidades del estudiante, de forma que se puedan tener en cuenta a la hora de planificar la instrucción. Este agente recibe como entrada el modelo del

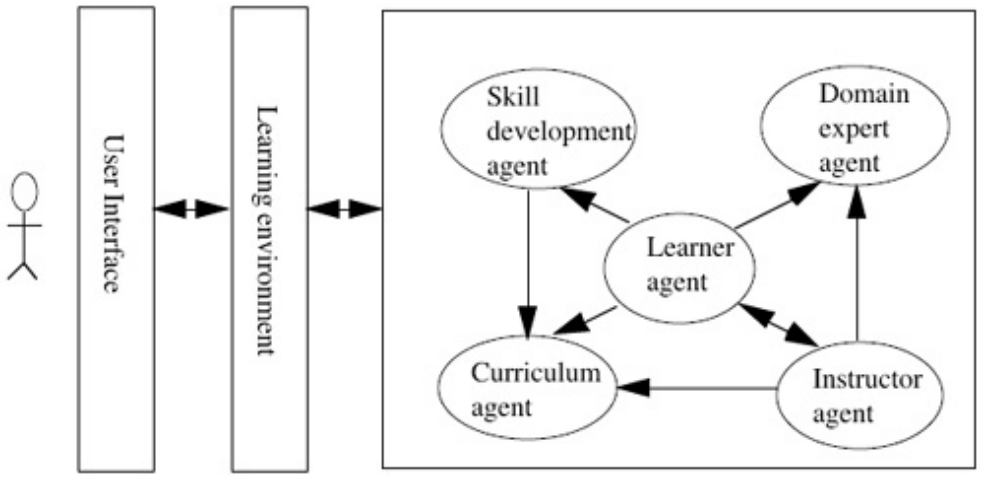

Figura 2.10: Arquitectura de Filip

estudiante realizado por el agente de estudiante, concretamente las medidas de rendimiento, y en base a ellas aconseja al agente instructor que aumente o no la dificultad del entrenamiento.

El agente experto posee conocimiento de las estrategias y soluciones para resolver las situaciones planteadas al estudiante. Esta información la usa el agente anterior para contrastar el rendimiento del estudiante.

El agente de curriculum proporciona información al agente de desarrollo de habilidades, para que pueda aconsejar al instructor, y al instructor para que pueda adaptar la dificultad del entrenamiento. Organiza la información siguiendo un Formalismo Curricular para el Entrenamiento de Habilidades Operativas (CFOST (Zhang y Alem, 1997)), que básicamente consiste en una organización del conocimiento en forma de árbol, pudiendo entrenarse en los nodos más altos a medida que se superan todos los que tiene por debajo.

El agente instructor adapta la instrucción a las habilidades del estudiante en función del modelo de estudiante y al contenido del CFOST. Esta adaptación se hace tanto de lo que se enseña como de su nivel de dificultad.

FILIP se ha utilizado para desarrollar un sistema llamado ATEEG (*Automation of Trainee Evaluation and Exercise Generation*) (Alem y Keeling, 1996), que se emplea para la instrucción de controladores aéreos. ATEEG utiliza un pizarra como medio para compartir información entre los distintos agentes, utilizando para ello un servidor de objetos que almacena los objetos compartidos por varios agentes.

2.8.1.7. DIT

DIT (*Distributed Intelligent Tutor*) es un tutor inteligente capaz de planificar una lección, generar ejercicios y plantear y explicar las soluciones a los ejercicios (Khoualdi y Benghezal, 2004). DIT está formado por un grupo de cinco agentes que se distribuyen la responsabilidad de la tutoría: el agente pedagógico, el generador de ejercicios, el resolutor de ejercicios, el explicador y el corrector.

El agente pedagógico realiza labores de coordinación y supervisión del resto de agentes. Además, también está encargado de la comunicación con el usuario. Para llevar a cabo el proceso de enseñanza cuenta con una librería de planes: unos están orientados a planificar el orden en que se enseña una materia, mientras que los otros están orientados a la manera en que se enseña un determinado concepto.

Según los autores, el uso de agentes en la construcción de un sistema de estas características hace que la resolución de un problema se pueda realizar en paralelo por todos ellos, de forma que los tiempos de respuesta son más cortos. Además, el sistema resultante es más modular, eficiente, flexible, fiable y reutilizable, aunque no aportan evidencia de ninguna de estas características.

2.8.1.8. Baghera

Baghera es un proyecto desarrollado para estudiar el comportamiento emergente de un sistema basado en agentes aplicado a la enseñanza de geometría por medio de una interfaz web (Webber y Pesty, 2002; Webber et al., 2002; Webber et al., 2001).

La arquitectura del sistema está estructurada en dos niveles, el primero de los cuales conforma la estructura de la aplicación, mientras que el segundo comprende agentes que se dedican a evaluar a los estudiantes. La aplicación está concebida como un sistema abierto, de manera que el número de agentes existente no es fijo, aunque sí lo es su tipo.

Así, cada estudiante cuenta con la asistencia de 3 agentes. El *agente personal de interfaz* tiene como principal objetivo monitorizar las acciones del alumno y comunicárselas a otros agentes. El *agente tutor* es el encargado de plantearle al alumno distintos problemas en función de los objetivos actuales, basándose en los conocimientos que posee el alumno. Supervisa a los agentes de segundo nivel cuando se hace necesario realizar alguna evaluación de los conocimientos del alumno, y cuando recibe el resultado planifica las acciones a realizar ayudado por los otros agentes de primer nivel. El *agente mediador*

tiene como responsabilidad elegir el resolutor de problemas más adecuado para enviarle la solución del alumno al problema planteado.

Por su parte, cada profesor cuenta con la asistencia de dos agentes. El *agente personal de interfaz* controla el acceso del profesor al material almacenado en el sistema. Se encarga de la comunicación con otros profesores y agentes, la creación de nuevas actividades para el estudiante y la supervisión del trabajo realizado por los estudiantes. El *agente asistente* ayuda al profesor en la creación de nuevas actividades.

Es en los agentes de segundo nivel, quienes evalúan los conocimientos del alumno, donde se intenta evaluar los comportamientos emergentes surgidos de la colaboración entre los agentes para realizar la mencionada evaluación.

Los agentes de segundo nivel pertenecen a una de entre cuatro posibles categorías: problemas, operadores, lenguaje y control. Estos agentes pueden encontrarse en estado activo y satisfecho o inactivo e insatisfecho. La función de estos agentes es comprobar si el elemento que representan está presente en el sistema. Si lo está, entonces pasan a estado satisfecho y pueden afectar el estado de satisfacción de otros agentes a través de un mecanismo de voto, del cual surgen los comportamientos emergentes.

El *agente problema* pasa a estado satisfecho cuando la categoría de problema que representa está presente en el sistema. El *agente operador* está satisfecho si el elemento al que representa está presente en la solución al ejercicio realizada por el estudiante. El *agente de lenguaje* está satisfecho si el elemento al que representa está presente en la solución del alumno. Este elemento puede ser una gramática, una representación gráfica o algún otro elemento que permita expresar la solución. Finalmente, el *agente de control* está satisfecho si, igual que en los anteriores, el elemento al que representa está presente en la solución.

Cuando el agente tutor pide un diagnóstico a los agentes de segundo nivel, les envía la solución del estudiante. Todos los agentes están en estado inactivo, y se van activando en función de las condiciones mencionadas anteriormente. Una vez están activos comienza una serie de rondas de votaciones para que cada agente, en función de lo que representa y de las votaciones de los otros agentes en rondas anteriores, vote si cree que el alumno conoce los conceptos que representa. El ciclo de votaciones finaliza cuando no se producen cambios significativos, y la evaluación queda decidida por el resultado de la votación. Según los autores, el mecanismo de votación está sin implementar, por lo que se desconoce la eficacia de su funcionamiento.

2.8.1.9. I-Help

I-Help es un sistema multiagente distribuido utilizado para que los alumnos de un curso puedan proporcionarse ayuda mutua (Vassileva et al., 1999; Vassileva et al., 2003). A diferencia de otros sistemas descritos en esta sección, I-Help no proporciona tutoría inteligente, sino que ayuda a que un estudiante encuentre la mejor ayuda posible de entre sus compañeros y otros recursos disponibles. A pesar de ello, su arquitectura basada en agentes y el particular modelo del estudiante que se realiza lo hacen igualmente interesante de cara al presente trabajo.

I-Help es una actualización de un sistema anterior llamado PHelpS (*Peer Help System*) (Collins et al., 1997), cuya arquitectura no estaba basada en agentes y era de tipo cliente/servidor, en lugar de distribuida. Este sistema se utilizaba para entrenamiento en lugares de trabajo.

La arquitectura de I-Help se basa en dos tipos de agentes: agentes personales y agentes de aplicación, que comparten una ontología y un lenguaje de comunicación comunes. Cada agente gestiona recursos específicos del usuario o la aplicación a la que representa, y negocian con ellos cuando necesitan recursos que no poseen.

Los agentes personales representan usuarios y mantienen modelos parciales del usuario, lo que incluye una lista de amigos y personas non-gratas o preferencias a la hora de negociar con agentes de otros usuarios. En definitiva, representan la manera en que un usuario quiere ser percibido por el resto de usuarios. Además, también construyen modelos de los usuarios con los que se relacionan, estableciendo relaciones duraderas en el tiempo que pueden ser utilizadas en el proceso de negociación para, por ejemplo, aumentar o disminuir el coste de lo que otro agente necesita.

Los agentes de aplicación representan a una aplicación en concreto, y construyen modelos de los estudiantes que han utilizado esa aplicación, como su grado de avance respecto al temario de una asignatura, lo que puede ser utilizado para que un agente de estudiante busque la ayuda de otro estudiante. Los agentes de diagnóstico representan un tipo especial de aplicación que crea un modelo de usuario para un área de conocimiento y actividad particulares, como por ejemplo un examen de una parte específica del temario.

Los agentes representantes gestionan colecciones de modelos de usuario. Su misión principal es trabajar con modelos de ciertas características de los usuarios y realizar emparejamientos con propósitos específicos, como localizar a los estudiantes con mayor conocimiento en un área para ayudar a otro con menos conocimientos. Estos agentes reciben modelos de los agentes de estudiante y de diagnóstico, que son los encargados de construir los modelos,

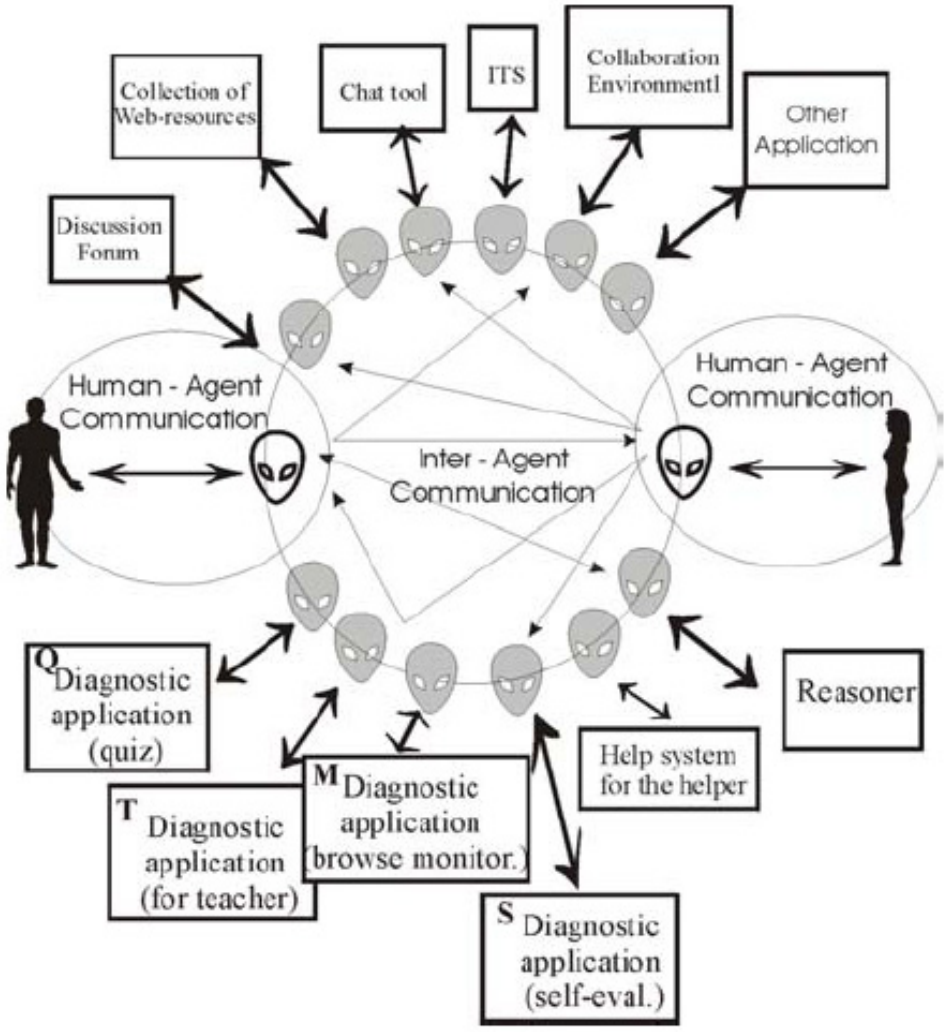

Figura 2.11: Arquitectura de I-Help

y los comentarios de los estudiantes respecto a la ayuda recibida por o proporcionada a otros estudiantes.

En una sesión típica, un usuario que necesita ayuda delega la responsabilidad de encontrarla en su agente de usuario. Éste puede intentar buscar ayuda en agentes de aplicación que representen, por ejemplo, un foro de discusión o una lista de preguntas frecuentes. También puede buscar ayuda en otros agentes de usuario. Ambos tipos de agentes se encontrarán por medio de un agente representante, que se encarga de hacer el emparejamiento del conocimiento que representa un agente y el que necesita el estudiante. En

el caso de buscar la ayuda de otros estudiantes, el agente representante elabora una lista de usuarios conectados que pueden ayudar, y el agente del estudiante comienza la negociación con ellos, intentando encontrar a uno que proporcione una buena ayuda a un precio razonable (en créditos I-Help, que luego se usan, por ejemplo, para subir la nota de los estudiantes). Una vez han llegado a un acuerdo, el agente del estudiante que va a proporcionar la ayuda le pregunta a su usuario si desea hacerlo, y en caso afirmativo se pone a los dos estudiantes en comunicación. Cuando uno de los estudiantes da por terminada la sesión de ayuda, el sistema solicita que cada estudiante evalúe al otro. Esta evaluación sirve para que cada agente de estudiante mantenga un perfil sobre el otro estudiante y para que el agente representante mantenga actualizados sus perfiles.

Para un grupo de novecientos usuarios, I-Help pone en ejecución a más de mil agentes que manejan alrededor de diez mil trozos de perfiles de estudiantes.

Existen varias razones por las que se decidió utilizar una arquitectura distribuida basada en agentes. En primer lugar, debido a la gran cantidad y heterogeneidad de las fuentes de ayuda, el uso de una arquitectura cliente/servidor apuntaba a que, finalmente, ésta se vería transformada en algo difícil de mantener y ampliar. Además, frente a lo que sucedía inicialmente con PHelpS, el número de interacciones simultáneas era bastante elevado, lo que producía colapsos en la aplicación. Finalmente, el uso de la mayor abstracción que aportan los agentes implica una mayor facilidad para diseñar la aplicación y los mecanismos de negociación.

2.8.1.10. Ines

INES (*Intelligent Nursing Education Software*) es un sistema basado en agentes desarrollado para ayudar en la formación de estudiantes de enfermería (Hospers et al., 2004). El sistema se ha desarrollado con el objetivo de que sea fácil de usar y que además pueda proporcionar demostraciones, explicaciones y realimentación a los estudiantes. Actualmente, la interfaz de la aplicación está basada en web, aunque parece que existía el propósito de conectar la aplicación con entornos 3D.

El sistema está construido, según los autores, basándose en dos estilos arquitectónicos. El primero de ellos, *abstracción de datos y organización orientada a objetos*, se utiliza para representar los objetos del entorno, como jeringas y agujas, y los agentes que conforman el sistema. El segundo estilo, *basado en eventos con invocación implícita*, se utiliza para la implementación de los agentes.

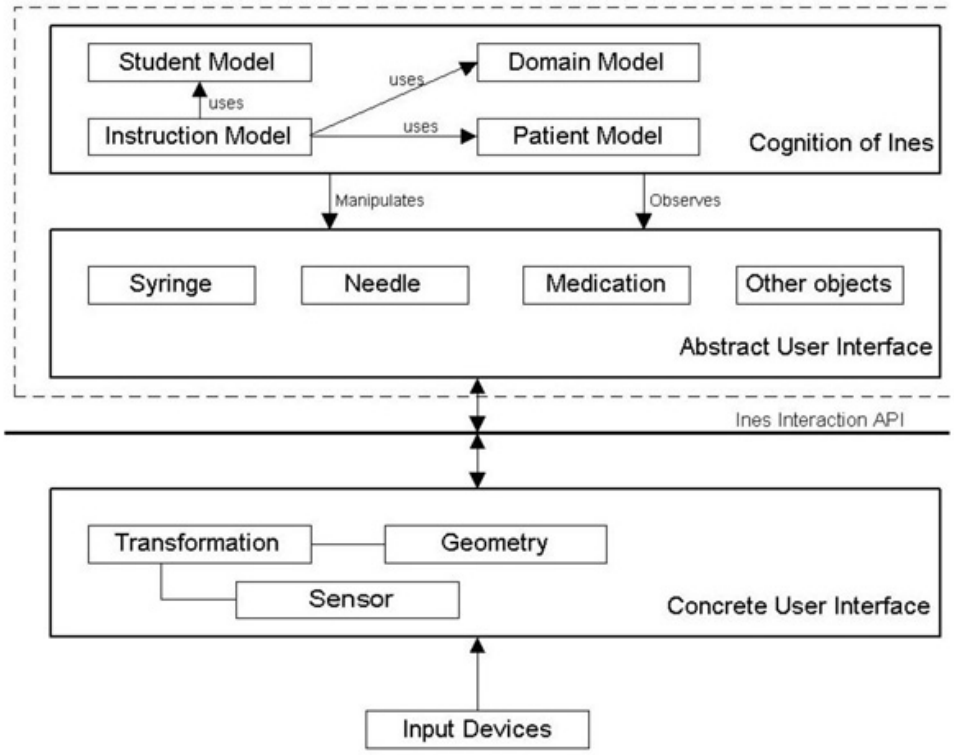

Figura 2.12: Arquitectura de INES

La arquitectura de INES consta de 4 capas. La primera de ellas controla los dispositivos de entrada. Al estar separado de INES propiamente dicho, la elección de los dispositivos de entrada sólo depende de la interfaz de usuario concreta que se utilice, pero no de lo que INES espere recibir desde el exterior. La siguiente capa, la interfaz de usuario concreta, se ocupa de la visualización de los ejercicios. La única restricción impuesta a la interfaz de usuario concreta es que comunique al sistema la información que necesita, como cuándo un estudiante coge un objeto, para poder así decidir si la acción es correcta o no. La tercera capa es la interfaz de usuario abstracta, una representación de alto nivel de lo que se muestra en la interfaz concreta. Esta tercera capa ya se considera parte de INES, y está constituida principalmente por métodos que permiten a la interfaz concreta comunicar al sistema las acciones del estudiante. La cuarta capa constituye el núcleo central de INES, y es donde se encuentran los agentes y donde se realiza el proceso de tutoría.

El núcleo de INES está constituido por tres modelos: uno de dominio, otro del estudiante y otro de instrucción.

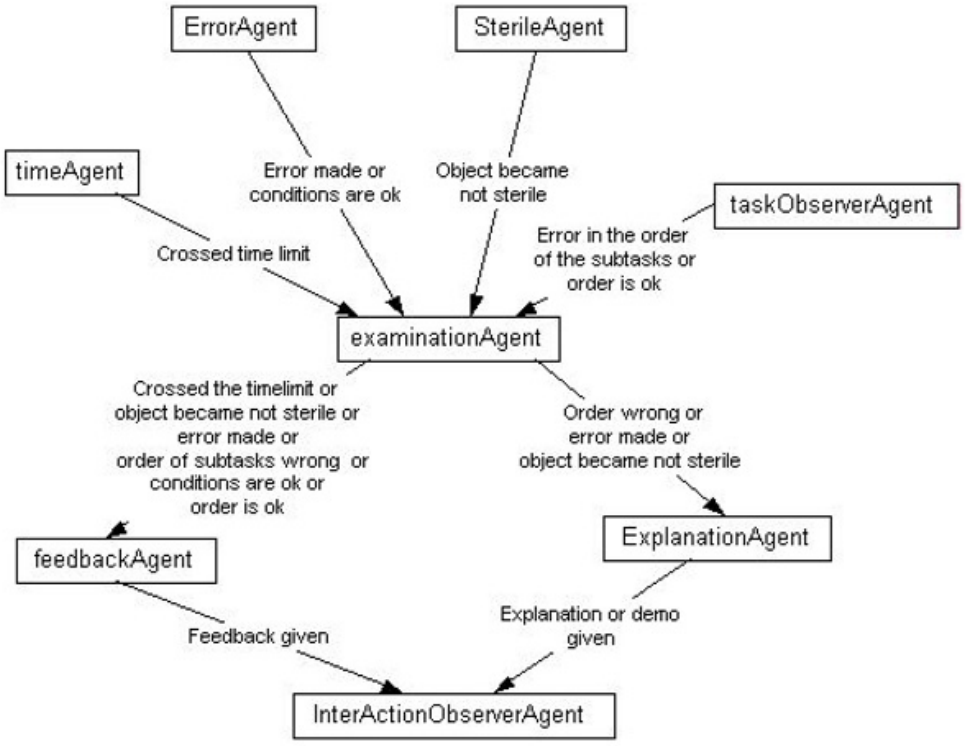

Figura 2.13: Arquitectura del sistema de agentes de INES

El modelo de dominio contiene el conocimiento acerca de los ejercicios que deben resolver los estudiantes, y se estructura a partir de la información contenida en ficheros XML. Estos ficheros contienen información acerca de tareas y subtareas a realizar, junto con sus precondiciones, postcondiciones y explicaciones respecto a las tareas y a la necesidad de que se cumplan las precondiciones. Dada la complejidad de estos ficheros han creado una herramienta, *Ines Exercise Editor*, para ayudar en su creación.

El modelo de estudiante contiene básicamente un histórico de los ejercicios que ha realizado el estudiante, junto con los resultados obtenidos en cada uno: si ha sido superado, con qué nota o cuántos intentos ha requerido.

El modelo de instrucción es el encargado de observar lo que hace el estudiante, compararlo con la información contenida en el modelo de dominio, rellenar la información contenida en el modelo del estudiante y darle a éste las explicaciones necesarias. Los agentes que conforman este modelo son agentes sensores, proactivos, y procesadores, reactivos. Cada agente sensor observa un aspecto concreto de la interfaz de usuario. Cuando se percibe algo nuevo, se le envía a un agente procesador, quien lo procesa y manda una respuesta.

Entre los agentes sensores se encuentran:

- Agente de tiempo: comprueba el tiempo que se tarda en resolver una tarea. Si se excede el tiempo permitido, manda un mensaje al agente de examen.
- Agente de error: cuando el estudiante realiza una acción, se registran las postcondiciones de la misma. El agente de error compara estas condiciones con las de la acción que debería haberse realizado. Si son las mismas no hace nada, pero si son distintas manda un mensaje al agente de examen con información acerca del error, que será usada posteriormente por el agente de explicación para indicarle al alumno la causa del error.
- Agente observador de tareas: está encargado de controlar el orden de realización de las subtareas. Sea correcto o no este orden, se manda un mensaje al agente de examen con el resultado de la comprobación.
- Agente estéril: este agente es específico del dominio de la enfermería. Su misión es comprobar que los utensilios utilizados sean estériles para no causar problemas al paciente. Cuando encuentra uno que no lo es, le manda un mensaje al agente de examen junto con información acerca de cómo dejó de ser estéril.

Por su parte, entre los procesadores están:

- Agente de realimentación: es el encargado de elaborar las respuestas que se le mandan al estudiante ante sus acciones en función de la información recibida del agente de examen. Se puede controlar, a través del valor de una variable, cada cuánto tiempo se le da esta información al estudiante, de manera que si es muy negativa no se deprima por las continuas reprimendas. Cada vez que se realiza una acción se envía un mensaje al agente observador de interacciones.
- Agente de explicación: es el encargado de decidir cuándo hay que dar una explicación o hacer una demostración, además de cuál es la explicación o demostración adecuada en cada caso. Cada vez que se realiza una acción se le envía un mensaje al agente observador de interacciones.
- Agente observador de interacciones: recibe mensajes de los agentes de realimentación y explicación, manteniendo una traza de ellos, lo que permite saber lo que el estudiante debería haber aprendido.

- Agente de examen: recibe mensajes de los agentes sensores y, en función del origen y del contenido del mensaje, se comunica con otros agentes procesadores para generar explicaciones, demostraciones o mensajes de realimentación. También es el encargado de mantener actualizado el modelo del estudiante.

Se han realizado diversas pruebas con un primer prototipo implementado de este sistema. Entre otros resultados, se ha podido ver que es relativamente fácil incorporar nuevos ejercicios sin realizar cambios. No sucede lo mismo si se quiere introducir un nuevo agente, siendo necesario modificar los agentes existentes.

2.8.1.11. GIA

GIA (*Generic Instructional Architecture*) es un proyecto cuyo objetivo es desarrollar una arquitectura y una serie de componentes software que proporcionen unas bases para el desarrollo evolutivo de SITs (Cheikes, 1995).

Los autores de este sistema ya identifican algunos de los problemas que motivan el desarrollo de la presente tesis doctoral. Uno es el hecho de que no se desarrollen sistemas completos, sino partes específicas de estos, las cuales son difíciles de integrar con otras, por no hablar de software comercial. Además, también destacan el diseño ad-hoc de cada SIT y la ausencia de diseños evolutivos.

La idea principal de GIA se basa en que un SIT se puede diseñar como una colección de agentes independientes que colaboran e intercambian información a través de un lenguaje. Además, los autores destacan que la clave de todo se encuentra en el lenguaje de comunicación.

La arquitectura de GIA es lo que los autores llaman una *arquitectura federada* (Genesereth, 1993) en la que existen unos agentes facilitadores que proporcionan servicios a otros agentes, facilitadores o no, formando una comunidad. Un servicio fundamental es el reenvío de mensajes en función de su contenido.

Los agentes que componen GIA proporcionan los servicios básicos de un SIT, lo que incluye un registro del estudiante (administrador de entrenamiento), planificación de lecciones (gestor de curriculum), administración de ejercicios (gestor de actividades), toma de decisiones de entrenamiento (planificador de entrenamiento) y modelado del conocimiento del estudiante (modelo de estudiante).

La interfaz de usuario está modelada como una serie de agentes de interfaz, cada uno de los cuales es responsable de gestionar los distintos recursos que utiliza el estudiante.

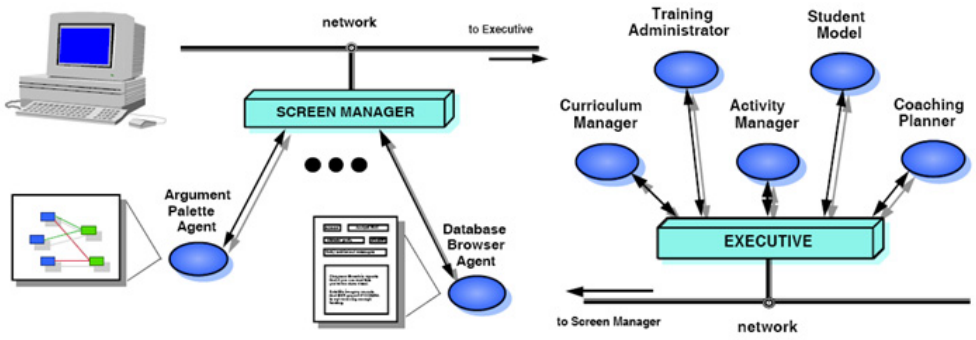

Figura 2.14: Arquitectura de GIA

El agente ejecutivo y el gestor de pantalla actúan como facilitadores de cada uno de los dos grupos de agentes descritos.

Según los autores, las cuestiones más importantes que quedan abiertas con este tipo de arquitectura es que hay que asegurarse de que existan agentes capaces de satisfacer las peticiones de otros agentes y hay que decidir qué hacer cuando más de un agente puede satisfacer una petición. Estas cuestiones, no resueltas cuando se desarrolló el sistema, han encontrado ya apoyo con el desarrollo del área de la ingeniería de software orientada a agentes. Queda aún sin resolver el impacto que sobre la reutilización tiene la ambigüedad de los mensajes.

2.8.1.12. Jonas

Jonas es un agente pedagógico que forma parte de un sistema llamado *Enxuto* desarrollado para entrenar a trabajadores de talleres en la creación de nuevos procesos de fabricación de productos (Baranauskas et al., 1997; Baranauskas y Borges, 1997; Borges y Baranauskas, 1997).

La filosofía de producción en la que se basa Enxuto es la conocida como *Kaizen* o *lean production*, según la cual la productividad se incrementa no sólo por el aumento en la cantidad de producción, sino también de la calidad de la misma. Para ello, es necesario que los trabajadores entiendan el proceso en el que están implicados, para así poder incorporar mejoras en el mismo. De esta manera, Enxuto pone a los trabajadores en el papel del encargado de la fábrica, y les permite construir sus propios modelos de fábrica y experimentar con ellos.

Enxuto se compone de tres módulos: uno de modelado, otro de simulación y un agente pedagógico.

El módulo de modelado contiene elementos para poder realizar el modelado, así como para monitorizar las acciones del alumno y proporcionarle realimentación. Contiene un editor de modelos en el que el alumno puede construir el modelo de fábrica, insertando módulos y estableciendo conexiones entre ellos.

El módulo de simulación utiliza el modelo dibujado en el editor para procesarlo y reflejar su funcionamiento. Una vez finalizada la simulación, el estudiante puede comprobar los resultados a través de estadísticas, gráficas o tablas.

En ocasiones puede suceder que la información proporcionada por el simulador no sea suficiente, por lo que el alumno puede interactuar con Jonás, quien puede realizar preguntas, proporcionar explicaciones o sugerir mejoras al modelo.

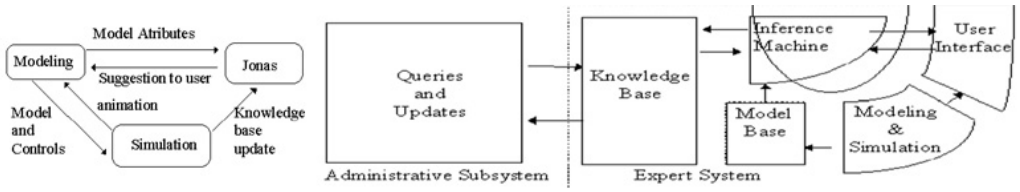

Figura 2.15: Arquitectura de Enxuto y de Jonas

Jonás está compuesto por un sistema experto, que consta de una base de conocimiento y una máquina de inferencia, una conexión con la interfaz de usuario, la base de datos de modelos y los resultados de la simulación, y un sistema de administración. La máquina de inferencia analiza el modelo base, comprueba los cambios introducidos por el estudiante y determina qué atributos han sufrido un cambio significativo tras la simulación. Después, usa la base de conocimiento para averiguar cómo mejorar los resultados. El sistema de administración está pensado para que un experto en el proceso de fabricación mantenga la base de conocimiento conforme a las necesidades de la planta.

2.8.1.13. ABITS

ABITS (*Agent Based Intelligent Tutoring System*) es un Sistema Inteligente de Tutoría que funciona en un entorno web y que es capaz de realizar modelado del estudiante y generación automática de temarios (Capuano et al., 2000).

Según los autores, existen tres tipos de sistemas de enseñanza basados en web: estáticos, donde un profesor cuelga el temario; personalizados, donde el

profesor puede monitorizar a los estudiantes y personalizar el temario para cada uno de ellos; y adaptativos, donde el profesor es sustituido por algún tipo de inteligencia artificial. ABITS es el módulo que permite transformar un sistema personalizado en uno adaptativo.

El conocimiento en ABITS está representado a través de *Objetos de Aprendizaje* (Hodgins, 2002) los cuales son utilizados también para realizar el modelado del estudiante. A través de los atributos de los objetos de aprendizaje utilizados se realiza un perfil de las preferencias del estudiante.

Además, se mantiene un modelo cognitivo del estudiante en función de los objetos de aprendizaje utilizados. Este modelo se realiza utilizando números borrosos. Si un estudiante utiliza un texto expositivo, el sistema interpreta que su conocimiento sobre ese concepto aumenta, pero con un bajo grado de certeza. Sin embargo, si el estudiante realiza un ejercicio, se puede tener mayor certeza del nivel de conocimiento del alumno.

Por otra parte, cada curso tiene una serie de metas a conseguir. En función de esas metas y de las preferencias de aprendizaje de un estudiante, el sistema puede diseñar un currículum personalizado, consistente en los objetos de aprendizaje necesarios para que el usuario adquiera los conocimientos necesarios para alcanzar las metas del curso.

Externamente, ABITS se comunica con el sistema de gestión del curso, al que le proporciona los objetos de aprendizaje que hay que presentarle al estudiante, y con una serie de bases de datos de donde obtiene información acerca de los usuarios y de los posibles objetos de aprendizaje que se pueden utilizar.

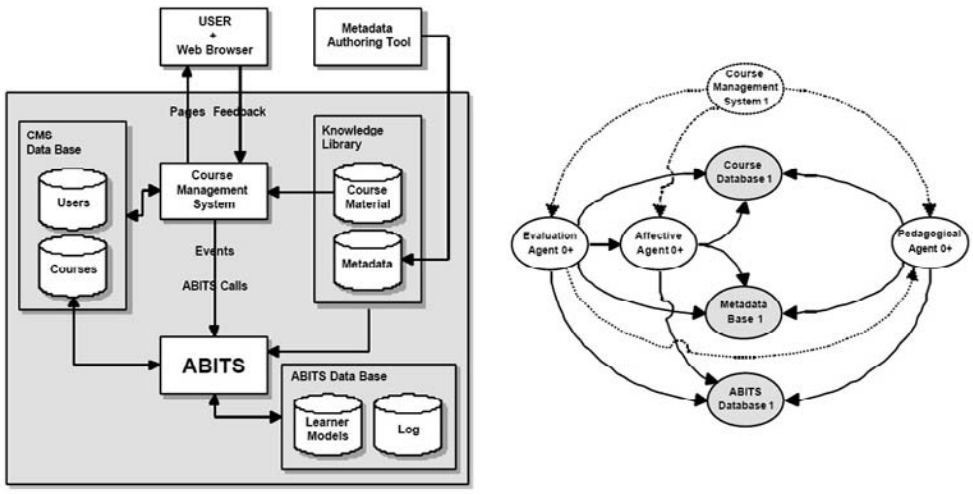

Figura 2.16: Arquitectura externa e interna de ABITS

Internamente, la arquitectura de ABITS se basa en la utilización de grupos de tres tipos de agentes: agentes de evaluación, responsables de evaluar a los estudiantes y de mantener el modelo de estudiante; agentes afectivos, responsables de actualizar las preferencias de aprendizaje de los estudiantes; y agentes pedagógicos, responsables de evaluar y actualizar el currículum diseñado para cada estudiante.

2.8.2. Entornos Virtuales de Enseñanza

2.8.2.1. GCW

El proyecto GCW (*Global Change World*) tiene como objetivo el estudio de la colaboración y el aprendizaje usando como herramienta un EV (Jackson y Fagan, 2000). Para ello, se ha diseñado una aplicación que permita aprender conceptos sobre calentamiento global en un EV que representa a la ciudad de Seattle.

Este proyecto ha sido desarrollado en el Centro de Aprendizaje del HITL (*Human Interface Technology Laboratory - University of Washington*), y ha sido financiado por la UW Royalty Research Fund.

GCW funciona sobre dos máquinas Hewlett-Packard 9000 que ejecutan el software de RV DVISE. La interfaz de usuario está formada por un casco de RV, un mando de navegación y control y un sensor de posición, todos ellos de Division Corporation, además de auriculares y micrófonos para permitir la comunicación entre los habitantes del EV.

La reproducción de la ciudad de Seattle se ha hecho de una manera selectiva, de forma que sólo se ha hecho un modelo del centro de la ciudad con los edificios más representativos, pero no se ha intentado hacer un modelo fiel de la zona, ya que el coste computacional se consideró excesivo.

Los estudiantes están representados en el sistema por un avatar que consta de ojos, orejas, boca y una mano que les permite manipular los objetos del EV. Para realizar tareas dentro del EV, los estudiantes acceden a una zona de control que les permite comprobar la temperatura media del aire durante un año, las partículas de gases en la atmósfera, las precipitaciones medias anuales y acceder a una máquina del tiempo que les permite ver el resultado de las acciones que llevan a cabo.

Para causar cambios en el medio ambiente, los estudiantes pueden ajustar tres grandes ruedas que aparecen en el EV y que representan: la cantidad de plantas que existen en el mundo, la cantidad de industrias que hay en el mundo y el impacto causado por la población mundial.

Se ha utilizado como sujetos de estudio a 56 estudiantes de 15 años y se les ha dividido en tres grupos para poder comparar distintas experiencias:

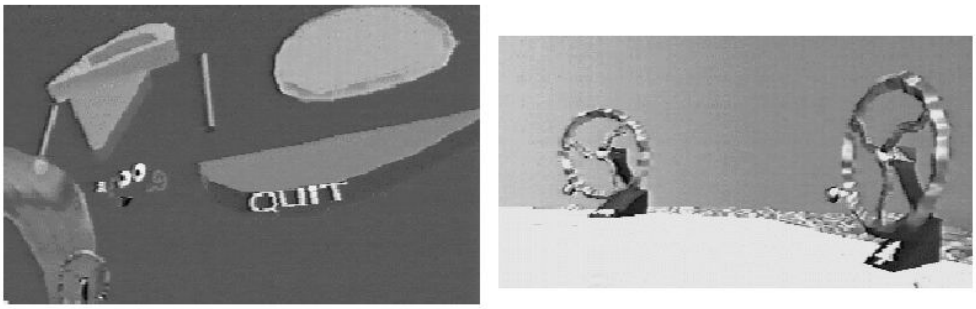

Figura 2.17: Dos vistas del sistema GCW

en un primer grupo, alumnos que no recibieron ningún tipo de ayuda; en otro, alumnos que trabajaban por parejas; y en un tercero, alumnos que trabajaban con un experto en el tema.

Como resultado del experimento, los individuos del primer grupo parecieron quedar algo menos satisfechos que el resto y obtuvieron peores resultados; los del segundo grupo obtuvieron mejores resultados y, a medida que avanzaba el experimento, se fueron sintiendo más cómodos tanto en el manejo del entorno como con la presencia de un compañero; los del tercer grupo fueron los que obtuvieron mejores resultados, si bien posteriormente fueron los que más importancia dieron a lo aprendido dentro del EV, aún por encima de lo que se les había explicado en clase, lo cual demuestra que se debe tener cuidado de no cometer errores en el diseño de los EVs, ya que esto puede acarrear problemas en el proceso de aprendizaje al poder influir mucho en los estudiantes.

2.8.2.2. NICE

El proyecto NICE (*Narrative, Immersive, Constructionist/Collaborative Environment*) es una iniciativa que pretende crear un EV que explote la colaboración como forma de motivación para el aprendizaje. Es una aplicación dirigida a niños cuyo objetivo es que creen un ecosistema y colaboren a través de la red para facilitar esta tarea. Además, se pretende que creen historias a partir de la colaboración que ha tenido lugar en el EV (Johnson et al., 1998a; Johnson et al., 1999a).

El EV que se ha creado es de tipo exploratorio, y los niños pueden hacer cosas tales como introducirse en un volcán, buscar peces alrededor de las isla que conforma el EV o cuidar el jardín que se ve en la Figura 2.18. En el jardín hay distintos tipos de semillas, y cuando siembran una de ellas, la planta

Figura 2.18: Un avatar en el jardín de NICE

correspondiente empieza a crecer. Para cuidarlas, tienen que asegurarse de que no están demasiado juntas y de que tienen suficiente agua y luz, y también pueden transformarse en ratones para poder ver las plantas más de cerca.

El jardín continúa su desarrollo sin la presencia de los niños, por lo que pueden desconectarse del sistema y volver más tarde para ver qué es lo que está sucediendo. Además, también es posible que los profesores se conecten al EV en forma de genios, lo que permite que puedan dirigir las actividades de los alumnos.

La aplicación consta de un jardín central al que puede conectarse un número ilimitado de clientes, los cuales pueden hacer uso de distintos tipos de interfaces. Cada usuario es representado por un avatar con cabeza, cuerpo y manos, y si los usuarios están utilizando sensores de movimiento, el sistema puede reproducir sus movimientos como gestos del avatar correspondiente. Además, los usuarios poseen micrófonos y altavoces, lo que permite que se comuniquen unos con otros de manera oral, especialmente si están en localizaciones cercanas.

El número de clientes que se pueden conectar a la aplicación viene marcado básicamente por el ancho de banda y la latencia de la red. La estructura de las conexiones es la que se puede ver en la Figura 2.19.

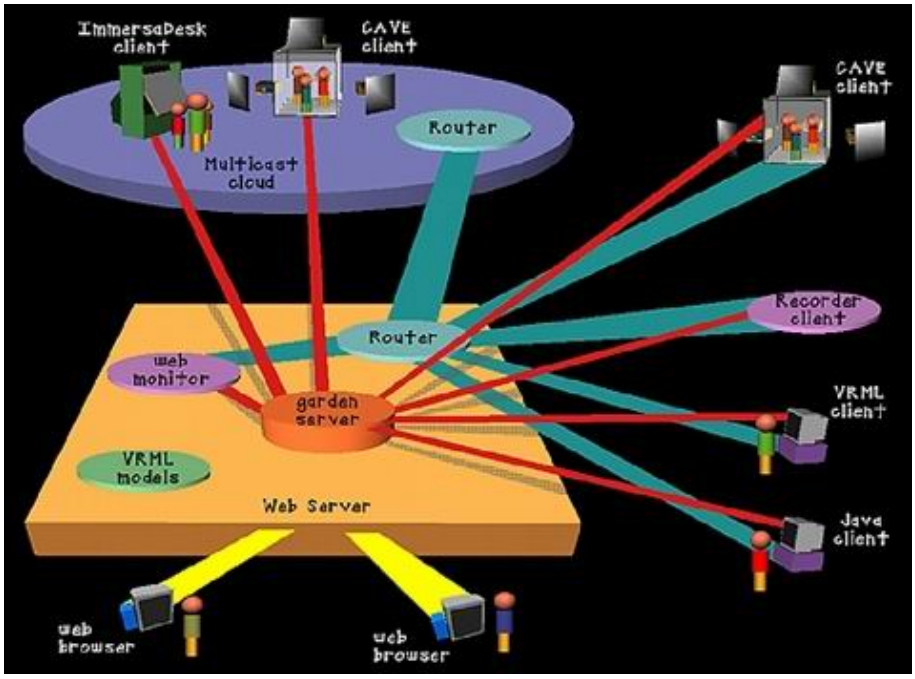

Figura 2.19: Conexiones a NICE a través de la red

Para transmitir la información del jardín que controla el crecimiento de las plantas se ha utilizado el protocolo TCP/IP, mientras que para transmitir la posición de los avatares se utiliza bien UDP, bien multicast. El segundo se utiliza dentro de subredes, mientras que el primero se prefiere para conectar clientes fuera de una subred.

A lo largo de distintas pruebas se ha podido constatar que los alumnos utilizaban el sistema con gran entusiasmo, no sólo por estar relacionado con la RV, sino por la posibilidad de poder interactuar con otras personas en otros lugares del mundo. Sin embargo, estos mismos estudios no pudieron determinar que el uso de la RV aportase ningún tipo de conocimiento real a los niños, ya que sólo un porcentaje muy bajo de los usuarios pareció aprender algo y ninguno de ellos pertenecía al grupo de usuarios con mayor nivel intelectual. Además, también quedó patente que cuanto más grandes eran los grupos de niños, menor era la colaboración, mayor el grado de competición entre ellos y menor el posible aprendizaje.

También se pudo determinar que la intervención de los profesores fue positiva, especialmente para favorecer la colaboración, aunque sólo en los grupos pequeños, ya que en los grupos grandes sólo sirvió para mantener el orden, de manera similar a como puede ocurrir en una clase.

De aquí se ha extraído la lección de que para resultar útil para el aprendizaje, la RV se debe usar con los siguientes criterios:

- La materia a aprender debe ser importante.
- La materia a aprender debe ser compleja, ya que la herramienta utilizada también lo es.
- La materia a aprender debe verse favorecida por el uso de la RV, es decir, que es más fácil que aporte algo para el aprendizaje de la estructura de las moléculas que para aprender a sumar.
- Las aplicaciones deben estar al día en cuanto a las materias a enseñar y la forma de enseñarlas, ya que si no corren el peligro de caer en el desuso y el olvido.

2.8.2.3. REP

El proyecto REP (*Round Earth Project*) persigue el estudio de cómo se puede usar la RV para enseñar conceptos que son contrarios a las concepciones que ya tiene el alumno (Johnson et al., 1999b). En concreto, se ha utilizado para enseñar a niños que la Tierra es redonda, cuando en la vida diaria todo parece apuntar a que es plana.

En este sistema se entiende por colaboración el hecho de que, al convivir dos individuos en un mismo entorno, se produce una conversación que aporta a cada individuo una visión ligeramente distinta a la propia, con lo que la percepción de cada individuo se enriquece con la aportación del otro.

Los autores de este proyecto son principalmente los mismos que en el proyecto NICE, del cual critican que el aprendizaje colaborativo no estaba estructurado ni dirigido, por lo que han dedicado sus esfuerzos a paliar estos errores.

Para que los niños pudiesen cambiar su concepción de que la Tierra es plana y entendiesen que es redonda, se creó un EV en el que dos niños pudiesen colaborar, de manera que uno pudiese concentrarse en observar mientras el otro hacía algo que le obligase a desplazarse por la Tierra y acabar apareciendo boca abajo para el otro niño. Además, se obligó a que la tarea de cada niño fuese necesaria para la del otro, forzando la comunicación y cooperación entre los dos.

En concreto, el experimento consistió en encontrar distintos objetos en la superficie del planeta. Uno de los niños era "El Astronauta", mientras que el otro era "El Controlador". El Astronauta se tenía que mover por el planeta cogiendo los objetos guiado por el Controlador, que tenía una visión

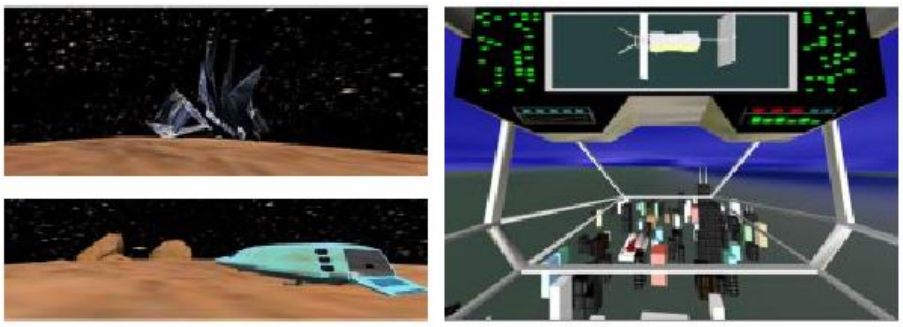

Figura 2.20: Vista del Astronauta y Vista del Controlador

esférica del planeta, mientras que el Astronauta lo veía plano, según se puede observar en la Figura 2.20. Después, los niños cambiaban de papel para tener la visión del otro.

El mismo experimento se repitió con tres grupos de niños. Con el primer grupo, se intentó hacer la aplicación fácil de usar. Con el segundo, se intentó ver qué hacían, en qué se concentraban y sobre qué hablaban. Con el tercero, tras de la experiencia de los anteriores, se intentó dirigir más el experimento, haciendo especial énfasis en que se fijasen en determinados aspectos de las tareas a realizar.

Como conclusión de estos tres experimentos se ha visto que los niños aceptan de muy buena gana el uso de la RV, sin mostrar demasiadas dificultades para su manejo, pero la ven más bien como un juego que como una forma de aprendizaje, más aún si puede participar más de un niño a la vez, de manera que la efectividad de su uso aún no ha podido ser probada.

2.8.2.4. CVE-VM

CVE-VM es una de las partes que integra el proyecto *Virtual Museum* (Wazlawick et al., 1999), cuyo objetivo es la creación de un EV colaborativo que ayude a la enseñanza en los colegios brasileños. Virtual Museum persigue el desarrollo de una serie de herramientas que hagan posible la creación de EVs de una forma colaborativa y según el enfoque construccionista, es decir, que mientras los alumnos crean el mundo de una forma colaborativa, también van absorbiendo los conocimientos relativos a la materia.

De esta forma, CVE-VM es un EV que funciona en Internet y que permite este tipo de aprendizaje colaborativo. No se persigue que el sistema haga las veces de tutor, papel que jugarán los profesores de los colegios, sino que constituya una herramienta que los profesores puedan utilizar para proponer

Figura 2.21: Interfaz de CVE-VM

actividades que deben realizar los alumnos y así ir adquiriendo conocimiento (Kirner et al., 2001).

Este proyecto está inspirado en las ideas puestas de manifiesto en el proyecto NICE, especialmente a la hora de adoptar el enfoque construccionista para construir EVs que sirvan al aprendizaje.

CVE-VM está desarrollado con Java y VRML, y la comunicación entre los dos se realiza a través de EAI (*External Authoring Interface*). Con VRML se proporciona toda la representación 3D, mientras que Java se encarga de manejar todos los eventos y mensajes que aparecen en el EV.

La aplicación utiliza un servidor cuyo principal objetivo es mantener la conexión con los usuarios y controlar la introducción de objetos en el EV. Por cada usuario que se conecta se crean dos tipos de objetos: uno que se encarga de gestionar la información relativa a la posición de los avatares y la inserción de objetos, y otro que se encarga de gestionar la comunicación entre los usuarios a través de un chat.

Una vez que los usuarios están conectados, pueden introducir objetos en el escenario seleccionándolos a partir de un catálogo de objetos que se pone a su disposición. Una vez insertado un objeto, es posible cambiar su posición y su orientación. Para ello, se debe seleccionar el objeto pinchando con el ratón sobre él, y en ese momento el usuario gana control sobre el objeto y nadie más que él lo puede manipular.

La comunicación entre los usuarios tiene lugar a través de un *chat*, con un aspecto similar al de los *chats* convencionales: un área para escribir los mensajes, otra para mostrar los mensajes de todos los usuarios y una lista con todos los usuarios conectados que ofrece la posibilidad de mantener conversaciones privadas con otros usuarios.

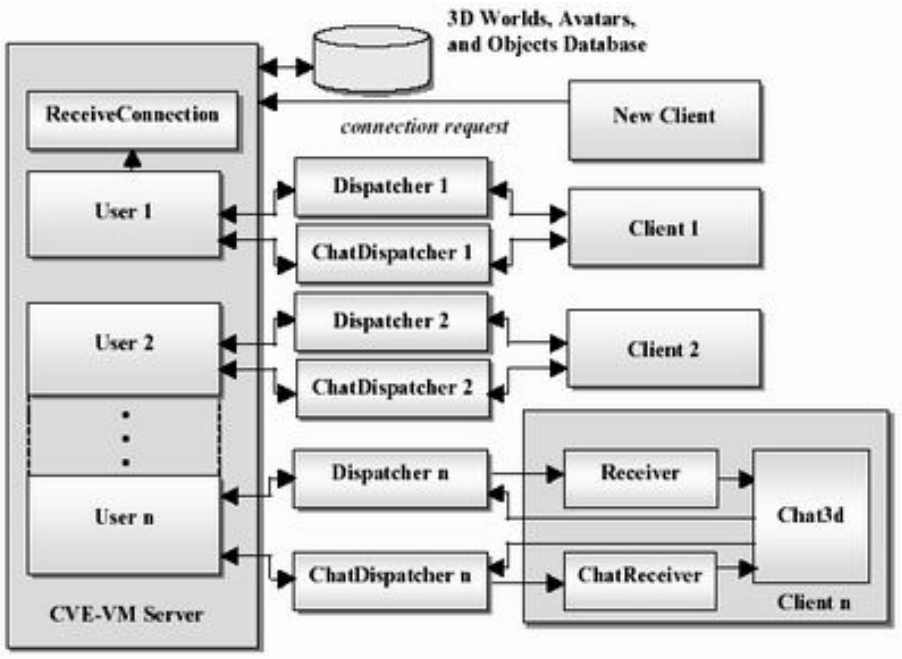

Figura 2.22: Arquitectura de Red de CVE-VM

2.8.2.5. Herman the Bug

Herman the Bug es un agente pedagógico desarrollado por el grupo dirigido por el profesor James Lester en Intellimedia (The Intellimedia Center for Intelligent Systems, North Carolina State University) (Lester et al., 1997c; Lester et al., 1999c). Herman es un insecto que hace las veces de tutor en Design-a-Plant, un EV diseñado para enseñar conceptos sobre anatomía y fisiología botánica. Para ello, Herman se introduce en una planta, observa las acciones de los estudiantes y les proporciona consejos para solucionar distintos problemas. El objetivo de los estudiantes es construir una planta capaz de sobrevivir en un entorno con unas condiciones dadas, como la cantidad de luz solar existente.

Figura 2.23: Herman the Bug en Design a Plant

El objetivo principal de este entorno es investigar la interacción entre agentes y estudiantes para resolver problemas (*mixed-initiative problem-solving*) en entornos de aprendizaje constructivista (Piaget, 1954).

Para ello, se ha dotado al agente de:

- Espacio de comportamientos: es una librería de gestos y movimientos que el agente puede realizar y de frases y sonidos que puede emitir.
- Modelo del contexto: es una base de conocimiento que contiene información sobre el problema a resolver, un historial de explicaciones y consejos que se le han dado al alumno y un modelo del artefacto que está siendo construido por el alumno.
- Motor de secuenciación de comportamientos: construye las acciones del agente utilizando el espacio de comportamientos en función del conocimiento contenido en el modelo del contexto.

Las habilidades de Herman no le permiten participar en interacciones complejas que requieran comunicación por turnos o análisis de textos, pero aún así, según los autores, ha servido para aprender importantes

lecciones acerca del desarrollo de agentes pedagógicos. Entre ellas, se encuentran: la necesidad de tener un amplio abanico de acciones para poder interactuar con el estudiante en distintas situaciones; diseñar una buena estrategia de intervención para no interrumpir al estudiante ni dejarle completamente solo, aunque ésta depende mucho del estudiante; necesidad de construir dinámicamente un modelo del progreso del estudiante; los consejos y explicaciones deben darse bien estructurados y con distintos niveles de abstracción; necesidad de herramientas que faciliten la construcción y escalabilidad de estas aplicaciones.

Este entorno ha servido para identificar el llamado *efecto persona* (Lester et al., 1997c), por el que se muestra una mejora del aprendizaje por el hecho de supervisarlo mediante un tutor animado.

2.8.2.6. Cosmo

Cosmo es un agente pedagógico animado desarrollado por el mismo grupo que Herman the Bug (Lester et al., 1997b; Lester et al., 1999a). El objetivo principal que se ha perseguido con el desarrollo de este agente pedagógico es el aumento de la credibilidad de este tipo de agentes.

Figura 2.24: Cosmo en Internet Advisor

Estudios realizados por el grupo de investigación que ha desarrollado Cosmo indican que los agentes pedagógicos pueden resultar eficaces en la enseñanza (Lester et al., 1997a), y la percepción de los estudiantes es que sirven de ayuda, son creíbles y entretenidos (Lester et al., 1997c).

Para que la sensación de credibilidad sea mayor, es necesario que los agentes pedagógicos aprovechen su conocimiento acerca de la posición de los objetos en el EV, su posición relativa a otros objetos o sus explicaciones previas para generar gestos, movimientos y explicaciones que resulten más naturales (Lester et al., 1999a).

Para ello, han creado un planificador de comportamiento que, tras la petición de una explicación o una pista, selecciona la explicación y construye una secuencia de acciones para proporcionársela al alumno. Este trabajo se basa en trabajos anteriores sobre animación y credibilidad de agentes pedagógicos (Stone y Lester, 1996; Lester y Stone, 1997).

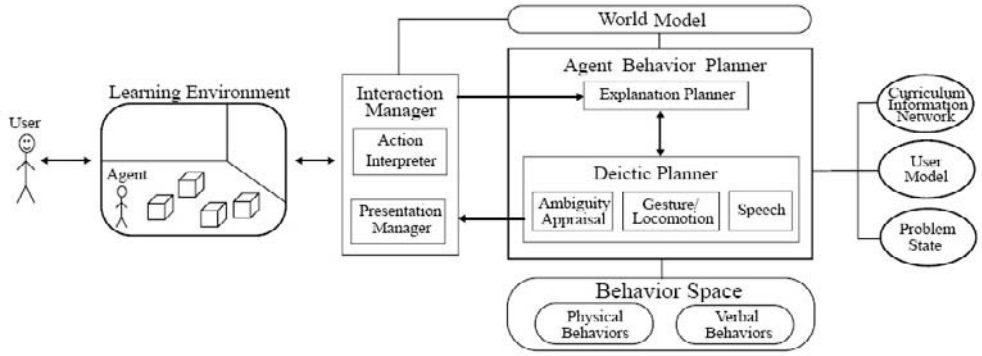

Figura 2.25: Arquitectura de Cosmo

Cosmo monitoriza las acciones de los estudiantes e interviene en dos situaciones: cuando el estudiante está mucho tiempo sin hacer nada o cuando se equivoca. En ese momento, el planificador de explicaciones selecciona la explicación que hay que proporcionarle al alumno y se la envía al planificador contextual (*deictic planner*). Éste, a su vez, consulta la información que posee acerca del mundo, el estudiante y sus acciones pasadas, decide los gestos y movimientos que deben realizarse, modifica la explicación cambiando demostrativos y pronombres y le entrega el resultado al planificador de comportamiento. Finalmente, éste se encarga de integrar los resultados de los dos planificadores en una serie de frases, gestos y movimientos que sirven para darle la explicación al alumno.

Este agente pedagógico se ha probado en un EV llamado *Internet Advisor*, en el que Cosmo enseña conceptos relacionados con redes IP: cómo se

construyen las direcciones, cómo se enrutan los paquetes, qué hace un router, etc., y la misión del estudiante es hacer que un paquete llegue desde un ordenador hasta su destino.

La representación de Cosmo en el *Internet Advisor* tiene una serie de partes móviles, como la cabeza, una antena, los ojos, los brazos y las manos, lo que le permite adoptar distintas expresiones y realizar distintos gestos, como coger objetos o señalarlos con el dedo.

La evaluación de este agente ha arrojado resultados positivos, ya que la mayor parte de los participantes en el estudio han destacado tanto los movimientos como las explicaciones ofrecidas por Cosmo, que incluyen justificaciones de las sugerencias y las posibles causas de los errores cometidos, lo cual ayuda al aprendizaje por parte del alumno.

2.8.2.7. WhizLow

WhizLow es otro agente pedagógico desarrollado en Intellimedia por los mismos autores de Cosmo y Herman the Bug. WhizLow habita la *CPU City*, un EV cuyo objetivo es servir para la enseñanza de arquitectura de computadores (Lester et al., 1999b).

Otros agentes pedagógicos observan al estudiante y le ofrecen explicaciones y consejos acerca de lo que está haciendo, pero en este caso el fun-

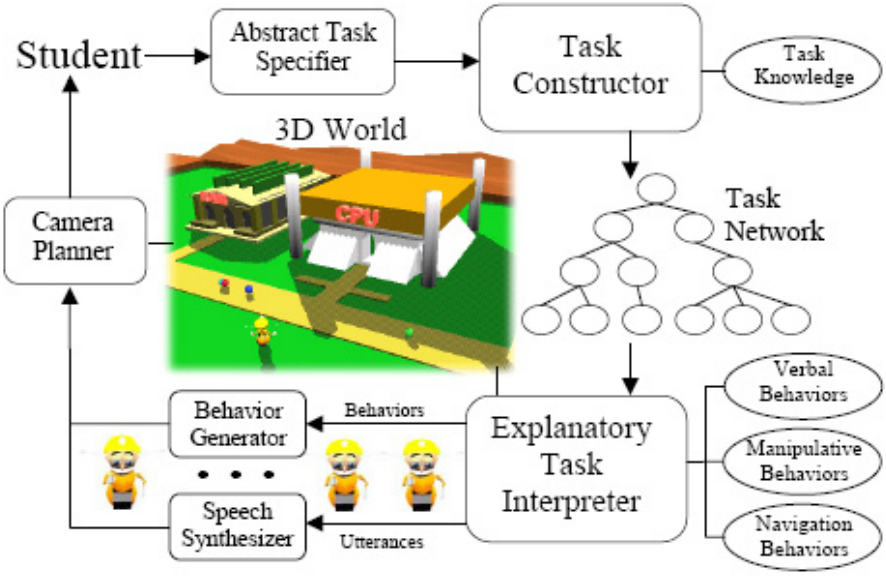

Figura 2.26: Arquitectura de WhizLow

cionamiento del agente es diferente. El alumno construye sus propias tareas utilizando una especificación de alto nivel y el agente se encarga de descomponerlas en un árbol de tareas de más bajo nivel, utilizando un planificador, y de llevarlas a cabo en el EV. Las hojas del árbol son las tareas que debe realizar el agente, pero están especificadas de forma independiente al EV utilizado, por lo que deberán ser interpretadas para que se puedan visualizar dentro del EV. Esta interpretación determina gestos, movimientos y frases que deben reproducirse en el EV.

Para evaluar la efectividad de WhizLow se han realizado cuatro experimentos, utilizando un EV sin agente pedagógico, otro con un agente sin presencia en el EV, otro con un agente que no ofrece explicaciones (sólo realiza las acciones indicadas por el estudiante) y otro con WhizLow completamente funcional. Los resultados parecen indicar que la ayuda proporcionada es bien recibida, tanto más cuanto menos experto es el alumno, y que el entorno que proporciona a WhizLow completamente funcional es el mejor valorado de todos.

2.8.3. Entornos Virtuales de Entrenamiento

2.8.3.1. STEVE

STEVE (*Soar Training Expert for Virtual Environments*) es un agente animado diseñado específicamente para ayudar a uno o varios estudiantes a llevar a cabo labores físicas que sigan un procedimiento determinado (Rickel y Johnson, 1999). Estas labores pueden requerir la intervención de más de una persona, y STEVE permite la participación de más de un estudiante, así como la sustitución de un humano por un agente que se encargue de realizar la labor de cualquier miembro del grupo de trabajo.

STEVE y el estudiante cohabitan en un escenario tridimensional que simula el entorno de trabajo donde el estudiante tendrá que realizar su labor, y STEVE puede demostrarle cómo realizar las tareas o puede supervisar la realización de las mismas por parte del estudiante, así como proporcionarle ayuda cuando éste lo requiera.

Este sistema incorpora características tales como la síntesis de voz, para dar las explicaciones pertinentes a los alumnos, el reconocimiento de voz, de manera que el alumno puede comunicarse con STEVE de manera oral, y el uso de cascos de RV, lo que posibilita, por una parte, que el alumno se pueda mover de una forma más realista por el EV, y por otra, que STEVE pueda saber fácilmente hacia dónde está mirando el alumno para detectar si está atendiendo a las explicaciones o si se encuentra distraído o incluso falto de motivación. Se ha tenido en cuenta la realización de un diseño modular, y

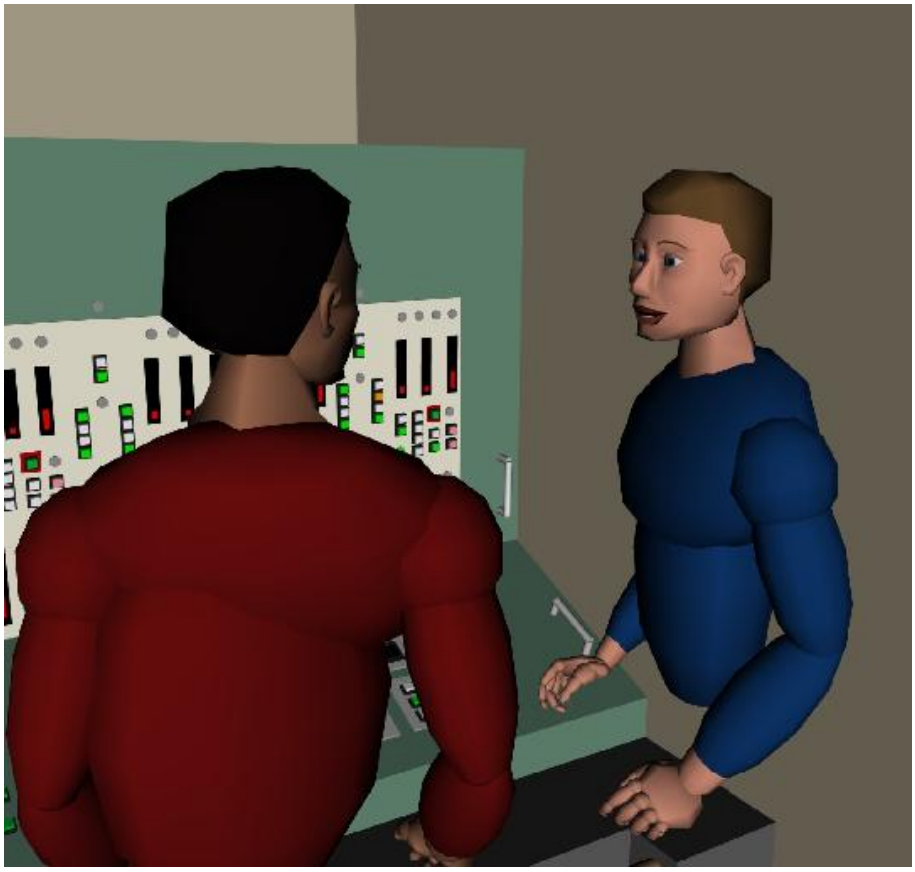

Figura 2.27: STEVE dando instrucciones a un alumno

la arquitectura de STEVE es lo suficientemente flexible como para permitir modificaciones, extensiones, etc., de una manera sencilla y rápida (Johnson et al., 1998b).

La arquitectura del EV, representada en la Figura 2.28, consta de una serie de componentes independientes que pueden ejecutarse como procesos separados o incluso en máquinas distintas, y se comunican entre ellos con paso de mensajes (Johnson et al., 1998b).

Los componentes de los que se hace uso son:

- Simulador: controla el comportamiento y la apariencia del EV.
- Interfaz Gráfica: cada usuario del EV utiliza un casco de RV, un guante de RV para tocar los objetos del EV y sensores de movimiento en el casco y en las manos para indicar su posición y sus movimientos. La interfaz se encarga de mostrar a través del casco de RV los cambios

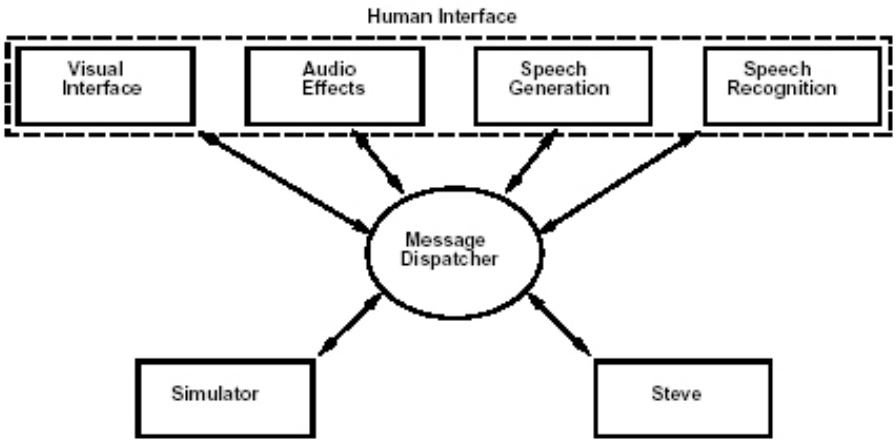

Figura 2.28: Arquitectura del Entorno Virtual de STEVE

que se producen en el EV, los cuales se reciben desde el simulador, e informa a otros componentes cuando el estudiante interactúa con objetos del EV.

- Interfaz de Sonido: recibe mensajes del simulador y transmite los sonidos pertinentes al estudiante.
- Síntesis de Voz: recibe mensajes de otros componentes, sobre todo de agentes, y los transforma en sonido para transmitírselo al alumno.
- Reconocimiento de Voz: Recibe mensajes a través del micrófono que utiliza el alumno, los reconoce y los transforma en *tokens* que se envían a otros componentes para que los utilicen (STEVE no posee ningún mecanismo de reconocimiento de lenguaje natural).
- Agente: cada agente se ejecuta como un componente separado.
- Centro de Mensajes: cada componente indica en el Centro de Mensajes qué tipos de mensajes desea recibir, y cuando llega un mensaje se envía a los componentes interesados. Esto aumenta la modularidad, al no tener que conectar los componentes entre sí, y la extensibilidad, ya que se pueden añadir componentes sin afectar a los existentes.

Como se puede ver en la Figura 2.29, STEVE se compone de tres módulos (Rickel y Johnson, 1999):

- Módulo de Percepción: monitoriza el estado del EV, mantiene una representación coherente del mismo y proporciona esta información a los otros dos módulos.

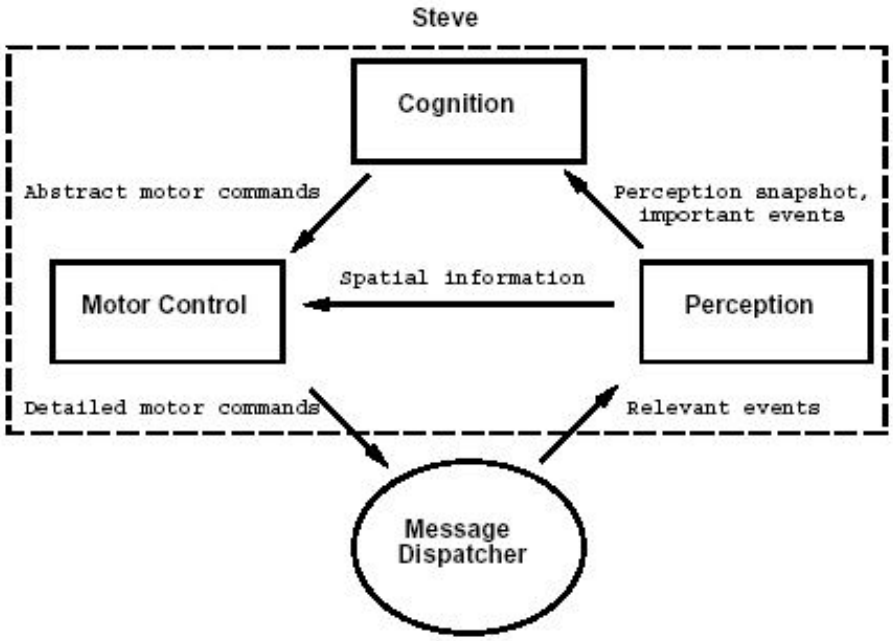

Figura 2.29: Arquitectura de los Agentes de STEVE

- Módulo Cognitivo: interpreta la entrada del Módulo de Percepción, elige las metas apropiadas, construye y ejecuta planes para conseguir estas metas y envía órdenes al Módulo de Acción.

- Módulo de Acción: controla la voz, movimientos, mirada y gestos de STEVE, y le permite manejar los objetos del EV.

Cada uno de estos módulos le proporciona a STEVE características y habilidades que son independientes del dominio, de forma que para capacitar a STEVE para trabajar en un nuevo dominio basta con especificar el conocimiento que se posee sobre él. Dicho conocimiento puede agruparse, a su vez, en dos categorías:

- Conocimiento Perceptivo: informa a STEVE sobre los distintos objetos existentes en el EV y sus características. Está asociado al Módulo de Percepción.

- Conocimiento Procedimental: informa a STEVE sobre los distintos procesos que pueden llevarse a cabo para realizar distintas tareas. Está asociado al Módulo Cognitivo.

STEVE empezó a desarrollarse como una aplicación orientada al entrenamiento de un solo estudiante. Sin embargo, todo el trabajo se realizó teniendo en mente futuras ampliaciones, como la inclusión de más de un alumno y la colaboración entre diversos miembros de un grupo para llevar a cabo la operación sobre la que se estuviese realizando el entrenamiento (Rickel y Johnson, 2002).

La arquitectura del EV que se ha mostrado anteriormente presenta una serie de ventajas de cara a realizar las ampliaciones mencionadas. Por una parte, el hecho de tener componentes independientes que se comunican a través de un centro de mensajes hace posible que los procesos puedan ejecutarse en diferentes máquinas, lo que constituye un primer paso hacia la posible colaboración entre individuos dispersos geográficamente. Por otro lado, también permite que cada usuario haga uso de sus propias interfaces, siempre y cuando respeten los mecanismos de comunicación utilizados por el sistema en su conjunto. Finalmente, es una aproximación fácilmente extensible, ya que el uso del centro de mensajes posibilita que pueda introducirse cualquier número de integrantes en el equipo, ya sean humanos o agentes, puesto que ningún miembro del mismo tiene por qué saber quién va a recibir sus mensajes.

Para poder trabajar en equipo, todos los miembros del mismo deben ser conscientes de lo que hace el resto, por lo que, por una parte, todos los integrantes humanos pueden ver lo que hacen el resto de miembros que estén a su alrededor, ya sean estos humanos o agentes, y todos los agentes deben ser informados de las labores que realizan el resto de los integrantes del equipo. La habilidad de STEVE para colaborar con humanos, ya sea como tutor o como miembro de un equipo, nace de la necesidad de "conocer" y "entender" las labores a realizar. STEVE debe saber siempre qué pasos hay que llevar a cabo, cómo contribuyen estos pasos al desarrollo de la tarea y quién debe llevarlos a cabo. Para adaptar a STEVE a las tareas de equipo, la única extensión que ha sido necesario hacer ha sido indicar qué miembro del equipo debe llevar a cabo cada paso dentro de la tarea. Esta nueva información será usada por todos los agentes existentes para construir un plan sobre cómo debe actuar el equipo en conjunto para realizar la tarea. Cada uno de los agentes que tenga un humano a su cargo adaptará el plan para su alumno en cuestión, teniendo en cuenta en todo momento lo que realicen el resto de los miembros del equipo.

Uno de los factores críticos del trabajo en equipo es la comunicación entre sus miembros, razón por la cual es fundamental que esa comunicación también pueda tener lugar durante el entrenamiento. En el caso de STEVE, es fundamental que los agentes sepan qué decir, cómo decirlo y cuándo decirlo, además de ser capaces de interpretar lo que dice cualquier otro miembro del

equipo y qué relación tiene con la tarea que se está llevando a cabo. Para poder dar soporte a esta comunicación, se han debido tener en cuenta dos aspectos (Rickel y Johnson, 2000):

- Comunicación verbal: ha sido necesario ampliar las acciones que puede realizar STEVE, incluyendo una nueva acción en la que se indica quién debe decir la frase, a quién se la dice, y qué efectos tiene, al igual que cualquier otra acción. Todos los agentes son capaces tanto de realizar como de entender este nuevo tipo de acción, y todos los alumnos pueden comunicarse con los agentes o con otros alumnos de esta manera. Además, todos los agentes con un alumno a su cargo deben ser capaces de entender las frases que otros habitantes del EV le digan a su alumno.

- Comunicación no verbal: se realiza utilizando las habilidades independientes del dominio de los agentes. Así, resultan fundamentales los gestos para señalar objetos, las miradas que indican dónde se desarrolla una acción o los gestos de aprobación o desaprobación ante la realización de una tarea por parte de un alumno.

La aproximación utilizada para llevar a cabo la comunicación sólo es válida para tareas estructuradas en las que la comunicación entre los miembros del equipo pueda ser especificada de manera concreta, pero STEVE no es capaz actualmente de trabajar con lenguaje natural.

Los resultados obtenidos con el desarrollo de STEVE pueden calificarse de bastante positivos. A pesar de no haber utilizado la tecnología punta de cada una de las áreas que abarca el proyecto, la conjunción de toda una serie de factores, como el uso de EVs que además manejan aspectos de colaboración, el uso de tutores inteligentes con capacidades de replanificación de manera dinámica o la inclusión de reconocimiento y síntesis de voz, hacen de STEVE un producto puntero dentro del área de la educación asistida por ordenador, y más concretamente de la RV aplicada a procesos de entrenamiento.

Los autores no disponen de datos objetivos que muestren la efectividad de STEVE para entrenamiento de un grupo de trabajo, pero sí se ha podido comprobar su eficacia en el entrenamiento de un sólo alumno. Además, el hecho de que se haya utilizado como base para otros proyectos, como se verá en la siguiente sección, da una idea acerca de los resultados que se espera obtener.

No obstante, STEVE adolece de una serie de inconvenientes que es necesario solventar para que llegue a constituir una aplicación que resulte de verdadera utilidad. Entre las limitaciones de STEVE se encuentran las que se enumeran a continuación:

- Necesita que las tareas estén bien estructuradas y se basen en procedimientos estándar. En este sentido, es de destacar que es necesario asignarle a cada miembro del grupo su papel correspondiente antes de iniciar el entrenamiento, y dicho papel no puede cambiar de manera dinámica, si bien esto no representa un serio obstáculo a la hora de realizar estas tareas.
- No admite un tutor para más de un alumno.
- Un agente no puede ser simultáneamente el tutor de un alumno desempeñando una labor y un miembro del grupo de trabajo que realiza otra labor distinta. Sin embargo, sí es posible que tanto humanos como agentes tengan varios roles dentro de un equipo, pero si un agente es el tutor de un humano tendrá los mismos roles que el humano que está a su cargo.
- El tutor no corrige las acciones del alumno, lo cual puede dificultar el proceso de aprendizaje si los errores de un alumno influyen en las tareas que debe realizar otro alumno.
- Los agentes nunca cometen errores, lo cual no ayuda a que los alumnos aprendan a reaccionar ante accidentes dentro de alguna de las tareas, salvo que los mismos alumnos provoquen estos accidentes.
- Los autores no entran en problemas tecnológicos inherentes a los Entornos Virtuales Distribuidos, como retardos de propagación o inconsistencias, ni a los EVCs, como el uso simultáneo de un mismo objeto.
- La colaboración que tiene lugar entre miembros de un equipo es bastante limitada. Sería deseable que varios miembros de un equipo pudiesen realizar labores de manera conjunta, como cargar un objeto entre dos o más individuos, ya que hasta ahora la colaboración se limita a un reparto de tareas.
- El reconocimiento y síntesis de voz se encuentran aún en un estado bastante elemental. Es necesario que se entre de lleno en el campo del reconocimiento de lenguaje natural para que un alumno pueda llegar a sentirse cómodo en un entorno de estas características.

2.8.3.2. MRE

El proyecto MRE (*Mission Rehearsal Exercise*) persigue la creación de un entorno de aprendizaje en el que los usuarios se encuentren inmersos en

un mar de imágenes, sonidos y circunstancias lo más parecidos al mundo real. Este proyecto se ha desarrollado en el ICT (*Institute for Creative Technologies - University of Southern California*) y se apoya en parte en el trabajo realizado para desarrollar STEVE.

Sin embargo, se trata de un proyecto mucho más ambicioso que cuenta con una importante financiación del ejército de los EEUU y cuyos participantes pertenecen tanto al mundo de la inteligencia artificial como al del diseño gráfico o el cine. La acción, como se ve en la Figura 2.30, se desarrolla en un pueblo de los Balcanes en el que el ejército se encuentra realizando una misión de paz.

Figura 2.30: Escenario principal del proyecto MRE

En este proyecto, y al contrario de lo que sucedía con STEVE, se realiza un tipo de entrenamiento para el cual no existe un objetivo claro ni una solución buena, sino que lo que se plantea es introducir a un individuo en una situación real para que resuelva un problema determinado de la mejor manera posible. Para ello, se ha intentado introducir el mayor número de elementos emocionales posibles, como expresividad en la cara y la voz de los personajes principales, humor cambiante en los personajes y una ambientación lo más realista posible, por ejemplo introduciendo personajes secundarios que van formando una multitud, sonido de helicópteros, bombas, gritos, etc. (Swartout et al., 2001).

La introducción en este proyecto de elementos y participantes provenientes del mundo del cine ha provocado que se hayan adoptado formas de trabajo utilizadas en la producción de películas. Así, lo más notable es la utilización de lo que han llamado "Desarrollo Híbrido", consistente en la diferenciación de elementos principales y secundarios para dedicar una cantidad de recursos proporcional a la importancia de cada elemento. Así, por ejemplo, tenemos

que para la creación de los distintos personajes que intervienen en el escenario se ha hecho la siguiente diferenciación:

- Programados: tienen un comportamiento predefinido que no varía en ningún caso y que se dispara según los distintos eventos que se produzcan.
- Inteligentes: interactúan con el alumno, y deben reaccionar ante situaciones imprevistas.
- Inteligentes con emociones: el comportamiento también se ve afectado por su estado emocional, por lo que su reacción ante una situación no siempre es la misma.

Así, se dedican más recursos y esfuerzo en el desarrollo y ejecución de los elementos más importantes, mientras que los elementos secundarios consumen menos tiempo de desarrollo, menos esfuerzo y menos CPU durante su ejecución.

Para diseñar la arquitectura del sistema se ha aprovechado la experiencia de STEVE para proponer un modelo similar al utilizado por éste, pero, como se ve en la Figura 2.31, con ligeras variaciones que se explican a continuación.

El proyecto MRE está pensado para ser utilizado en un escenario en el que el usuario se sitúa frente a una pantalla de 2'7 metros de alto y 9 metros de largo donde se proyectan las imágenes a través de tres proyectores que dan un ángulo de visión de 150º. Los gráficos están producidos por una SGI Onyx Reality Monster con 16 procesadores y 4 tarjetas gráficas.

El Sistema de Animación en Tiempo Real que se puede ver en la Figura 2.31 se ha conseguido integrando dos productos comerciales, Vega y PeopleShop. El primero de ellos se encarga de generar el entorno (edificios, caminos, árboles, vehículos, etc.) y los efectos especiales, mientras que el segundo se encarga de la animación de los personajes. Además, para aquellos personajes para los que se requiere una cara más expresiva, se ha usado VirtualFriend para el modelado de las cabezas, ya que este producto genera caras que pueden cambiar de expresión y que sincronizan el movimiento de los labios con las frases que pronuncian los personajes.

Para poder sincronizar el sonido con las imágenes y dar una sensación de sonido envolvente se ha utilizado un nuevo sistema multicanal que utiliza un total de 12 altavoces. Las señales que se envían por cada canal se procesan en un conjunto de filtros que simulan características acústicas del oído humano. Además, la sincronización de sonido e imagen también ha requerido mantener un control constante de lo que sucede en la escena y efectuar nuevas sincronizaciones de manera dinámica en caso de ser necesario. Esto ha sido

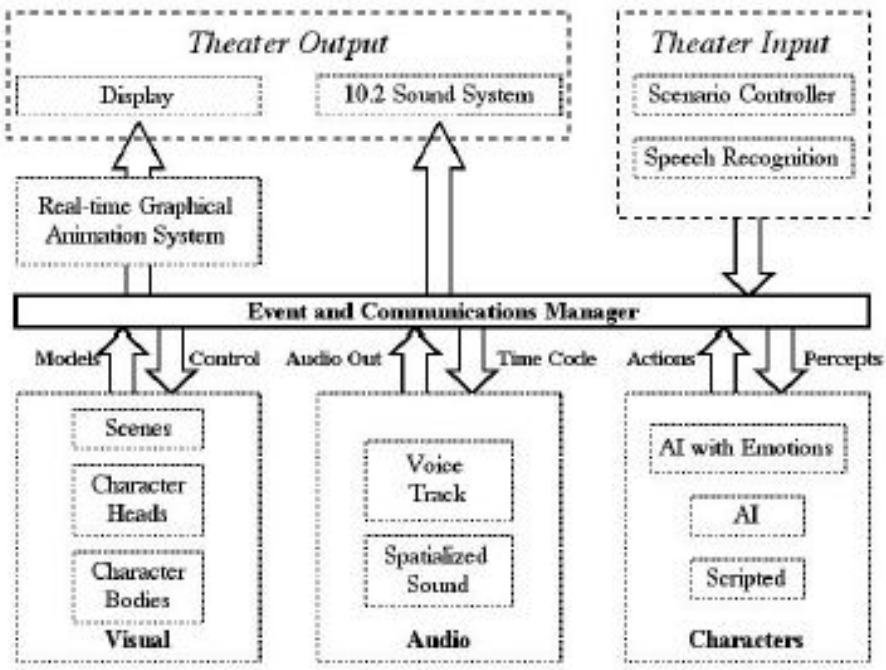

Figura 2.31: Arquitectura del proyecto MRE

especialmente necesario para mantener la sincronización en el habla de los personajes.

Como se ha dicho anteriormente, existen tres tipos de personajes: programados, inteligentes e inteligentes con emociones. Los personajes programados se han desarrollado usando PeopleShop. Los inteligentes, ya sean con o sin emociones, se han desarrollado apoyándose en STEVE, y sus capacidades se han aumentado con el uso de reconocimiento de lenguaje natural o la adaptación a una situación en la que las tareas pueden no tener una meta clara. A los personajes con emociones se les ha añadido una serie de características que afectan a su toma de decisiones dependiendo del estado emocional en que se encuentren.

En cuanto a la interacción del usuario con la aplicación, ésta se produce mediante un sistema de reconocimiento de lenguaje natural (Rickel et al., 2002b).

Es en la creación del contenido, la historia y la ambientación donde más se nota la influencia de la parte del equipo procedente del mundo del cine, y en concreto de Hollywood. Esto es debido a su experiencia en la creación de guiones que van más allá de una sucesión de eventos, lo cual hace que la inmersión dentro de la escena sea aún más profunda.

Sin embargo, en una aplicación como la que se describe esto no es suficiente, ya que lo que se persigue no es una acción lineal, sino que el usuario, mediante la interacción con otros personajes, pueda modificar la sucesión de eventos de muy diversas maneras. Tampoco es posible darle una libertad de movimiento completa, ya que de esa manera se puede llegar a perder por completo el carácter pedagógico de la aplicación, el cual es el fin fundamental que se persigue con su desarrollo.

Para poder adaptarse a estas dos necesidades, han desarrollado un método para crear la trama de la acción que recibe el nombre de *StoryNet* (ver Figura 2.32).

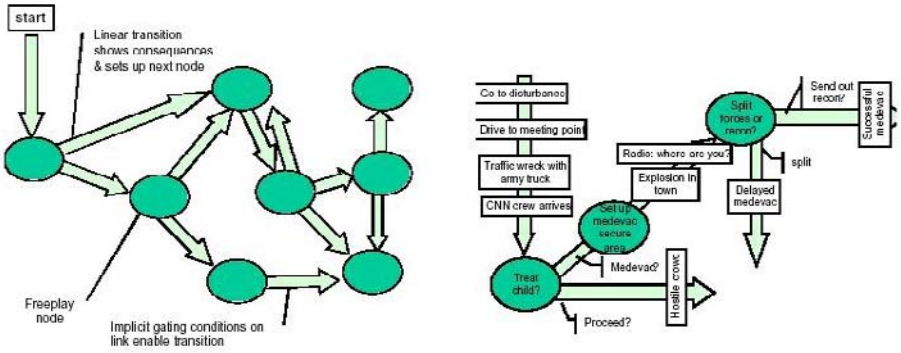

Figura 2.32: Representación del guión del escenario

Básicamente, lo que se hace es estructurar la historia en *nodos*, zonas de la historia en las que el usuario puede ejercitar su iniciativa, y *enlaces*, secuencias lineales de eventos que conectan dos nodos y sobre los cuales se concede muy poco control al usuario. Los nodos se corresponden con tareas bien definidas que el usuario debe aprender, o también con puntos en los que el usuario debe tomar una decisión que afecta al desarrollo de la historia. Puesto que los nodos tienen un alcance relativamente limitado, ésta es la manera de limitar el número de posibles acciones que puede llevar a cabo el usuario. Los enlaces, en general, se usan a modo de secuencias que unen la acción que tiene lugar entre dos nodos. Sin embargo, antes de que se pueda atravesar un enlace es necesario que se cumplan unas ciertas condiciones que se alcanzan mediante las acciones que realiza el usuario, si bien éste nunca es consciente de cuáles son esas condiciones para que no influyan en las decisiones que tiene que tomar.

El proceso de desarrollo, según afirman los autores, ha proporcionado una serie de experiencias que conviene no perder de vista:

- Gran parte del esfuerzo inicial se ha tenido que emplear para integrar las distintas aplicaciones que se han utilizado, en gran medida porque al ser herramientas de última generación es un trabajo que no se había hecho nunca antes. Esto ha requerido un gran equipo de trabajo y ha apuntado la necesidad de planificar y sistematizar el proceso de integración.

- La necesidad de trabajar con un equipo multidisciplinar ha acentuado la necesidad de coordinar a los distintos equipos de trabajo, ya que en muchos casos ha quedado patente que no se había tenido en cuenta la dependencia entre lo que producía un grupo y lo que necesitaba otro (como las expresiones faciales, necesarias para probar el modelo de emociones de los personajes).

- La imaginación desbordante de los guiones tuvo que ser en gran medida limitada por restricciones tecnológicas.

- El desarrollo del software se realizó de una manera bastante informal, lo que causó enormes dificultades a la hora de realizar la integración por la falta de un control de versiones de los distintos componentes que se iban desarrollando. Esto provocó que se produjesen aparentes fallos debidos simplemente a la utilización de una versión incorrecta de alguna parte de la aplicación.

- Unas pruebas preliminares muestran que el producto puede tener una buena acogida y unos buenos resultados pedagógicos de cara a su utilización por parte del ejército para entrenar a los oficiales que desplazan a realizar misiones de paz en distintas partes del mundo.

2.8.3.3. TLCTS

Tactical Language and Culture Training System (TLCTS) es un proyecto de la Universidad de Sur de California (USC) financiado por DARPA que tiene como objetivo la enseñanza de lenguas autóctonas a militares que participan en misiones fuera de su país (Johnson, 2007; Johnson y Valente, 2008). Posteriormente a su finalización, el proyecto continúa su desarrollo comercial a cargo de Alelo, una compañía fundada con este objetivo por algunos de los participantes en el proyecto.

Parte de los problemas con los que se encuentran los militares que participan en misiones fuera de su país tiene su origen en el desconocimiento de la lengua y las costumbres del país en el que se encuentran. Esto ocasiona

Figura 2.33: Entrenando una misión en TLCTS

enfrentamientos con la población a causa de malentendidos, además de dificultar la labor de los militares por la desconfianza de la población.

El objetivo principal de la aplicación es que los usuarios adquieran unas habilidades básicas fluidas en el menor tiempo posible, para poder interactuar con la población y llevar a cabo las misiones asignadas con el menor número de contratiempos.

Para ello, la aplicación se ha estructurado en tres tipos de actividades, una en la que los estudiantes pueden adquirir nuevas habilidades y otras dos en las que pueden ponerlas en práctica.

La primera de ellas es el *Skill Builder*, en la que los usuarios aprenden poco a poco palabras y frases del nuevo idioma, así como las situaciones en las que se pueden y deben utilizar. Aquí, los usuarios tienen la oportunidad de practicar la pronunciación, y pueden realizar ejercicios que implicar hablar y entender el lenguaje que se estudia. Cada lección tiene como objetivo adquirir una serie de habilidades, ya sea de aprendizaje de vocabulario o de pronunciación, que tendrán que poner a prueba en las otras dos actividades.

La segunda actividad es el *Arcade*, que es un juego que consiste en desplazarse por un escenario siguiendo las instrucciones que da el ordenador en la lengua que se estudia. En determinados momentos aparecen enemigos de distintos colores a los que se puede matar empleando la palabra que designa el color del enemigo. Con este ejercicio se consigue que el estudiante adquiera fluidez en el uso de la lengua que estudia.

La tercera actividad es la *Misión*, un ejercicio que reproduce una misión que pueden tener que llevar a cabo cuando se encuentre desplegados en un país extranjero. En esta misión tienen que completar una tarea en un determinado escenario, lo que requiere que se relacionen con la población para conseguirlo. El alumno debe ir dando instrucciones de lo que quiere hacer, y debe conseguir poner a la población de su lado para que le ayuden a conseguir su objetivo. Para ello, debe hacer un uso correcto de los conocimientos adquiridos anteriormente, utilizando correctamente frases y pronunciación y exhibiendo un conocimiento de las costumbres locales.

En los escenarios de la misión aparecen personajes con los que se deberá interactuar para alcanzar los objetivos de la misma. Estos

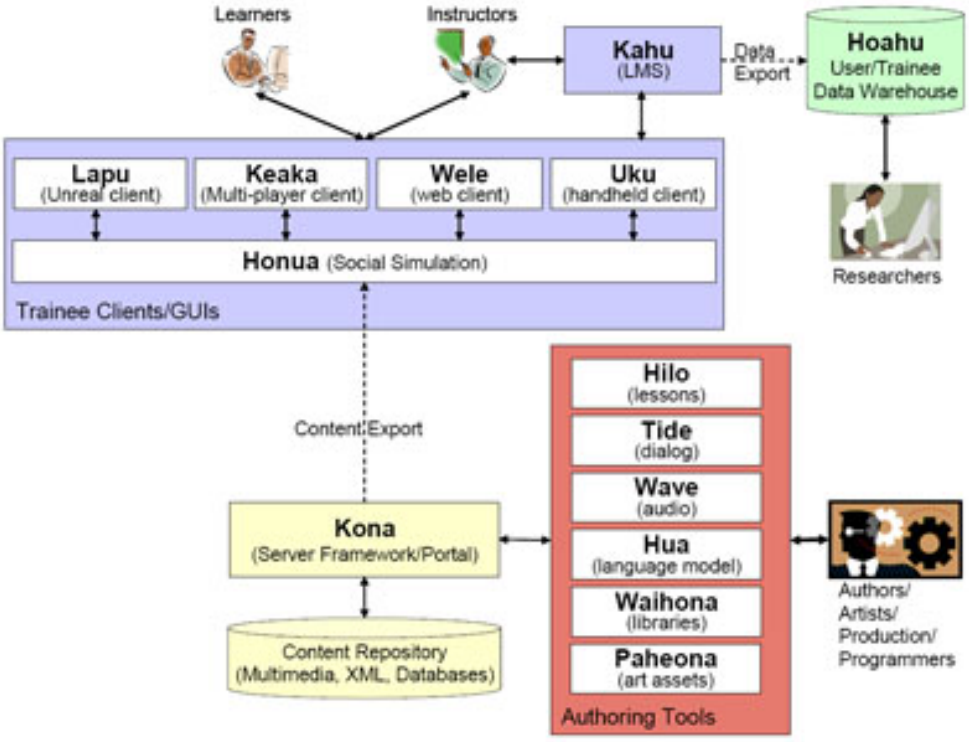

Figura 2.34: Arquitectura de TLCTS

personajes están controlados por agentes, pero su comportamiento, aunque relativamente complejo, está limitado a las posibilidades que ofrece el diagrama de estados que modela dicho comportamiento.

Los usuarios pueden alternar entre los tres tipos de actividades a voluntad, aunque tienen a su disposición una guía que les ayuda a utilizarlos en el orden que les puede proporcionar mayores beneficios.

Actualmente, la aplicación soporta distintos tipos de clientes, aunque algunos de ellos se encuentra en fase experimental. El único cliente que soporta la funcionalidad completa es el cliente monousuario en Windows, al que han dado el nombre de *Lapu*.

Este cliente está construido sobre el motor del juego *Unreal Tournament*. El estudiante se comunica con los personajes de la misión a través de un reconocedor de voz y de unos gestos que puede seleccionar en un menú de la pantalla. Un gestor de entradas interpreta tanto la voz como los gestos realizados y los transforma en actos comunicativos. En ese momento, el motor de simulación social determina la respuesta del entorno y de los personajes, y un planificador de acciones envía estas respuestas al motor gráfico para que las dibuje.

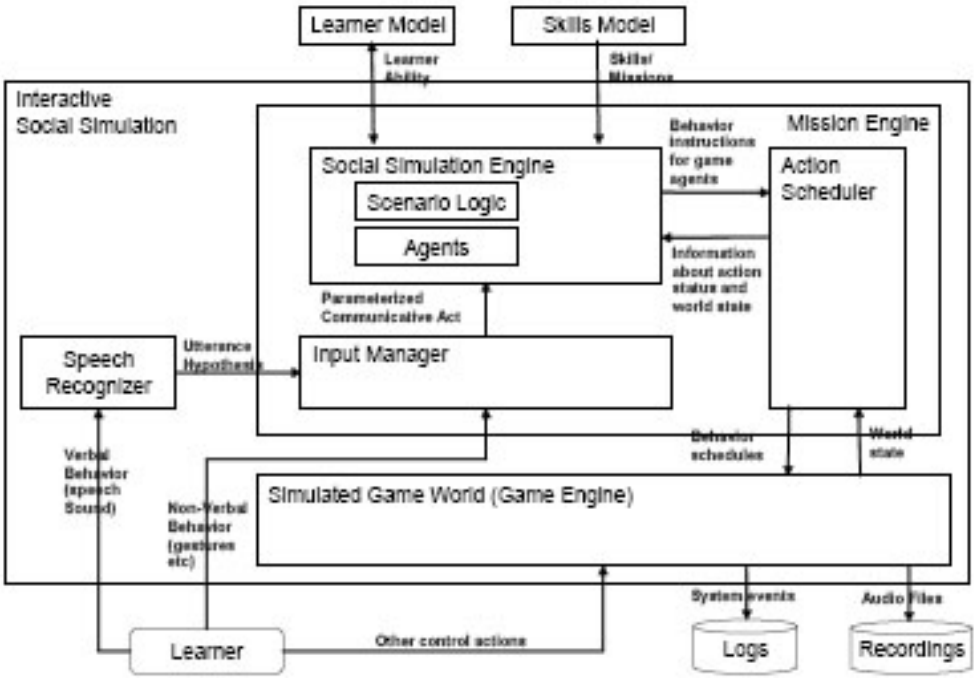

Figura 2.35: Arquitectura del cliente para PC de TLCTS

A medida que el estudiante interactúa con el sistema, se rellena un modelo del estudiante que realiza una estimación probabilística de los conocimientos del estudiante. Al mismo tiempo, se mantienen el log de acciones y se realizan las grabaciones de las frases del usuario para poder utilizarlas posteriormente.

Los clientes se conectan a un servidor web que recibe el nombre de *Kona*, donde se almacena todo el contenido de los cursos en formato xml. En este servidor, cada curso se representa como un libro compuesto de capítulos, que constituyen las diferentes lecciones del entrenamiento.

El sistema inicial era un proyecto de investigación que ha sufrido notables modificaciones hasta alcanzar la situación actual. En concreto, se han debido eliminar características que proporcionaban modificabilidad al sistema en aras de conseguir un mejor rendimiento, por lo que se han recortado algunas de las capacidades de los agentes. Esto, según los autores, también ha dotado al sistema de mayor estabilidad y facilidad de funcionamiento.

El sistema ha sido evaluado en diversas ocasiones con unos resultados bastante positivos. Además, los resultados de las evaluaciones se han utilizado para rediseñar algunas pates de la aplicación para hacerla más usable, rápida y efectiva. De hecho, la aplicación dispone de una funcionalidad para recopilar ficheros de log de las sesiones de entrenamiento, así como grabaciones de los usuarios hablando, lo que ha permitido, por un lado, adaptar el sistema a la forma de uso de los alumnos, y por otro, mejorar la tasa de aciertos del reconocedor de voz en diferentes situaciones.

2.8.3.4. Jacob

Jacob es un agente pedagógico con capacidad de procesamiento de lenguaje natural que se ha utilizado para enseñar a resolver el problema de las torres de Hanoi (Evers y Nijholt, 2000; Evers, 2000).

Los objetivos que se persiguen con el desarrollo de este agente son: interacción multimodal entre el usuario y Jacob; conseguir que Jacob ayude al usuario de manera proactiva y que aprenda de la interacción; animación de Jacob, incluyendo expresiones faciales y sincronización de los labios con el habla; manipulación de los objetos del EV tanto por parte del usuario como de Jacob; interacción agente-agente.

El sistema está construido siguiendo un enfoque orientado a objetos, poniendo especial énfasis en la mantenibilidad. Está estructurado en tres capas para, según los autores, separar las particularidades de la visualización 3D de la funcionalidad básica del sistema. La capa del mundo 3D concreto contiene información de la estructura del mundo en VRML. La capa del mundo 3D abstracto contiene información acerca de ejes, bloques y avatares. La tercera capa, relacionada con la funcionalidad del sistema, está, a su vez,

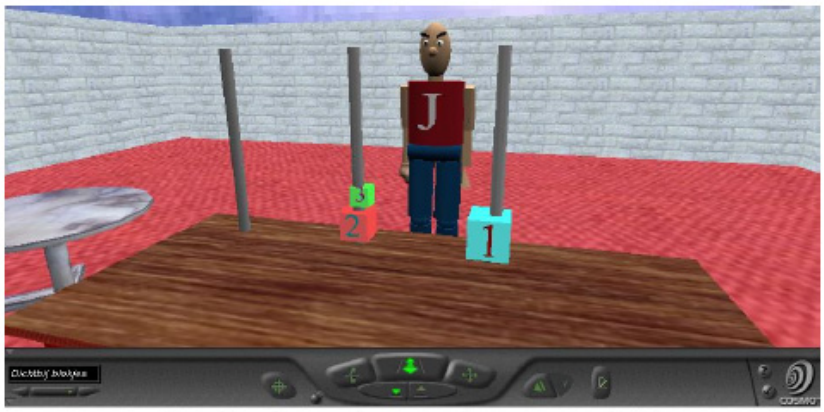

Figura 2.36: Jacob y las torres de Hanoi

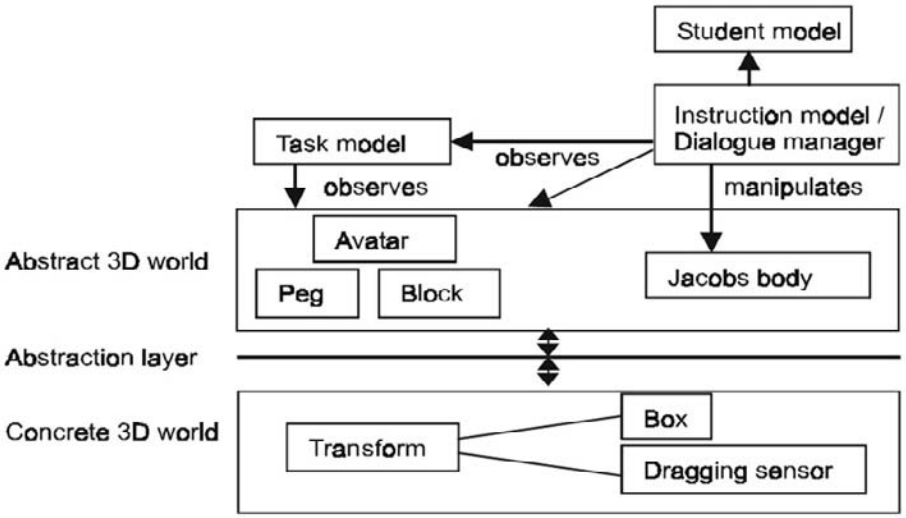

Figura 2.37: Arquitectura de Jacob

dividida en módulos. El modelo de tarea y el modelo de instrucción controlan el comportamiento de Jacob. Por un lado, el modelo de tarea contiene la información acerca del problema a resolver, en este caso las torres de Hanoi. Por otro, el modelo de instrucción contiene conocimiento pedagógico, como la forma de adaptar la tutoría al estudiante. También existe un modelo de estudiante, aunque no ha sido implementado.

Básicamente, la arquitectura descrita se corresponde con la de un SIT en el que tanto el módulo experto (modelo de tarea) como el de tutoría (modelo de instrucción) se comunican con el EV.

2.8.3.5. MASCARET

MASCARET (*Multi-Agent Systems to simulate Collaborative, Adaptive and Realistic Environments for Training*) es un sistema basado en agentes que integra un SIT para crear EVs de entrenamiento (Buche et al., 2004; Querrec et al., 2004). Este sistema se ha utilizado para desarrollar Sécurévi, entorno para entrenar a oficiales de bomberos en la dirección de operaciones.

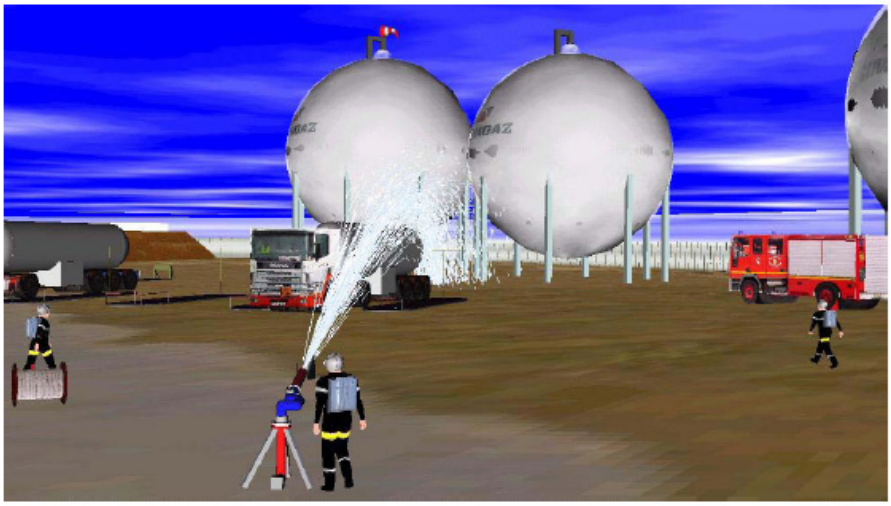

Figura 2.38: Operación para apagar un incendio en Sécurévi

MASCARET está diseñado en función de la pertenencia de los agentes a distintas organizaciones, y se basa en los conceptos de agente, organización, rol y comportamiento.

Aunque se pueden utilizar tantas organizaciones como se desee, para el desarrollo de Sécurévi se han utilizado las siguientes:

- Organización física: especifica las interacciones que tienen lugar en el entorno físico. El creador de la sesión de entrenamiento puede activar y desactivar estas interacciones, controlando así el nivel de dificultad de las actividades. Los agentes se organizan en una red de interacciones, en la que unos agentes funcionan como origen de las interacciones y otros como destino, dependiendo del rol de cada uno.

- Organización social: representa las interacciones que tienen lugar entre los miembros del equipo que realiza el entrenamiento, ya sean humanos o agentes. Todos los agentes de esta organización se conocen unos a

otros, y son conscientes de los roles que desempeña cada uno. Los procedimientos a seguir están predefinidos, pero para evitar situaciones inesperadas cada agente debe comprobar que se cumplen una serie de precondiciones antes de realizar una acción. En caso de no ser así, construye un plan local para resolver el problema y, si no es capaz de conseguirlo, recurre a su supervisor, quien a su vez recurre a todos sus subordinados para resolver el problema. Este ciclo se repite hasta que se encuentra una solución al problema y se puede continuar.

- Organización pedagógica: permite especificar lo que los alumnos deben hacer y lo que los agentes pedagógicos son capaces de realizar. Estos agentes pedagógicos pueden desempeñar distintos roles, como tutor, experto, compañero de equipo o fuente de errores. Cualquier agente puede ser parte de esta organización.

- Organización de mediación: establece de manera explícita la manera en que el alumno puede interactuar con el EV y hasta qué punto puede delegar tareas en su avatar.

- Organización humana: aunque no está completamente bajo el control del sistema, esta organización tiene en cuenta las interacciones entre humanos, intentando prevenirlas o propiciarlas (p.e. animando al estudiante a realizar preguntas a un tutor humano).

Para integrar un SIT dentro de MASCARET, los agentes pertenecientes a la organización pedagógica se estructuran de manera que cubren los cuatro módulos de un SIT clásico más un quinto de nueva creación.

El modelo de dominio contiene el conocimiento del procedimiento a realizar, así como de la manera de realizarlo en el EV, por lo que también implica a las organizaciones física y social. Existe un agente encargado de mantener el conocimiento del dominio, que es capaz de informar acerca de las precondiciones y metas de una acción, el orden de las acciones y la responsabilidad de los intervinientes. También maneja información de los distintos roles que intervienen en un fenómeno físico.

El modelo de errores, introducido en MASCARET, es un agente que maneja una base de conocimiento con errores típicos que cometen los estudiantes. Se utiliza para identificar los errores cometidos por los estudiantes y averiguar la causa de los mismos.

El modelo del estudiante está constituido por un agente que maneja información acerca de las acciones del estudiante. Este agente es el llamado agente avatar, que es el mismo que controla el usuario y en el que puede delegar algunas de sus acciones.

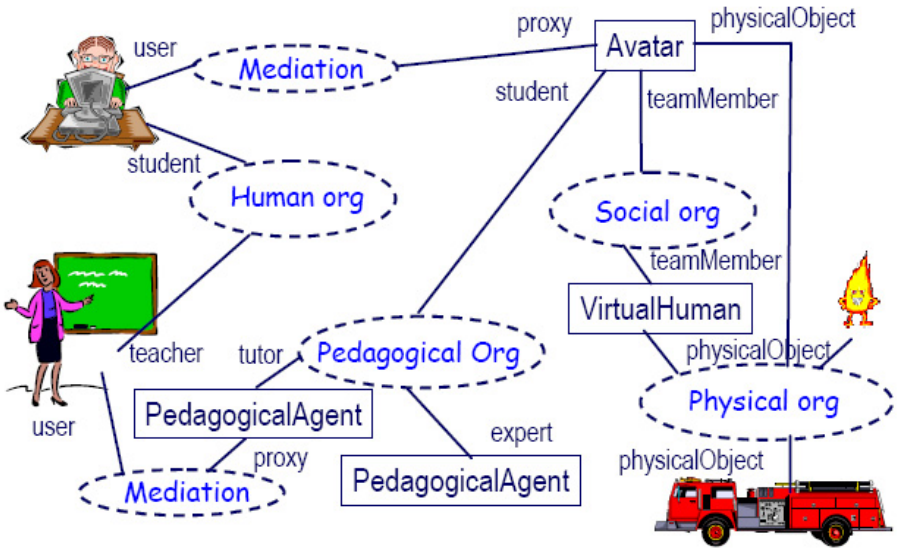

Figura 2.39: Arquitectura organizacional de MASCARET

El modelo de interfaz no se considera actualmente en MASCARET, por lo que la comunicación se realiza directamente desde los otros modelos del SIT.

Como se ha mencionado anteriormente, el modelo pedagógico implica potencialmente a todos los demás agentes, quienes colaboran entre sí, cada uno con su rol correspondiente, para dar un soporte pedagógico colaborativo.

En un ciclo de acción, el estudiante realiza una acción, que es observada por uno de los agentes de la organización pedagógica que actualiza el modelo del estudiante. A continuación se compara el modelo del estudiante con el experto y, si se detecta un error, se compara con el modelo de error y se actualiza el modelo del estudiante. Finalmente, de acuerdo con la información de la causa del error, un agente pedagógico selecciona una acción a realizar y se le muestra al estudiante.

2.8.3.6. ADRI

ADRI (Artificial Didactic Recital Instructor) es un sistema desarrollado para ayudar a aprender a tocar el piano (Broersen y Nijholt, 2002). El EV consta de un piano virtual que se puede manejar a través de un teclado MIDI, y un grupo de agentes se encarga de monitorizar lo que toca el estudiante. Cada uno de los agentes tiene un conocimiento y una responsabilidad específica en el proceso de enseñanza.

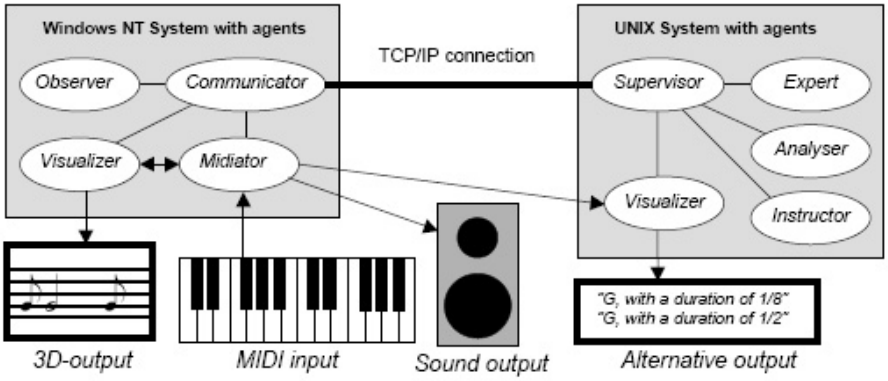

Figura 2.40: Arquitectura de ADRI

El agente *visualizador* se encarga de controlar la representación del EV; el *midiador* captura el sonido de lo que está tocando el estudiante, y el *experto* lo compara con las piezas que están almacenadas en la base de conocimiento. El agente *analizador* es capaz de añadir características concretas a una canción para mostrárselas al estudiante. El agente *navegador* decide cuál es el próximo paso a dar dentro del currículum, y el *instructor* es el encargado de enseñar esa parte de la lección al estudiante. El agente de *realimentación* está encargado de hacerle comentarios al estudiante respecto a su actuación, y el *observador* utiliza las actuaciones del estudiante para evaluarlo. Todos estos agentes están bajo la supervisión de un agente *supervisor* y otro de *comunicación*, de manera que se facilite la adición o eliminación de agentes.

Aunque el sistema es todavía un prototipo, ya se ha llevado a cabo una implementación mínima de algunos de los agentes propuestos. Pese a que los autores afirman que el grupo de agentes puede ser fácilmente modificado, no aportan pruebas de esta afirmación, y el diagrama de la arquitectura tampoco permite corroborarlo.

2.8.3.7. LAHYSTOTRAIN

Lahystotrain es un proyecto desarrollado para entrenar a cirujanos en operaciones de laparoscopia e histeroscopia (Arcos et al., 2000; Voss et al., 2000; Muller-Wittig et al., 2001). Debido a los riesgos que comporta el entrenamiento en el propio quirófano y a las cuestiones éticas que implica la práctica con animales, la utilización de un sistema de entrenamiento basado en realidad virtual ha supuesto una alternativa bastante atractiva para la formación de cirujanos en este área.

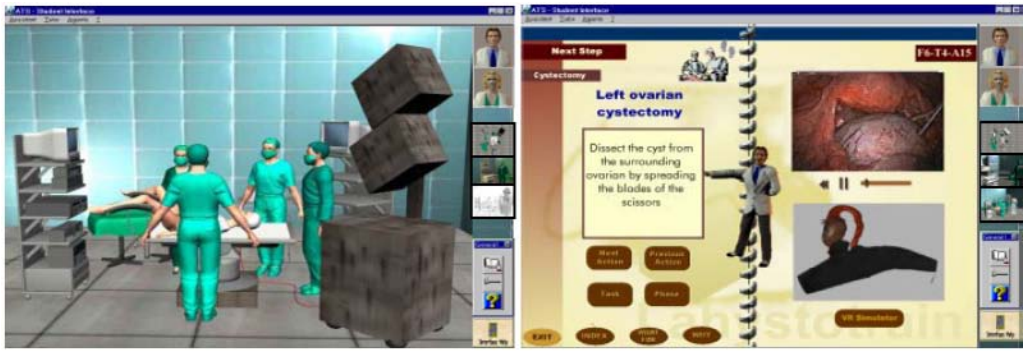

Figura 2.41: Interfaz del estudiante de Lahystotrain

El sistema admite dos tipos de usuarios: estudiantes, separados en cuatro niveles de experiencia, e instructores, encargados de supervisar la evolución de los estudiantes. Estos últimos tienen la capacidad de proponer nuevos ejercicios y consultar información respecto a los distintos ejercicios y estudiantes.

La formación tiene lugar en dos pasos. En el primero, el estudiante adquiere los conocimientos teóricos relacionados con las patologías a tratar, y en una segunda fase pone a prueba esos conocimientos y adquiere habilidades prácticas a través de la realización de diversas operaciones.

Durante la intervención, el estudiante recibe la ayuda de un asistente que le proporciona explicaciones de forma proactiva y reactiva. Las explicaciones proactivas tratan de distintos aspectos del procedimiento que se lleva a cabo, como la mejor forma de acceder a un órgano, y se proporcionan a petición el estudiante. Las reactivas se dan cuando el estudiante comete un error o sucede algo anómalo en la operación. Al no ser a petición del estudiante, pueden interrumpirle y desconcentrarle, por lo que se permite que el estudiante defina un nivel de interrupción.

Existe un agente tutor que se encarga de supervisar el entrenamiento: registra información del estudiante y datos de sus habilidades y genera y ejecuta el plan de instrucción. Este plan se crea y modifica de manera dinámica, en función de la información recopilada acerca del estudiante y sus resultados durante la sesión actual.

Existen otros tres agentes, un cirujano auxiliar, una enfermera y un anestesista, que completan el equipo que realiza las operaciones emulando el comportamiento de los mismos. El estudiante debe aprender su papel dentro del equipo, así como la forma de coordinarlos.

La aplicación está compuesta por una simulación de RV y un sistema de entrenamiento, el ATS (*Advanced Training System*). Éste, a su vez,

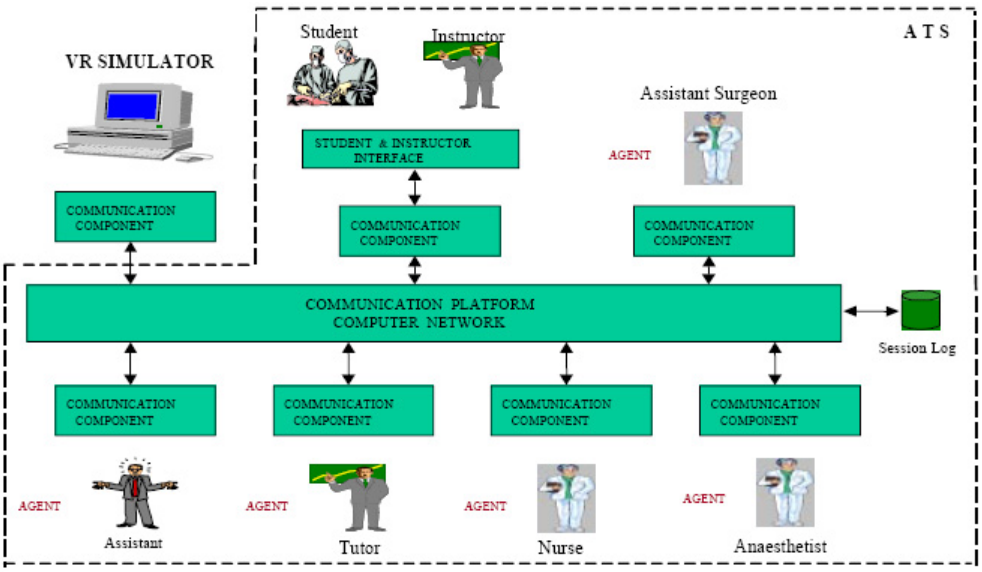

Figura 2.42: Arquitectura de Lahystotrain

se compone de dos agentes pedagógicos, el asistente y el tutor, tres agentes de apoyo, el cirujano auxiliar, la enfermera y el anestesista, y una interfaz de usuario. Esta interfaz se utiliza para que el estudiante solicite explicaciones, así como la intervención del asistente y el tutor. También permite comunicarse con la enfermera, el anestesista y el cirujano auxiliar. Además, existe una base de datos que almacena el modelo del estudiante. Ésta contiene una parte estática para guardar los datos del estudiante, y otra dinámica, donde se van almacenando las operaciones realizadas por el estudiante, su resultado, los errores cometidos y su conocimiento de las patologías y el instrumental usado.

El asistente debe controlar la evolución del ejercicio, por lo que recibe del simulador tanto las acciones del estudiante como el estado del paciente, y envía de vuelta instrucciones acerca de lo que debe suceder, como acciones de los agentes y complicaciones en la operación. Las acciones recibidas se pasan al monitor, que las compara con los datos almacenados en la base de datos del dominio para detectar errores. El módulo de diagnóstico recibe los errores y trata de descubrir las causas. A continuación, el explicador genera el contenido y estructura de las explicaciones, además de gestionar el diálogo con el estudiante. Finalmente, el avatar animado se encarga de representar al tutor dándole la explicación pertinente al estudiante.

La responsabilidad principal del tutor consiste en supervisar y gestionar el entrenamiento. El módulo supervisor de entrenamiento analiza las acciones

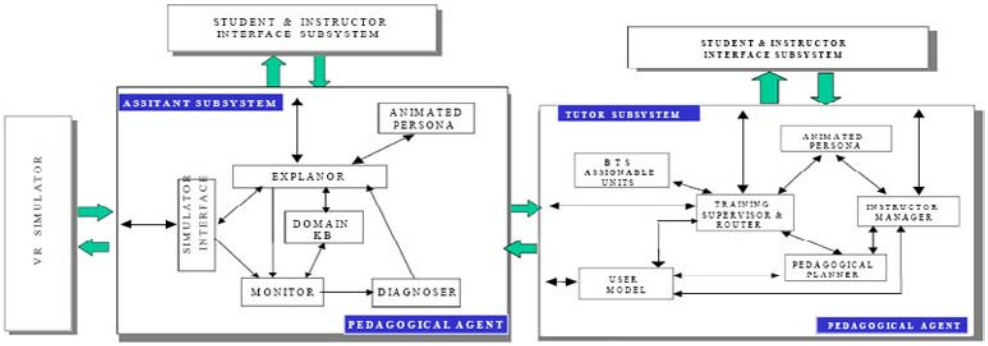

Figura 2.43: Arquitectura de los agentes Tutor y Asistente

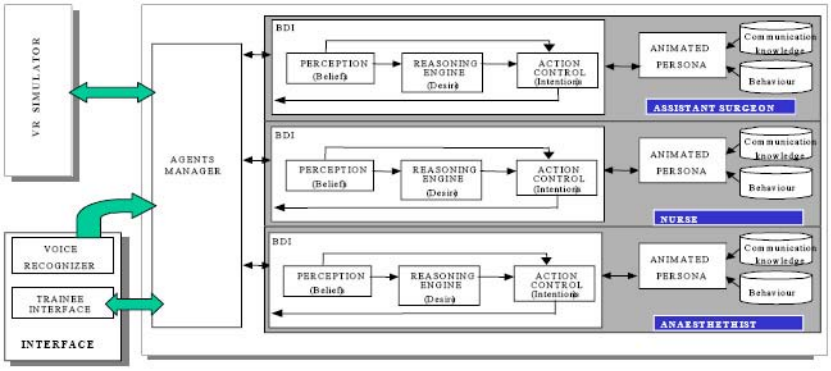

Figura 2.44: Arquitectura de los agentes de comportamiento

del estudiante para evaluar sus errores y replanificar los objetivos de la sesión si es necesario (i.e. si hay errores graves o si lo solicita el estudiante). El planificador pedagógico es el encargado de construir el plan de entrenamiento, tanto al principio de la sesión como cuando lo solicita el supervisor de entrenamiento. Este plan se construye de acuerdo con el conocimiento pedagógico y la información almacenada en el modelo del estudiante.

Los tres agentes de apoyo tienen un comportamiento reactivo, y constan de cinco módulos. El gestor de agentes obtiene datos de la simulación y le envía las instrucciones que deben ejecutar los agentes. El módulo de percepción recibe mensajes de otros componentes, identifica eventos relevantes y mantiene un modelo del estado del paciente. El motor de razonamiento interpreta las entradas recibidas por el módulo de percepción, las procesa y genera las respuestas adecuadas en forma de tareas que se deben llevar a cabo en la intervención. El control de acciones descompone las tareas generadas por el motor de razonamiento en acciones de más bajo nivel que se

envían a los demás módulos. El avatar animado utiliza un espacio de acciones con animaciones simples y frases cortas que se reproducen en el simulador.

2.9. Análisis del Trabajo Relacionado

Aunque existen en la literatura más sistemas que los presentados en la sección anterior, los descritos constituyen una representación significativa de los distintos tipos de sistemas que pueden encontrarse en la literatura. Además, muchos de ellos constituyen sistemas de referencia para el resto de autores del área, y tanto el número de citas que los referencian como el número de artículos sobre ellos en publicaciones relevantes específicas del área son superiores a los de otros sistemas. Algunas de las aplicaciones elegidas, pese a su menor relevancia, se han incluido por tratarse de sistemas con características similares al que es objeto del presente trabajo, por lo que se ha creído conveniente incluirlos para tener sistemas con los que poder establecer una comparación más cercana.

Lo primero que llama la atención de los sistemas descritos es que los autores de los Entornos Virtuales educativos sin tutores inteligentes, como REP, NICE o GCW, parecen estar de acuerdo en el hecho de que su utilidad parece dudosa.

Los usuarios de estos sistemas casi siempre son niños, y aunque parecen estar motivados para utilizar estas aplicaciones, los resultados obtenidos en cuanto a su aprendizaje no parecen mostrar un beneficio que justifique el uso de estas aplicaciones, si es que ese beneficio existe. De hecho, uno de los autores señala que los niños conceden mayor importancia a lo aprendido en el EV que a lo aprendido en clase, por lo que si adquieren conceptos erróneos será más difícil hacer que lo corrijan mediante el concepto correcto.

Este hecho refuerza la idea de centrar los sistemas objeto de estudio de este trabajo en los Entornos Virtuales de Entrenamiento, en lugar de los dedicados a la enseñanza.

En cuanto a estos otros, también es destacable que los autores de sistemas basados en diálogos, como AutoTutor, hacen hincapié en dos hechos. El primero es que resulta necesario establecer una estructura bastante rígida en el sistema de diálogos, pues si no se dificulta enormemente el desarrollo de la aplicación. Esto trae como consecuencia el segundo hecho, y es que estos sistemas reciben una valoración significativamente más elevada por parte de usuarios noveles en el dominio, mientras que los usuarios un poco más avanzados parecen dejar de ver su utilidad. Este fenómeno sucede incluso en sistemas como TLCTS, en el que parte del

aprendizaje está basado en diálogos, aunque tenga una parte importante de entrenamiento procedimental.

Respecto a los sistemas de entrenamiento procedimental, el aspecto más reseñable se encuentra en el hecho de que los usuarios tampoco los valoran bien cuando no les dejan la libertad suficiente para llevar a cabo las acciones que les parecen adecuadas, aunque éstas resulten ser incorrectas. Así, los sistemas que han sido evaluados por usuarios han obtenido resultados negativos en ese aspecto, pues eran sistemas con planes predefinidos o con una capacidad de variación no muy grande.

En este sentido, los usuarios valoran la flexibilidad tanto en la tutoría como en el comportamiento del tutor, lo que lleva a la utilización de planificadores, no excesivamente sofisticados, para planificar la tutoría, los movimientos y los comportamientos del tutor, en caso de encontrarse éste presente en el entorno.

En general, tanto en sistemas de enseñanza como de entrenamiento se aprecia el hecho de que los usuarios parecen irritarse si el tutor les interrumpe con excesiva frecuencia, por lo que algunos de estos sistemas ofrecen la posibilidad de deshabilitar la tutoría o de establecer un nivel más tolerante del tutor para que intervenga menos ante los fallos de los estudiantes.

Respecto a la estructura de los distintos sistemas que incorporan tutores inteligentes, existen notables variaciones entre unos y otros.

Los sistemas que se han centrado en el desarrollo de un agente pedagógico, como Steve, Doris y Vincent, centran su arquitectura en torno a éste, y presentan una estructura con tres bloques de tipo *Percepción - Razonamiento - Acción*. Por el contrario, los sistemas en los que el agente pedagógico está presente pero no es un elemento tan importante, la estructura suele diferir de la anterior, y puede asemejarse a la de un Sistema Inteligente de Tutoría, aunque los distintos autores se toman las licencias que consideran necesarias.

Además, sistemas como Jacob tratan específicamente el problema de representar de forma separada el mundo y el conocimiento del experto. De la misma manera, en sistemas que se han visto enfrentados con el problema se señala la necesidad de incluir de forma explícita algún elemento que trate la interacción con simuladores externos. Tal es el caso de Steve y Jonás.

Una característica que se ha buscado en esta tesis a la hora de analizar todas las aplicaciones son referencias sobre su modificabilidad y la posibilidad de reutilizar sus componentes.

En general, se aprecia que no se produce reutilización salvo en sistemas desarrollados por el mismo grupo en distintos momentos. Así sucede con Adele, Steve, MRE y, en menor medida, TLCTS. También sucede con Nice

y REP. Sin embargo, pocas veces se produce una verdadera reutilización de elementos.

Sí cabe destacar, sin embargo, que tanto Steve como AutoTutor hacen uso de un mecanismo de paso de mensajes a través de un módulo que se encarga de retransmitirlos y que, según sus autores, ha introducido mucha flexibilidad en los diseños y ha incorporado la posibilidad de añadir nuevos módulos sin que los existentes se vean afectados.

Respecto a los sistemas que se basan en agentes, cabe destacar que en general no se obtienen los beneficios que se le atribuyen a los agentes, llegando al extremo de los autores de INES, que indican que es bastante difícil introducir ciertas modificaciones, como un agente nuevo. Lo mismo se puede decir de sistemas como MASCARET, en el que hay algunas organizaciones de agentes en las que los agentes conocen la identidad de todos los demás.

Por tanto, cabe destacar que los sistemas existentes son poco flexibles en su funcionalidad y poco modificables en su estructura. En los casos en los que se basan en la estructura de un SIT, lo hacen con una interpretación bastante libre de la misma. Estas dos últimas características son las que más dificultan la reutilización de componentes entre distintas aplicaciones.

2.10. Trabajo Previo

El origen de la presente tesis doctoral se basa en los trabajos previos del autor en el desarrollo de Entornos Virtuales de Entrenamiento. En ellos quedó patente la necesidad de reutilizar trabajos anteriores para evitar tener que desarrollar los sistemas por completo y desde el principio cada vez que se desea obtener una aplicación de entrenamiento para un dominio nuevo.

A continuación se realiza una descripción de los dos trabajos previos principales, junto con los problemas que presentaron y las lecciones aprendidas.

2.10.1. PRVIR

PRVIR (Protección Radiológica Virtual) es un proyecto que se desarrolló para dar formación al personal de la Central Nuclear de Vandellós (Méndez et al., 2001; Méndez et al., 2004).

El trabajo en una central nuclear presenta el inconveniente de que éste es un entorno peligroso en el que los trabajadores están expuestos a recibir dosis de radiación perjudiciales para la salud.

Por ese motivo, lo habitual es que el acceso a zonas restringidas de la central sólo se produzca cuando el reactor está parado y los niveles de

radiación son prácticamente inexistentes. Como tener un reactor parado es económicamente costoso, cuando se realiza una parada de un reactor se intenta que dure el menor tiempo posible, por lo que todas las tareas necesarias deben llevarse a cabo con rapidez y eficiencia. Para ello se requiere que el personal esté perfectamente formado.

Dado que el acceso a la central no es una opción viable para realizar el entrenamiento, habitualmente tiene lugar en reproducciones de determinadas zonas de interés de la central, si bien la construcción de dichas reproducciones suele resultar cara y necesita de mucho espacio físico.

Por todo ello, se decidió experimentar con una aplicación basada en la utilización de Entornos Virtuales en un entorno de bajo riesgo, como es el procedimiento de acceso y salida de la central y de sus zonas controladas.

Para ello, la aplicación constaba de una parte de formación básica que utilizaba elementos multimedia para que el alumno fuese aprendiendo conceptos relacionados con la energía nuclear y radiactividad, que luego ponía en práctica a través de la realización de ejercicios.

Como temas avanzados de la formación se incluían procedimientos asociados a la entrada y salida de las zonas controladas, y los ejercicios consistían en practicar esos conocimientos dentro de un Entorno Virtual.

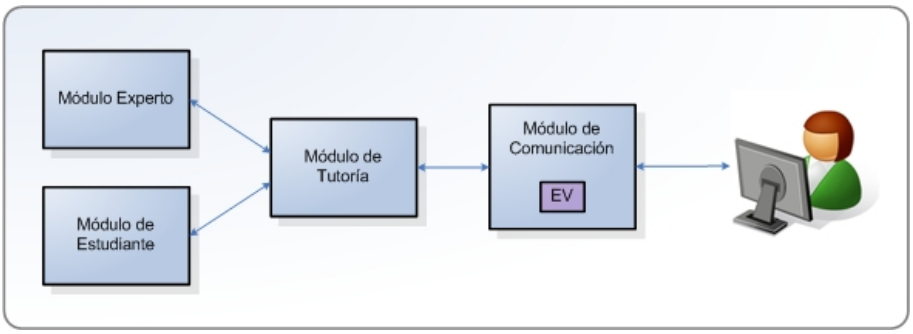

Figura 2.45: Arquitectura de PRVIR

La aplicación constaba de un Sistema Inteligente de Tutoría con la arquitectura clásica de estos sistemas, y el Entorno Virtual se incluía como un elemento más del módulo de comunicación y, como tal, con comunicación directa con el módulo de Tutoría.

Los alumnos podían desplazarse por el modelo 3D de la central nuclear libremente y cada vez que salían de una estancia y entraban en una nueva se modificaba el contenido de un cuadro de texto donde podían ver los elementos de interés existentes en esa estancia, junto a otro cuadro con los objetos que ellos llevaban encima en ese momento. Las acciones que podían llevar a cabo

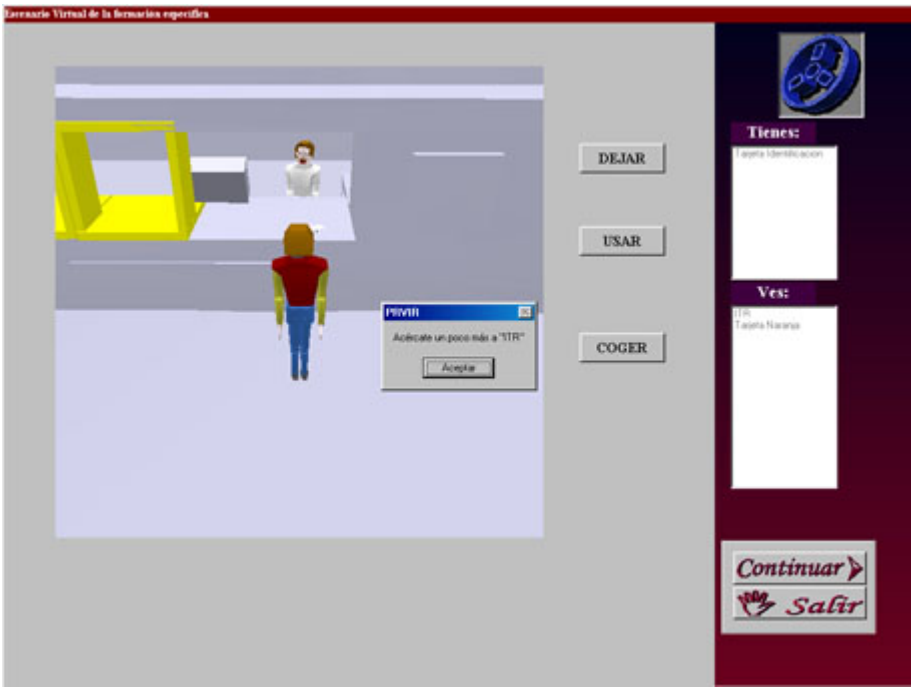

Figura 2.46: Interfaz de PRVIR

eran coger un objeto de la estancia, dejar un objeto de su inventario o usar un objeto de cualquiera de los dos grupos. También podían interactuar con algunos personajes que aparecían en algunos lugares, pero siempre como consecuencia del uso de un objeto.

El alumno realizaba sus acciones a través de los controles de la ventana en la que se mostraba el Entorno Virtual, y la ventana le comunicaba al EV la acción realizada. En ese momento, el EV comprobaba si era físicamente posible realizar la acción, y además consultaba con el módulo de tutoría si se debía permitir la realización de esa acción. En caso afirmativo, la acción se llevaba a cabo, se le mostraba al estudiante y se informaba de ello al módulo de tutoría. En caso negativo, el módulo de tutoría proporcionaba un mensaje de error que se le mostraba al usuario a través de un cuadro de diálogo.

Aunque el desarrollo de esta aplicación supuso una experiencia enormemente positiva, adolecía principalmente de tres problemas que la hacían poco apropiada para su reutilización:

- El EV era un elemento más de la interfaz gráfica, que además se comunicaba directamente con el SIT, por lo que su sustitución resultaba compleja y obligaba a reimplementar parte de la lógica de la aplicación.

- La información semántica del entorno estaba programada dentro del código del EV, por lo que plantear nuevos escenarios de entrenamiento implicaba recompilar la aplicación.
- La secuencia de acciones a realizar dentro de una actividad se encontraba almacenada dentro de una base de datos, por lo que las posibilidades de desviarse de esa secuencia eran nulas.

Del desarrollo de este proyecto se aprendió que era necesario resolver los tres problemas anteriores, lo que se tradujo en separar el EV del Módulo de Comunicación, manejar por separado la información semántica del EV e introducir la capacidad de planificar las actividades en lugar de manejarlas de forma prediseñada.

2.10.2. Integración de Steve y HeSPI

El proyecto VRIMOR supuso la continuación del trabajo relacionado con el desarrollo de herramientas basadas en Entornos Virtuales para centrales nucleares. En este caso, uno de los objetivos del proyecto consistía en la elaboración de una herramienta que permitiese planificar operaciones de mantenimiento en las centrales nucleares.

El resultado fue HeSPI (Herramienta de Simulación y Planificación de Intervenciones), un Entorno Virtual con una interfaz basada en ventanas que permitía a un operario diseñar una intervención en una zona de la central previamente modelada en tres dimensiones.

HeSPI utilizaba un EV construido sobre Jack (Badler et al., 1993), una aplicación desarrollada en la Universidad de Pensilvania para realizar estudios ergonómicos con figuras humanas. Jack facilitaba la carga de modelos 3D creados con otras aplicaciones y la utilización de avatares con figura humana y movimientos realistas.

Sobre esta aplicación se construyó una interfaz basada en ventanas que proporcionaba acceso a las acciones que podía realizar el operario y una visión de la planificación actual de la intervención similar a un diagrama de Gantt.

Esta aplicación se utilizó posteriormente, junto con un equipo de la Universidad del Sur de California (USC), para integrarla con Steve (Rickel y Johnson, 1999), un tutor inteligente diseñado para realizar entrenamientos en Entornos Virtuales.

Puesto que ambas aplicaciones no se habían diseñado para trabajar conjuntamente, se consideró como una oportunidad para probar varios factores:

- La flexibilidad de ambas aplicaciones para interactuar con otras.

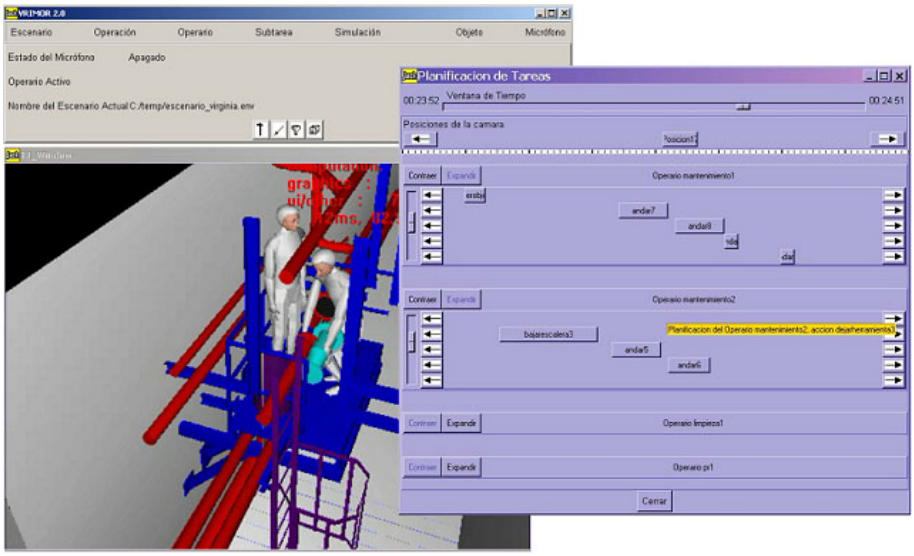

Figura 2.47: Interfaz de HeSPI

- La posibilidad de utilizar HeSPI como una aplicación de entrenamiento, además de como herramienta de planificación.

- La facilidad para adaptar Steve a dominios distintos a los considerados originalmente.

A pesar de las aparentes dificultades iniciales, la integración se llevó a cabo sin excesivos problemas (Méndez et al., 2003). Además, sirvió para identificar necesidades en el diseño de ambas aplicaciones.

En general, ambas aplicaciones proporcionaron la funcionalidad requerida por la otra, pero se identificaron problemas relativos al flujo de control entre ellas y a la representación de la información que utilizaba cada una, lo que hizo necesario incorporar un traductor entre las dos para solucionar ambos problemas.

Además, las dos aplicaciones asumían que se les podía proporcionar determinada información que la otra aplicación no era capaz de manejar, lo que también obligó a mantener registros temporales de información y a utilizar información por omisión allí donde no era posible generar los datos requeridos.

En la figura 2.48 se puede ver un esquema del resultado de la integración de ambas aplicaciones.

Esta integración sirvió como experiencia para identificar las necesidades de ambos tipos de aplicaciones y, sobre todo, el tipo de dependencias que se

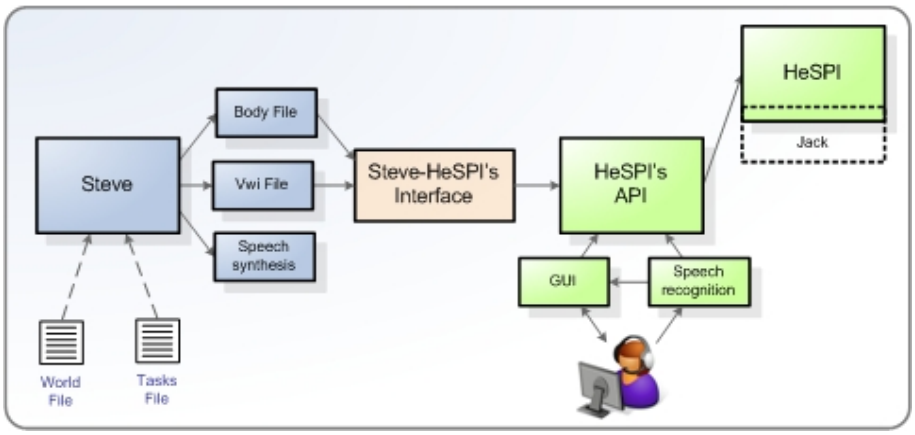

Figura 2.48: Integración de Steve y HeSPI

deben evitar, o al menos documentar, cuando se desarrollan tanto Entornos Virtuales como Tutores Inteligentes que se pretendan reutilizar en otros dominios o junto a otras aplicaciones.

2.11. Reflexiones

Como puede apreciarse por lo expuesto en las secciones anteriores, existe un gran número de comunidades que están contribuyendo al desarrollo de los Entornos Virtuales de Entrenamiento: gráficos 3D, Inteligencia Artificial, comunicaciones, psicología o sociología. Esto, por una parte, resulta enormemente positivo, pues enriquece la visión que se tiene de estos sistemas. Por otra parte, sin embargo, presenta serios retos que son cada vez más urgentes de resolver a medida que crece la cantidad y complejidad de estos sistemas.

En primer lugar, la falta de componentes que puedan ser reutilizados para construir estos sistemas hace que los miembros de todas las comunidades se vean obligados a desarrollar componentes que están fuera de su campo de investigación. El resultado son, por ejemplo, aplicaciones con excelentes capacidades de modelado del estudiante, pero con una apariencia bastante pobre y una ejecución lenta, o sistemas muy vistosos pero cuyas posibilidades son bastante limitadas.

En los casos en los que sí es posible encontrar algún elemento que se pueda reutilizar, lo habitual es experimentar serios problemas a la hora de integrarlos con la parte de desarrollo propio, si no una imposibilidad total, lo cual conduce también a fracasos o a sistemas con carencias notables.

Finalmente, en muchas ocasiones, la falta de formación y de experiencia en el desarrollo de aplicaciones informáticas hace que gran parte de estos sistemas presente problemas de diseño, implementación y mantenimiento, por no hablar de su reutilización.

Se puede concluir, por tanto, que sería deseable que los miembros de las distintas comunidades no tuviesen que desarrollar ellos mismos toda la infraestructura que necesitan para implementar los frutos de sus investigaciones. En su lugar, resultaría más sencillo para ellos y más fructífero para sus investigaciones que pudiesen utilizar componentes ya desarrollados que pudiesen interconectarse fácilmente entre sí.

El primer punto al que hay que prestar especial atención para conseguir este objetivo se centra en el diseño de la arquitectura software de estos sistemas. Este es un tema al que el mundo del software no le está prestando la atención que requiere, y el área de los Entornos Virtuales de Entrenamiento no constituye una excepción.

Por ello, es necesario realizar un esfuerzo para alcanzar estándares que faciliten el desarrollo y la reutilización de los distintos elementos que pueden formar parte de un Entorno Virtual de Entrenamiento, lo cual redundará en el beneficio de todos los integrantes de esta comunidad. Con la elaboración de esta tesis doctoral se pretende dar un paso en esa dirección.

Parte II

Propuesta de Solución

Capítulo 3

Planteamiento de la Solución

"The sciences do not try to explain, they hardly even try to interpret, they mainly make models"
— John von Neumann

3.1. Introducción

El propósito del presente capítulo es definir los objetivos que se persiguen con la realización de esta tesis doctoral.

A lo largo de las siguientes secciones se justifica por qué se consideran los Entornos Virtuales de Entrenamiento con Tutoría Inteligente, así como la razón de utilizar agentes en el diseño de su arquitectura y posterior implementación. Se explicará también la motivación de enfocar su construcción desde el punto de vista de la arquitectura software, así como la utilidad de proporcionar recomendaciones metodológicas para la modificación de la arquitectura propuesta.

3.2. Sistemas Objeto de la Investigación

Los Entornos Virtuales (EV) son un tipo de sistema software de aparición relativamente reciente. Los primeros ejemplos se pueden situar a finales de los años 70, si bien éstas eran aplicaciones textuales enfocadas principalmente hacia el entretenimiento. A mediados de los 80 se les empieza a dotar ya de interfaz gráfica, y a principios de los 90 surgen las primeras aplicaciones en tres dimensiones (3D). A partir de ese momento, los EVs han sido utilizados con gran variedad de fines, incluyendo entretenimiento, formación, trabajo colaborativo, simulación o *marketing*.

El usuario de un sistema de Realidad Virtual se encuentra asistido por un conjunto de dispositivos que le permiten interactuar con el Mundo Virtual, y que refuerzan la sensación de inmersión dentro del mismo. Dependiendo del tipo de configuración realizada, basada en un conjunto determinado de componentes auditivos, visuales, táctiles y un software de simulación, puede hablarse de distintos niveles de inmersión en el sistema:

1. La denominada *Realidad Virtual de Escritorio* es la que proporciona los niveles más bajos de inmersión, ya que sólo permite interactuar con el mundo a través de un teclado, un joystick o un ratón, y donde el mundo se visualiza en un monitor de computadora normal y corriente.

2. La *Realidad Virtual Inmersiva*, en cambio, emplea dispositivos tales como HMD (*Head Mounted Displays*, cascos de Realidad Virtual), guantes, sistemas de rastreo del movimiento, etc. La sensación de presencia en el EV es mucho mayor en este tipo de sistemas, pero debido a que su coste es muy elevado, queda fuera del alcance de la mayor parte de los usuarios.

La inmersión en un mundo virtual hace desaparecer la interfaz persona-ordenador, de forma que nuestras experiencias en el mundo virtual pueden llegar a ser de la misma calidad que nuestras experiencias en el mundo real. El conocimiento que se genera es directo, personal, subjetivo y en primera persona, frente al conocimiento en tercera persona que se adquiere cuando nos ha sido enseñado por alguien más.

Numerosos estudios indican que la tecnología de la Realidad Virtual es una poderosa herramienta para la enseñanza, debido fundamentalmente a su capacidad para proveer entornos inmersivos, multisensoriales, creíbles y experienciales, entre otras características (Bricken, 1990a; Bricken, 1990b; Byrne, 1993; Winn, 1993; Larijani, 1994; Pantelidis, 1996; Youngblut, 1998; Hughes y Moshell, 1997). Estos trabajos sugieren que la aplicación de esta tecnología en los procesos de enseñanza/aprendizaje, ya sea en la educación o en el entrenamiento, puede generar beneficios superiores con respecto a los obtenidos mediante la utilización de tecnología no basada en Realidad Virtual. Sin embargo, debido al actual estado de desarrollo de la Realidad Virtual, muchas de las aplicaciones que se construyen en instituciones de investigación y universidades se encuentran todavía sin comercializar.

Dentro de los EVs educativos se pueden encontrar tres tipos de aplicaciones bien diferenciadas. En primer lugar, existe un tipo de EVs que sirven de soporte al *e-learning*, como INVITE (Kuljis, 2002). En este tipo de aplicación, el EV se utiliza como lugar de encuentro para dar charlas, impartir lecciones o establecer discusiones.

En segundo lugar se encuentran los EVs para enseñanza, como NICE (Johnson et al., 1998a) y REP (Johnson et al., 1999b). Suelen ser aplicaciones que reproducen un entorno más o menos real, aunque con ciertas licencias, y que se utilizan, en la mayor parte de los casos, para enseñar conceptos a niños en edad escolar. Aunque existe un número razonablemente amplio de aplicaciones de este estilo, los propios autores reconocen que su utilidad no es concluyente, pues los resultados obtenidos son similares a los los de situaciones en que no se usan estos entornos. Asimismo, posiblemente por su novedad, los niños tienden a dar mayor importancia a lo aprendido al usar el EV que a lo explicado por sus profesores, de manera que, si se aprende algo de manera incorrecta, resulta mucho más difícil lograr que los niños desechen esa idea y aprendan el concepto correcto.

En tercer lugar tenemos los Entornos Virtuales de Entrenamiento (EVE). Este tipo de EV suele reproducir un entorno real de manera bastante fidedigna, y suele estar enfocado a la enseñanza de tareas procedimentales que posteriormente habrán de llevarse a cabo en la realidad, razón por la cual es importante que el EV sea lo más parecido posible al escenario real, tanto en apariencia como en funcionamiento. Hasta la fecha son éstos los que han dado mejores resultados en cuanto a su eficacia.

Los EVEs están experimentando un gran avance durante los últimos años, siendo utilizados fundamentalmente en tres áreas: medicina (Korze et al., 2001), industria (Méndez et al., 2001) y entrenamiento militar (Rickel et al., 2002b). Dentro de los EVEs podemos establecer dos grupos claramente diferenciados:

1. En algunos casos, el EV no posee ninguna característica que lo oriente específicamente hacia el entrenamiento, sino que el uso que se le da es el que hace que sea considerado un EVE.

2. En otros casos se han incorporado sistemas de tutoría más o menos inteligentes que apoyan el proceso de entrenamiento.

Dentro de este último grupo se puede encontrar una amplia gama de aplicaciones, en función de su inteligencia y funcionalidad. Estas aplicaciones varían desde aquellas que controlan totalmente las acciones del estudiante, permitiendo realizar únicamente las acciones correctas para completar la actividad de entrenamiento, hasta otras que proporcionan tutores con representación física en el EV y que permiten que el alumno realice acciones erróneas y haga preguntas para saber qué tiene que hacer.

En (Lester et al., 1997c) se identifica el llamado *efecto persona*, el cual, según los autores de la investigación, parece tener un impacto considerable

en el aprendizaje. Este trabajo muestra que los alumnos obtienen mejores resultados cuando reciben las explicaciones de un tutor con presencia física en el EV y con apariencia humana, en lugar de recibirlas de un tutor con apariencia no humana o de un tutor sin representación física. Por esta razón, es cada vez más habitual que los EVEs proporcionen tutores virtuales que acompañan al alumno durante el entrenamiento.

Otro aspecto a destacar de este tipo de entornos es la posibilidad de realizar entrenamientos en grupo, ya que habitualmente las actividades que requieren entrenamiento se suelen llevar a cabo por más de una persona. Como se ha mencionado anteriormente, se requiere que el EVE sea lo más parecido a lo que el alumno se encontrará en el entorno real, y resulta cada vez más importante que más de una persona pueda tomar parte en el entrenamiento.

Considerando las características enumeradas a lo largo de esta sección, los sistemas que van a centrar la atención de este trabajo de tesis doctoral son Entornos Virtuales de Entrenamiento con capacidad para soportar múltiples usuarios y que incorporan tutores inteligentes, con o sin presencia física en el entorno.

Como se ha podido observar en los trabajos descritos en el estado de la cuestión, existe una amplia gama de funcionalidades que este tipo de aplicación puede ofrecer. Por lo tanto, resulta conveniente delimitar la funcionalidad que debe ofrecer la arquitectura que se va a proponer en esta tesis. No se pretende elaborar una descripción detallada de la funcionalidad de los sistemas objeto de estudio a la manera de una especificación de requisitos. Esto requeriría, por una parte, cierta dependencia del dominio concreto en el que se quisiese trabajar, y por otro, incluiría numerosos detalles que no tendrían mayor impacto en la arquitectura. Además, como se indica en (Clements et al., 2002a), no es necesario disponer de toda la información detallada acerca del sistema para poder comenzar el diseño de la arquitectura.

Por lo tanto, lo que aquí se recoge es una descripción de la funcionalidad que se espera obtener en una aplicación que utilice la arquitectura propuesta como solución del problema que se plantea. Parte de la funcionalidad descrita se ha obtenido a partir de las aplicaciones desarrolladas por el equipo dirigido por la directora de la presente tesis doctoral, mientras que otra parte es el resultado del análisis de las investigaciones descritas en el capítulo 2 de estado de la cuestión.

Dado que uno de los objetivos que se persiguen es la posibilidad de intercambiar componentes de la arquitectura, la descripción de la funcionalidad se va a organizar basándose en los módulos que componen los Sistemas Inteligentes de Tutoría, lo cual no obligará a que la arquitectura de la aplicación responda a la estructura clásica de un SIT.

3.2.1. Funciones de Tutoría

El sistema deberá plantear distintas actividades para que el alumno sea capaz de resolverlas dentro del EV, ya sea de manera supervisada por el SIT o sin supervisión alguna. Si las actividades se realizan con supervisión, el alumno podrá realizar preguntas acerca de lo que tiene que hacer, y el SIT podrá decidir si las responde o no, así como el nivel de detalle con el que ofrece la respuesta.

Además, en el caso en que se detecte que el estudiante realiza acciones incorrectas, se podrá decidir si dejarle continuar o si detenerlo e indicarle cuál es el error para que intente realizar una acción diferente. Tanto en este caso como si se queda parado sin saber qué hacer, el sistema podrá ofrecerle al estudiante pistas que le ayuden a continuar.

Idealmente, la tutoría deberá adaptarse a las características de aprendizaje del alumno, lo cual se conseguirá a través del modelo que se realice del estudiante. En cualquier caso, la tutoría también deberá poder realizarse de una manera estándar, sin tener en cuenta las características del estudiante.

3.2.2. Funciones de Experto

Es responsabilidad del sistema como experto poseer el conocimiento que debe ser enseñado al estudiante. Dentro de este conocimiento se encuentra el necesario para resolver los tipos de ejercicios que se le plantean al alumno para que los lleve a cabo dentro del Entorno Virtual.

Para poder realizar una tutoría más personalizada y flexible, el experto debe ser capaz no sólo de proporcionar una solución a los ejercicios, sino que también deberá poder resolver las situaciones planteadas por el alumno cuando resuelva los ejercicios de manera distinta a la propuesta por el experto. Además, deberá facilitar a los estudiantes explicaciones sobre los ejercicios, tendrá que ser capaz de responder a sus preguntas y proporcionar información para darles pistas que les ayuden con la resolución de un ejercicio.

Con el objetivo de personalizar la tutoría, deberá facilitar al tutor la información que éste le solicite para, por ejemplo, saber si es correcta la acción que quiere llevar a cabo un estudiante, de manera que, de no ser así, el tutor pueda decidir si le deja continuar con su realización.

Puesto que el objetivo del sistema es posibilitar el entrenamiento en un Entorno Virtual, el experto debe tener conocimiento no sólo de la materia a impartir y de cómo resolver ejercicios, sino también de la estructura física de los escenarios y su contenido, de manera que sea posible saber cómo moverse por un escenario o contestar una pregunta respecto a la situación de un objeto.

En cualquier caso, debería ser posible realizar una tutoría menos sofisticada en caso de que el experto no proporcione alguna de las funcionalidades mencionadas.

3.2.3. Funciones de Modelado del Estudiante

El propósito de realizar un modelo del estudiante es adaptar el proceso de tutoría a las características del alumno. Para ello, es necesario que la estrategia de tutoría empleada haga uso de los datos que se vayan recopilando sobre el estudiante.

La información que se recopila está relacionada tanto con las acciones que el estudiante realiza como con sus movimientos por el EV. También se puede registrar el tiempo que tarda en realizar una determinada acción o grupo de acciones, cuántas preguntas realiza, de qué tipo, cuántas pistas hay que darle, con cuánto nivel de detalle o, incluso, hacia dónde mira, para poder inferir información acerca de su atención.

Como se ha dicho, el modelo del estudiante está bastante relacionado con la estrategia de tutoría, por lo que la información concreta que contenga irá en relación con la estrategia seleccionada. Ello no obstaculiza que el modelo pueda incluir información que no utilice directamente la estrategia de tutoría, ya que puede servir para ser analizada posteriormente por un tutor humano. Por otro lado, la estrategia de tutoría debería ser capaz de tomar acciones por omisión si el modelo del estudiante no contiene alguna información que sea necesaria para una mayor personalización de la tutoría.

3.2.4. Funciones Relativas al Entorno Virtual

La naturaleza y funcionalidad de los Entornos Virtuales que se pueden utilizar junto con un SIT son bastante variadas, por lo que, para una integración entre ambos con unas mínimas garantías de éxito, al menos debería ser posible que el EV informase de las acciones realizadas por el usuario, así como de las posiciones de los distintos personajes y objetos que lo pueblan.

En cuanto a las entradas que debería aceptar el EV, al menos debería ser posible cambiar la posición y orientación de los objetos, así como manejar un posible tutor con representación física dentro del EV de una manera similar a como un usuario manejaría su avatar.

Respecto a la funcionalidad que debería ofrecer a los usuarios, ésta depende enormemente del dominio en el que se realice el entrenamiento. No obstante, teniendo en cuenta que lo que se pretende es realizar un entrenamiento procedimental, debería ser posible manipular de alguna

manera los objetos que forman parte del procedimiento que se entrena. Cuando las características del escenario y del entrenamiento así lo requieran, también se deberá poder navegar por el entorno.

En cuanto a la forma de interacción entre el usuario y el EV, no existe ninguna restricción al respecto, por lo que son aceptables desde sistemas de escritorio, con un teclado, un ratón y un monitor, hasta sistemas totalmente inmersivos, que utilicen un dispositivo similar a un *CAVE*, gafas polarizadas, un casco de realidad aumentada o un dispositivo similar a un *wand*. Tanto es así que debería ser posible sustituir el EV por cualquier otro elemento que produzca las mismas salidas (como una consola o una interfaz basada en ventanas) sin que el SIT se vea afectado. Esto puede resultar de especial utilidad en el proceso de desarrollo, en el que no sería necesario poner en funcionamiento el EV para poder probar el SIT.

3.2.5. Funciones de Simulación

En determinados dominios, como sucede con el entrenamiento médico o en centrales nucleares, se puede dar el caso de que el sistema deba funcionar junto con un simulador que reproduzca el comportamiento de la central o la evolución del estado del paciente. Este simulador puede estar integrado con el EV o puede funcionar por separado, pero en cualquier caso debe ser posible que el SIT se comunique con él para obtener información relativa a la simulación que sea relevante para el proceso de tutoría.

3.3. Sobre el Uso de Agentes

Como se ha podido ver en el capítulo 2 sobre el estado de la cuestión, el término agente posee distintas definiciones, que reflejan los diferentes usos que les dan los grupos de investigación que los utilizan, así como las particularidades de las áreas de investigación en que se emplean. Por este motivo, no existe aún una definición comúnmente aceptada, así como unas características definitorias.

Esta situación provoca que el término de agente corra el peligro de perder cualquier tipo de significado que pueda resultar de utilidad, como afirman los autores de (Wooldridge y Jennings, 1998).

Dentro del área de aplicación de la tesis se maneja el concepto de *Agente Virtual Inteligente*, con el cual se denomina a un personaje que habita un Entorno Virtual, que generalmente interactúa con el usuario y que exhibe un comportamiento más o menos complejo en función de las necesidades e intereses de sus desarrolladores.

Estos Agentes Virtuales Inteligentes pueden estar diseñados e implementados de muy diversas maneras, y no existe una correspondencia directa entre ellos y los agentes que, considerados como elementos software, se utilizan para desarrollar aplicaciones de manera similar, por ejemplo, a los objetos.

Son éstos últimos, los agentes vistos como elemento de desarrollo de software, los que se considerarán en esta tesis para diseñar una arquitectura para Entornos Virtuales de Entrenamiento. A través de la interacción entre estos agentes se dotará a la arquitectura de un comportamiento que simule a un tutor, el cual, a su vez, podrá estar representado como un Agente Virtual Inteligente dentro del Entorno Virtual de Entrenamiento.

Existen diversos trabajos que ponen de manifiesto las diferencias y analogías entre objetos y agentes, como (Odell, 2002), donde se resalta la relativa similitud de ambos con una cierta ventaja para los agentes cuando se trata de construir sistemas muy interactivos, distribuidos o descentralizados. En (Zambonelli et al., 2003) se realiza una comparación entre desarrollos utilizando objetos, componentes y agentes para mostrar sus diferencias y, en última instancia, resaltar las ventajas de los últimos para el desarrollo de determinado tipo de sistemas modernos.

Para poder razonar sobre la utilización de agentes es necesario, en primer lugar, adoptar una definición de agente. Se van a considerar las definiciones proporcionadas en (Wooldridge, 1997; Wooldridge y Jennings, 1995b; Zambonelli et al., 2003), que establecen que un agente es una entidad software que posee las siguientes características:

- *Autonomía*: capacidad para tomar el control de sus propias acciones.
- *Situación*: capacidad de percibir el entorno en el que se encuentra y actuar sobre él.
- *Reactividad*: capacidad para responder a estímulos externos.
- *Proactividad*: capacidad de tomar la iniciativa para conseguir sus metas.
- *Capacidad social*: habilidad para relacionarse con otros agentes.

Esta definición de agente proporciona un nivel de abstracción mayor que el de los objetos o los componentes (Wooldridge, 1997), a pesar de que la evolución de ambos apunta a una paulatina convergencia de características (Zambonelli et al., 2003). Este mayor nivel de abstracción, como herramienta de diseño, ayuda a afrontar la complejidad que presentan los sistemas objeto de estudio.

En esa línea apunta el trabajo descrito en (Medinilla y Gutiérrez, 2007), donde los autores ponen de manifiesto que las abstracciones son una herramienta para combatir la creciente complejidad del software. Esto es posible porque, según los mismos autores, las abstracciones son un recurso de simplificación que introduce ambigüedad en los diseños. Esta ambigüedad conduce a su vez a una mayor facilidad para realizar cambios en los diseños y, como consecuencia, a una mayor modificabilidad de las aplicaciones, allí donde la introducción de ambigüedad se haya utilizado como recurso de diseño.

Los Entornos Virtuales de Entrenamiento son aplicaciones cuya complejidad es notable por varias razones: el alto grado de interactividad con el usuario que deben soportar, el considerable nivel de adaptación al estudiante que se requiere de ellos, el elevado esfuerzo que requiere modelar y simular el comportamiento de un tutor humano a través de un ordenador y la variedad de perfiles involucrados en su desarrollo. Esta complejidad es la que provoca que los agentes, como herramienta para combatirla, constituyan la mejor alternativa a explorar a la hora de elegir una forma de afrontar la construcción de estos sistemas.

De ninguna manera se pretende dar la visión de que el uso de agentes es una garantía de éxito, a pesar de que existe una tendencia relativamente generalizada a atribuir a los agentes propiedades que de por sí no poseen, como, por ejemplo, que las aplicaciones sean modificables por el hecho de usarlos. Muy al contrario, dada la relativa falta de experiencia global en este campo, el uso de agentes es un riesgo adicional que puede constituir un elemento de fracaso.

Por sus características de autonomía, reactividad y proactividad, y la flexibilidad que éstas les proporcionan en su comportamiento, los agentes parecen estar especialmente indicados para sistemas que requieren un alto nivel de interacción y adaptabilidad a los usuarios, como es el caso de los EVEs.

La característica de situación también los hace especialmente indicados para su utilización en un EV, el cual deben percibir y sobre el cual deben actuar.

La capacidad social los hace adecuados para trabajar en sistemas formados por más de un agente, ya que deberán relacionarse entre ellos para lograr un comportamiento que simule el de un tutor humano.

En el caso de los EVEs, el funcionamiento en paralelo y distribuido facilita la posibilidad de que se realicen entrenamientos en grupos geográficamente dispersos, donde cada estudiante puede tener su propio tutor o, incluso, alguno de los miembros del grupo puede estar controlado por un agente.

Existe, para finalizar, otra característica, aunque menos relevante, que hace que el uso de agentes constituya una alternativa razonable de desarrollo de software, como es la existencia de plataformas como JADE, Zeus o FIPA-OS, que proporcionan ya una infraestructura básica para poder implementar los agentes, de manera que se eliminan algunas dificultades inherentes a estos sistemas, tales como las que se señalan en (Wooldridge y Jennings, 1998). En sistemas basados en agentes, gran parte del tiempo de desarrollo se consume en el diseño e implementación de infraestructuras, como mecanismos de comunicación, mensajes, directorios o la arquitectura de los propios agentes, por lo que la existencia de plataformas para su desarrollo constituye un valor añadido que facilita su uso.

3.4. Arquitectura Software

Según se menciona en (Wooldridge y Jennings, 1998), existen bastantes sistemas desarrollados por medio de la utilización de agentes, y se dispone de cierta cantidad de resultados tanto teóricos como prácticos que atestiguan su validez. Sin embargo, aunque parece que se están produciendo avances en el aspecto más teórico de los agentes, no son tantos los trabajos realizados que ayuden a entender cómo diseñar y construir sistemas basados en agentes.

Con el interés de desarrollar sistemas modificables en los que sus componentes sean fácilmente sustituibles, el planteamiento inicial de esta investigación consistía en aportar alguna herramienta que ayudase a la consecución de este objetivo. Para ello, inicialmente se consideró la creación de una metodología como extensión de Senda3D (Sánchez, 2001), metodología para el desarrollo de Entornos Virtuales.

Se puede considerar que el propósito de una metodología de desarrollo es establecer un conjunto estructurado de actividades para obtener un sistema software de alta calidad con unos costes aceptables en un tiempo determinado (Pressman, 2004; Sommerville, 2007).

Sin embargo, cada una de las actividades de una metodología puede dar lugar a productos muy diversos, ya que serán el resultado de distintas decisiones de diseño en función de la persona que lo esté realizando. Ninguna metodología establece unas directrices tan estrictas como para que se obtenga el mismo resultado independientemente de quien lleve a cabo el trabajo, y además, raramente se consigue que un diseñador siga en la práctica todas las directrices marcadas por una metodología (Bansler y Bodker, 1993).

Llevado al problema que nos ocupa, ninguna metodología existente establece criterios para determinar qué agentes deben existir, cuáles deben

comunicarse entre ellos o cómo deben comunicarse, sino que el diseñador debe utilizar otros criterios para llegar al resultado final.

Este hecho, unido a la actual existencia de diversas metodologías para el desarrollo de sistemas basados en agentes, ha llevado a descartar la necesidad de crear otra metodología más, aunque sea específica para Sistemas Inteligentes de Tutoría basados en agentes.

En cambio, resulta de mayor utilidad centrarse en uno de esos aspectos donde las metodologías dejan al diseñador a merced de su criterio y experiencia, como es la arquitectura software. El objetivo que se plantea en este trabajo consiste en entender mejor cómo diseñar y construir un Entorno Virtual de Entrenamiento en el que el Tutor Virtual Inteligente se desarrolle a través de un sistema basado en agentes. De esta manera, la metodología que se esté utilizando, en caso de que se use alguna, puede completarse con el diseño de la arquitectura que se propone como solución en este trabajo.

3.4.1. ¿Por qué una Arquitectura?

Una posible definición de arquitectura software, similar a la ofrecida en el capítulo 2, es la que proporcionan los autores de (Kroll y Kruchten, 2003), quienes establecen que una arquitectura software comprende los principales bloques del sistema junto con sus interfaces, los componentes más relevantes y sus respectivas interfaces, y los llamados *mecanismos arquitectónicos*, soluciones efectivas para problemas comunes en el diseño de la arquitectura, similares a los patrones de diseño en la orientación a objetos.

Los mismos autores afirman que un buen diseño de la arquitectura de la aplicación es crucial para el éxito de un proyecto, por lo que debe ser uno de los principales objetivos al inicio del mismo. En esta misma línea apuntan las conclusiones de otros relevantes autores en el campo de las arquitecturas software, como (Bass et al., 2003; Buschmann et al., 1996; Shaw y Clements, 2006a).

El objetivo último que se persigue con la presente tesis doctoral es conseguir que los distintos componentes que forman estos sistemas sean fácilmente sustituibles e intercambiables, teniendo en cuenta que el uso del término *fácilmente* no implica que sea una sustitución de tipo *plug and play*, pero sí poco costosa. Así, por ejemplo, se espera poder utilizar distintas estrategias de tutoría, diferentes maneras de calcular trayectorias dentro del EV o tutores inteligentes que estén dotados de características cada vez más similares a las humanas.

Con este objetivo en mente, y considerando las definiciones de arquitectura software que se han visto hasta el momento, resulta de especial utilidad diseñar la arquitectura que tienen que tener estas aplicaciones, de manera

que se identifiquen claramente sus subsistemas y la comunicación entre ellos para, en último término, poder sustituirlos y minimizar el impacto de estas modificaciones tanto en la arquitectura como en el sistema resultante.

3.4.2. ¿Cómo se Evalúa la Arquitectura?

Como se ha podido ver en la sección 2.5, no resulta fácil medir y evaluar las características de una arquitectura software. Esto es debido a varios factores:

- no existe una identificación clara de las características que debe tener una arquitectura software
- no hay una definición unánime de los atributos de calidad de las mismas
- determinados atributos de calidad se solapan con otros
- la mejora de unos atributos de calidad interfiere con otros
- hay factores externos a la arquitectura que también interfieren en las medidas

Sin embargo, lo que sí parece cierto es que las características con las que se trabaja para evaluar las arquitecturas se refieren a la calidad del software, entendiendo por calidad la *adecuación al uso previsto para el sistema software* (IEEE, 2002). Algunos de los atributos de calidad más habituales son (Bass et al., 2003):

- rendimiento: capacidad de ejecución; está relacionado con la velocidad de ejecución y el tiempo de respuesta.
- flexibilidad: también llamada modificabilidad, es la facilidad para incorporar cambios.
- fiabilidad: tolerancia a fallos, es decir, respuesta correcta del sistema ante situaciones erróneas o inesperadas.
- seguridad: resistencia a distintos tipos de ataques, como denegación de servicio o *spoofing*.
- disponibilidad: capacidad de servir peticiones.
- usabilidad: facilidad de uso.

Aunque todos los atributos de calidad mencionados son importantes de cara al resultado final, ya que todos ellos influyen en la calidad final del sistema, las características de los EVETIs, al no ser sistemas críticos, hacen que fiabilidad, seguridad y disponibilidad sean características que no resultan esenciales para los objetivos del presente trabajo. Por otra parte, la usabilidad, mucho más relacionada con la interacción del usuario con el EV, se escapa del ámbito de la presente tesis doctoral, aunque la funcionalidad propia de los SITs para adaptar la tutoría a las características de los estudiantes ya incluye aspectos relacionados con este atributo de calidad en la arquitectura.

Sí son características más relevantes, en cambio, la **modificabilidad**, pues es el objetivo principal que queremos conseguir, y el **rendimiento**, pues presenta interacciones con los restantes atributos de calidad y puede verse afectado al intentar aumentar la modificabilidad, constituyendo además uno de los aspectos clave para la consecución del éxito en el desarrollo de sistemas de entrenamiento basados en realidad virtual.

Para las dos características de interés, se pueden identificar los estímulos externos que las afectan y la manera de medir su impacto. En el caso de la modificabilidad, el estímulo externo serían los cambios en el sistema, y la forma de medir el impacto sería a través del coste del cambio. Para el caso del rendimiento, los estímulos serían los eventos desde el exterior, y la manera de medir el impacto sería por la velocidad de respuesta. Sin embargo, estas medidas sólo se pueden tomar cuando el diseño se encuentra en un estado avanzado o con el sistema ya implementado, y existen cuestiones adicionales que se deben tener en cuenta.

El análisis y evaluación de la arquitectura puede iniciarse en etapas tempranas del diseño de la misma, siendo los resultados tanto más fiables cuanto mayor detalle tenga ésta. Además, no es práctico producir análisis detallados sobre los resultados de las medidas de los atributos de calidad. En su lugar, resulta más productivo identificar tendencias (Kazman et al., 2000) y puntos sensibles.

Los atributos de calidad, por sí mismos, no son suficientes a la hora de realizar un diseño o de evaluarlo, sino que será necesario concretarlos a las características del proyecto en desarrollo (Barbacci et al., 2003). Por ejemplo, en el caso de la modificabilidad, se puede discutir que un sistema sea modificable si permite cambiar con facilidad la interfaz de usuario pero es dependiente del sistema operativo. La respuesta dependerá del tipo de modificaciones que se espera que experimente el sistema a lo largo de su existencia. Por ello, resulta necesario describir situaciones más específicas, junto con la respuesta esperada, a través de la utilización de escenarios, como por ejemplo:

> *Se desarrolla un planificador que permite la incorporación de operaciones aritméticas sencillas, y el sistema permite la sustitución del antiguo por el nuevo en menos de dos semanas·hombre.*

Como se puede observar, este escenario define claramente un estímulo (la existencia de un nuevo planificador que se quiere incorporar) y la respuesta del sistema (sustitución en menos de dos semanas·hombre), lo cual da un criterio claro para poder realizar una evaluación.

Por tanto, para estudiar la calidad de la arquitectura, no se van a utilizar medidas numéricas sobre su modificabilidad, ya que estas técnicas de evaluación aún están poco desarrolladas y ofrecen una información limitada sobre la calidad de la arquitectura resultante.

Por contra, lo que se propone es la realización de una evaluación de la arquitectura software del sistema utilizando ATAM (Clements et al., 2002b) y tomando como atributos de calidad fundamentales la modificabilidad y el rendimiento. El resultado será un informe que refleje si la arquitectura diseñada cumple con los requisitos de modificabilidad y rendimiento fijados inicialmente, así como una lista de puntos sensibles que pueden ser origen de conflictos cuando se produzcan modificaciones.

3.4.3. ¿Cómo se Utiliza la Arquitectura?

Detrás de los objetivos de este trabajo se encuentra la intención de proporcionar herramientas que sean de utilidad a la hora de desarrollar EVEs basados en agentes. Si bien, como ya se ha dicho, la elaboración de una arquitectura para estos sistemas puede constituir una herramienta valiosa allí donde las metodologías de desarrollo no ofrecen ninguna indicación, también es cierto que una arquitectura, por sí sola, constituye una ayuda que se queda coja si carece de un conjunto de indicaciones que muestren cómo utilizarla y cómo es posible hacerla evolucionar.

Por este motivo se ha pensado que, como aportación adicional de este trabajo, resulta conveniente elaborar un conjunto de recomendaciones metodológicas que sirvan de ayuda a quien pretenda utilizar la arquitectura resultante, ya sea para implementar un Entorno Virtual de Entrenamiento a partir de ella o para realizar modificaciones sobre la misma y adaptarla a sus propias necesidades.

Estas recomendaciones metodológicas se deben interpretar como criterios a utilizar para modificar la arquitectura propuesta, lo cual permitirá realizar modificaciones sobre ella conservando la integridad conceptual de la misma. De esta manera se conseguirá que la arquitectura no se degrade por efecto de los sucesivos cambios que se realicen sobre ella, lo cual facilitará su

mantenimiento y prolongará su existencia sin necesidad de tener que rediseñarla por completo.

3.5. Objetivos del Trabajo

Como se ha podido ver, existen algunas carencias fundamentales en el desarrollo actual de Entornos Virtuales con Tutoría Inteligente (EVETI) que impiden que estos sistemas puedan alcanzar su madurez.

En primer lugar, la inexistencia de arquitecturas adecuadas provoca que los EVETIs no sean lo suficientemente modificables. Por poner un ejemplo, se están realizando notables avances para dotar a los tutores virtuales de rasgos propios de los humanos, como mecanismos de comunicación utilizando lenguaje natural o arquitecturas cognitivas para proporcionarles rasgos de personalidad y estados de ánimo, así como distintos mecanismos de percepción para que sepan lo que pasa a su alrededor. Sin embargo, todavía resulta complejo dotar a los sistemas existentes de estos nuevos avances, ya sea implementándolos desde cero o añadiendo módulos ya implementados.

En segundo lugar, la falta de estándares para el desarrollo de estos sistemas dificulta enormemente el intercambio no sólo de tutores inteligentes y de EVs, sino de los elementos que los componen. Así, resulta bastante complejo adoptar distintas estrategias de tutoría, mecanismos de planificación y cálculo de rutas o utilizar dispositivos de interacción de distinta naturaleza de los que fueron diseñados para funcionar inicialmente con el EV.

Con el presente trabajo de tesis doctoral se pretende definir una arquitectura software para EVETIs que facilite la modificabilidad de los mismos y la reutilización de sus componentes. Para ello, es necesario tener en cuenta qué elementos pueden formar parte de un EVETI y cuáles de estos interesa que puedan ser sujetos a modificación, ya sea ampliación, sustitución o incluso eliminación. Entre estos elementos deben encontrarse:

- Entorno Gráfico: es uno de los elementos cuya sustitución se debe poder llevar a cabo con facilidad. Debido a la complejidad de su desarrollo, resulta difícil de implementar para equipos no especializados en ello. Por este motivo, disponer de un EV externo que pueda integrarse con el software propio podría dar como resultado un sistema mejor construido.

- Módulo de Tutoría: al trabajar con sistemas de enseñanza, uno de los elementos que es previsible que evolucionen con frecuencia son precisamente los tutores, ya sea incorporando nuevas estrategias de tutoría o haciendo uso de nuevos elementos para valorar las aptitudes de un alumno.

- Modelo del Estudiante: está muy relacionado con el anterior, ya que cuanto más rico y completo sea el modelo que se realiza del estudiante, mayores y mejores herramientas tendrá el tutor para trabajar con los alumnos. Por esta razón, también es probable que sufra ampliaciones a medida que se avance en el estudio de las actitudes y aptitudes de los estudiantes.

- Módulo Experto: maneja el conocimiento que posee el SIT sobre el dominio en el cual se realiza el entrenamiento. Resulta, por tanto, necesario para ofrecer explicaciones acerca de las actividades a realizar, para decidir la adecuación de las acciones llevadas a cabo por los estudiantes y para contestar sus preguntas.

- Planificador: al trabajar con EVEs para tareas procedimentales, el papel de un planificador resulta básico para poder plantear ejercicios al alumno, responder sus preguntas o estudiar las soluciones que los alumnos dan a los ejercicios propuestos. Los distintos dominios de aplicación de un EVETI pueden requerir diferentes tipos de planificación, incluyendo la posibilidad de plantear tareas paralelas, realizar operaciones aritméticas o planificar hacia delante o hacia atrás. También puede ser conveniente incorporar algoritmos de planificación más eficientes o utilizar planificación cooperativa. Así, éste es otro de los elementos susceptible de variar dependiendo del dominio de aplicación y de los avances en el campo de la planificación.

- Planificador de Rutas: unida a la planificación de tareas, la planificación de rutas se hace necesaria para que el sistema pueda realizar planes en los que intervengan desplazamientos dentro del EV, así como para ejercicios en los que, por ejemplo, haga falta seguir la ruta más corta entre dos puntos o aquella que cumpla con alguna otra restricción. Dependiendo de la estructura del EV utilizado y de las necesidades, es posible utilizar distintos métodos de cálculo de rutas, como el algoritmo de Dijkstra (Dijkstra, 1959), el A* (Nilsson, 1971) o diagramas de Voronoi (Voronoi, 1907), por mencionar algunos. Por tanto, también es necesario que este elemento pueda ser sustituido con facilidad.

- Simulación: el control de la simulación se requiere cuando es necesario reproducir el comportamiento de algún elemento presente en el EV tras la acción de un estudiante. Así, por ejemplo, teclear una clave de acceso puede requerir la comprobación de la corrección de la clave y la anotación del suceso en el sistema simulado; pulsar el botón de encendido de una lavadora hace necesario reproducir el funcionamiento

de la misma; el estado de un paciente varía con el transcurso del tiempo, independientemente de las acciones del estudiante o como resultado de ellas. Atendiendo al dominio de aplicación y las características de lo que se quiera simular será necesario disponer de distintos mecanismos de simulación, basados, por ejemplo, en reglas o en autómatas finitos. Por tanto, es otro de los elementos susceptibles de ser sustituido con relativa frecuencia, en función de las necesidades del dominio de entrenamiento.

Para lograr que los elementos anteriores se puedan sustituir fácilmente, resulta fundamental que la arquitectura software del sistema sea lo suficientemente flexible como para permitir el intercambio de agentes, así como la presencia o ausencia de algún agente en función de las necesidades de tutoría de una determinada actividad.

También resultará de utilidad la existencia de criterios que guíen el desarrollo de sistemas con características análogas al considerado, tanto para su diseño como para la posterior implementación. Por ello, se elaborará un conjunto de recomendaciones metodológicas que den soporte a la utilización y modificación de la arquitectura propuesta.

El cumplimiento de los objetivos expuestos a lo largo de esta sección se llevará a cabo mediante las aportaciones científicas y tecnológicas que se enumeran a continuación:

- Aportaciones científicas:
  - Arquitectura software basada en agentes para Entornos Virtuales de Entrenamiento con Tutoría Inteligente que permita sustituir con facilidad los siguientes elementos: entorno gráfico; módulo de tutoría; modelo del estudiante; módulo experto; planificador; planificador de rutas; y simulador.
  - Recomendaciones metodológicas para la construcción de Entornos Virtuales de Entrenamiento con Tutoría Inteligente utilizando la arquitectura basada en agentes descrita anteriormente.
- Aportaciones tecnológicas:
  - Implementación de una infraestructura básica que utilice la arquitectura diseñada y que sirva como punto de partida para poder implementar EVETIs a partir de ella.

Capítulo 4

Propuesta de Arquitectura Organizacional

"Esta es la caja. El cordero que quieres está dentro"
— El Principito

4.1. Introducción

Como se ha mencionado en capítulos anteriores, uno de los principales objetivos de este trabajo de tesis es diseñar una arquitectura software para Entornos Virtuales de Entrenamiento con Tutoría Inteligente (EVETI).

En el presente capítulo se describe, sin entrar en excesivos detalles, una propuesta inicial de arquitectura con estructura jerárquica, obtenida a partir de la descomposición de los elementos que conforman la arquitectura tradicional de un Sistema Inteligente de Tutoría (SIT) en otros elementos más especializados.

Como se podrá ver al final del capítulo, la arquitectura resultante no presenta las propiedades esperadas respecto a la facilidad de modificación que se deseaba obtener, y presenta otra serie de características que la hacen poco adecuada para los objetivos propuestos.

Pese a ello, constituye una experiencia previa que resulta de gran utilidad para comprender las motivaciones y criterios que han conducido al diseño de la arquitectura que se presenta en el capítulo siguiente, motivo por el cual se ha decidido incluirla como un capítulo del trabajo. No se describe en detalle dado que no constituye una aportación definitiva para los objetivos de esta tesis.

Es necesario aclarar también que la arquitectura descrita en este capítulo se diseñó en una etapa bastante temprana de la realización de este trabajo,

y los resultados obtenidos resaltaron la necesidad de buscar soluciones más allá del campo de la ingeniería de software orientada a agentes. Así fue como la investigación derivó hacia el área de las arquitecturas software, y por tanto conviene dejar claro que el contenido de este capítulo no hace uso de todo el conocimiento que se ha adquirido posteriormente de ese área.

Para diseñar esta arquitectura se utilizó un enfoque organizacional propuesto por diversas metodologías de desarrollo de software orientado a agentes, siendo la metodología Gaia (Zambonelli et al., 2003) el referente principal para el diseño de la arquitectura que se describe a continuación.

Gaia parte de una estructura inicial sobre la que se trabaja para obtener el resultado final. Esta estructura inicial es la de la organización (*organizational metaphor*) para la que se desarrolla el sistema. En el caso que nos ocupa, se ha decidido adoptar como estructura de partida la de un Sistema Inteligente de Tutoría tradicional (Sleeman y Brown, 1982; Wenger, 1987), debido a la inexistencia de otra organización que tomar como referencia y también a la experiencia previa del autor con estos sistemas.

Por esta razón, se comenzará presentando una visión de los Sistemas Inteligentes de Tutoría, para continuar desarrollando la arquitectura de un EVETI a partir de la de un SIT, terminando con el diseño de una arquitectura conceptual del sistema (Hofmeister et al., 2000).

4.2. Arquitectura de Alto Nivel

Con la aparición de la Inteligencia Artificial (IA) a principios de los 60 y su aplicación al desarrollo de sistemas de enseñanza se acuñó el término *Sistema Inteligente de Tutoría* (SIT). Un SIT (Sleeman y Brown, 1982) tiene como objetivo actuar como un tutor humano inteligente, de manera que sea capaz de orientar y enseñar a un estudiante en el proceso de aprendizaje de una materia dada, detectar los errores que el estudiante comete, tratar de determinar el punto en que éste falla para corregirlo y aclarar confusiones o dudas que se le presenten, considerando las peculiaridades de cada estudiante y permitiendo, de esta forma, una enseñanza más individualizada.

Para poder alcanzar dicho objetivo, durante el desarrollo de un sistema de este tipo deben abordarse cuestiones como la representación en el sistema de la materia objeto de estudio, la representación del conocimiento acerca del estudiante, la estrategia de comunicación con el estudiante, y las estrategias de tutoría a seguir. La arquitectura más general de un SIT (Wenger, 1987), que puede verse en la Figura 4.1, comprende cuatro componentes principales, cada uno de los cuales viene a dar respuesta a una de las cuestiones anteriores:

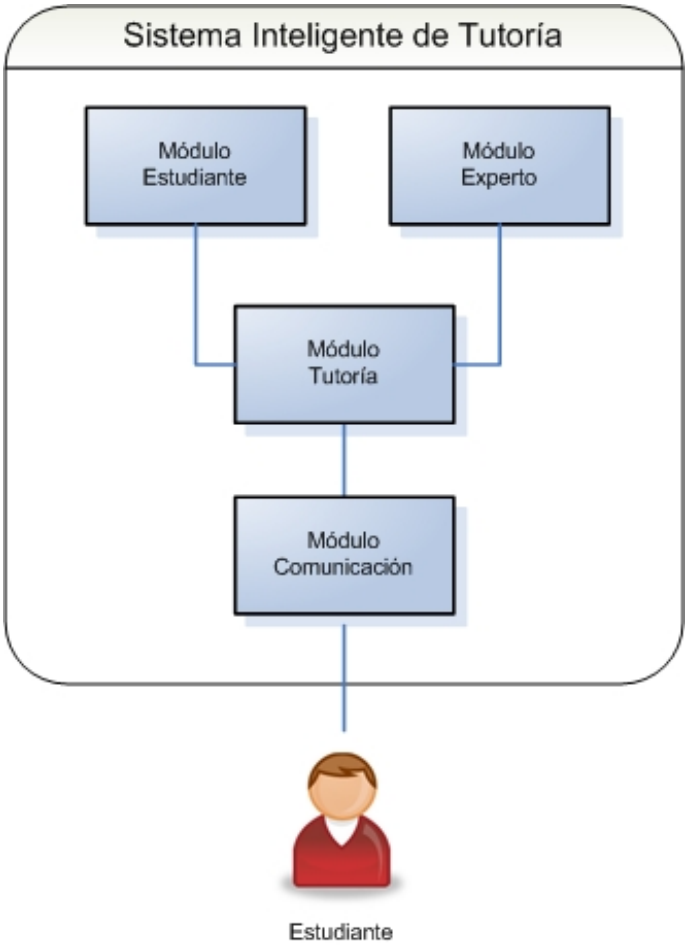

Figura 4.1: Arquitectura clásica de un SIT

- Módulo Experto: contiene el conocimiento que el estudiante debe adquirir, y sirve para comprobar el grado de corrección de las acciones y respuestas del estudiante.

- Modelo del Estudiante: contiene toda la información que el SIT posee acerca del estudiante: el estado de sus conocimientos actuales, la naturaleza de sus problemas de aprendizaje o su historial desde que empezó a utilizar el sistema. Se emplea para predecir el nivel de comprensión del estudiante y reconocer su forma particular de aprendizaje.

- Módulo de Comunicación: soporta la interfaz con el estudiante, proporcionando salidas de la forma más adecuada para cada estudiante

e interpretando sus respuestas según las clasificaciones a las que el módulo de tutoría es sensible.

- Módulo de Tutoría: contiene las estrategias, reglas y procesos que dirigen las interacciones del sistema con el estudiante, y que permiten al SIT tomar decisiones acerca del material que debe presentar, qué preguntas o ejemplos sugerir y por qué y cuándo se debe interrumpir al estudiante. Se basa en la información proporcionada por el modelo del estudiante.

La manera más clásica de construir un SIT utilizando esta arquitectura se puede asimilar con una típica arquitectura en tres capas, donde el Módulo de Comunicación hace las veces de interfaz de usuario, el Módulo de Tutoría contiene la lógica de la aplicación y los Módulos Experto y del Estudiante serían sendas bases de datos, una con el conocimiento del experto y otra con la información que se posee sobre el estudiante.

Esta arquitectura, aunque suficiente para un SIT tradicional, no se ajusta a los requisitos de un EVETI por varios motivos:

- Un EVETI debe servir para poder entrenar a un grupo de estudiantes al mismo tiempo, y no a uno solo, y debe ser capaz de adaptar el entrenamiento a las necesidades de cada alumno, lo cual no está contemplado en la arquitectura de la Figura 4.1.

- El Módulo del Estudiante debe modelar el conocimiento de cada estudiante, pero también el del grupo completo.

- El estudiante no queda fuera de los límites del sistema, ya que la interacción con el EV se realiza a través de un avatar que se encuentra dentro del propio EV y que está controlado con dispositivos de Realidad Virtual como guantes o sensores. Además, cada estudiante tiene una visión distinta del entorno dependiendo del punto en el que se encuentre.

- Aunque sería muy intuitivo considerar que el EV forma parte de la interfaz gráfica de usuario, existe una razón que impide hacer esta consideración. Cuando se utiliza una interfaz gráfica clásica, la interacción con la aplicación se realiza a través de menús, botones y controles similares, pero el fin no es aprender a utilizar esos controles, sino que son un medio para comunicarse con la aplicación. Por contra, cuando se utiliza un EV, el fin que se persigue sí es aprender a manipular lo que aparece dentro de él. Por este motivo, es necesario que el SIT

tenga conocimiento de lo que ocurre dentro del EV, su estado y las posibilidades de interacción.

Teniendo en cuenta las diferencias anteriores y la experiencia previa del autor (ver sección 2.10), si se desea que un SIT interactúe con un EV multiusuario es necesario extender la arquitectura clásica para incorporar las nuevas características que de él se requieren. Los cambios que es necesario llevar a cabo son:

- Ampliar el Módulo de Comunicación para que dé soporte a varios estudiantes de manera simultánea.
- Ampliar el Módulo del Estudiante para que mantenga un modelo para cada estudiante, así como del grupo completo.
- Ampliar el Módulo Experto para incluir el conocimiento de lo que ocurre en la simulación cuando el estudiante realiza alguna acción.
- Incluir un nuevo módulo, el Módulo del Mundo, para mantener información geométrica y semántica acerca del estado de los objetos y habitantes del EV. Este nuevo módulo debe comunicarse con el de Tutoría, para que éste a su vez pueda tener acceso al estado del mundo para, por ejemplo, indicar dónde se ubica un objeto que el estudiante no es capaz de encontrar, y con el de Comunicación, para poder actualizar el estado del mundo en respuesta a las acciones de los estudiantes.

Con estas modificaciones, la arquitectura resultante es la que se muestra en la Figura 4.2.

4.3. Arquitectura Conceptual con Agentes

Aunque para el desarrollo de la arquitectura conceptual de un sistema software no es necesario especificar la forma en que se construirá (e.g. utilizando objetos o agentes), en este punto se considera adecuado y necesario introducir los agentes en el diseño arquitectónico. Esto se debe a que sus características particulares pueden obligar a la modificación de la arquitectura si su utilización se deja para más adelante.

Para el diseño de la arquitectura, los distintos autores ofrecen diferentes alternativas, que acaban reduciéndose a dos junto con las distintas variantes que pueden surgir entre ellas.

En (Zambonelli et al., 2003), a través de la *metáfora de la organización*, se sugiere tanto una estructura jerárquica como una de pares, en la que

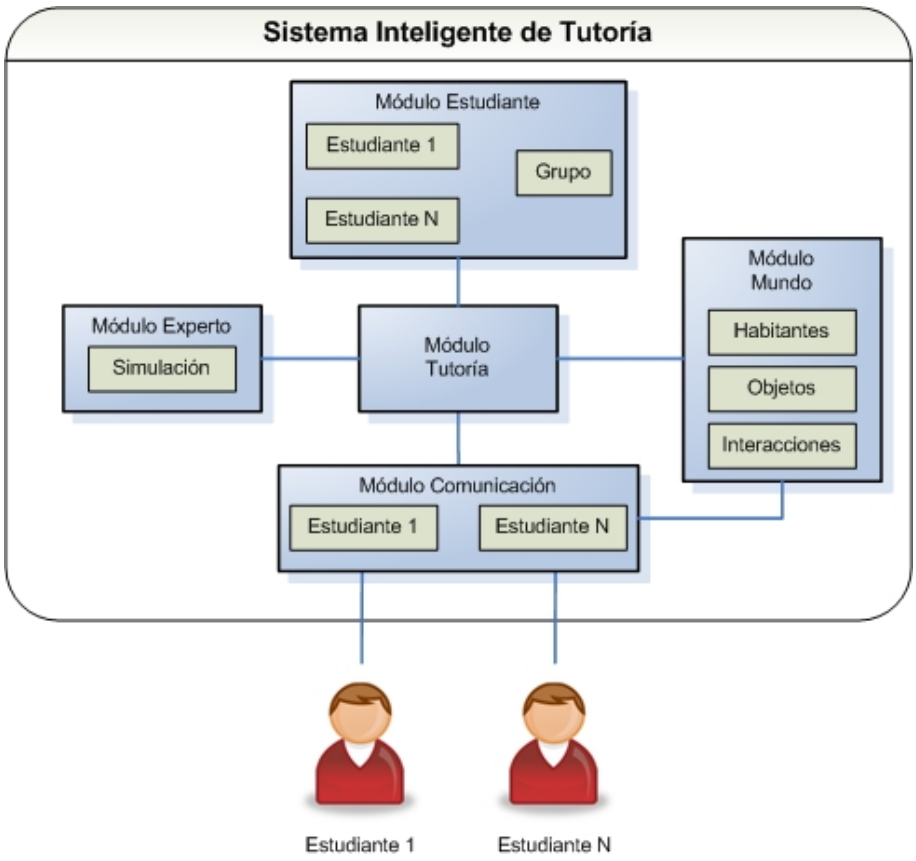

Figura 4.2: Arquitectura ampliada de un SIT

todos los componentes se sitúan al mismo nivel. En (Bass et al., 2001) parecen decantarse por el modelo jerárquico, a partir de una descomposición y refinamiento iterativo de cada elemento de la arquitectura, de manera casi independiente al resto de elementos.

En vista de lo expuesto por ambos autores, y dada la clara separación de responsabilidades de los módulos que forman un SIT, se decidió adoptar un enfoque jerárquico para el diseño de la arquitectura.

4.3.1. Agentes de Primer Nivel

Estos agentes, como se puede ver en la Figura 4.3, se derivan directamente de la arquitectura de alto nivel, y corresponden a los módulos del SIT más el Módulo Mundo, que queda representado por el Agente Mundo.

Las responsabilidades de los agentes son las que se describen a continuación.

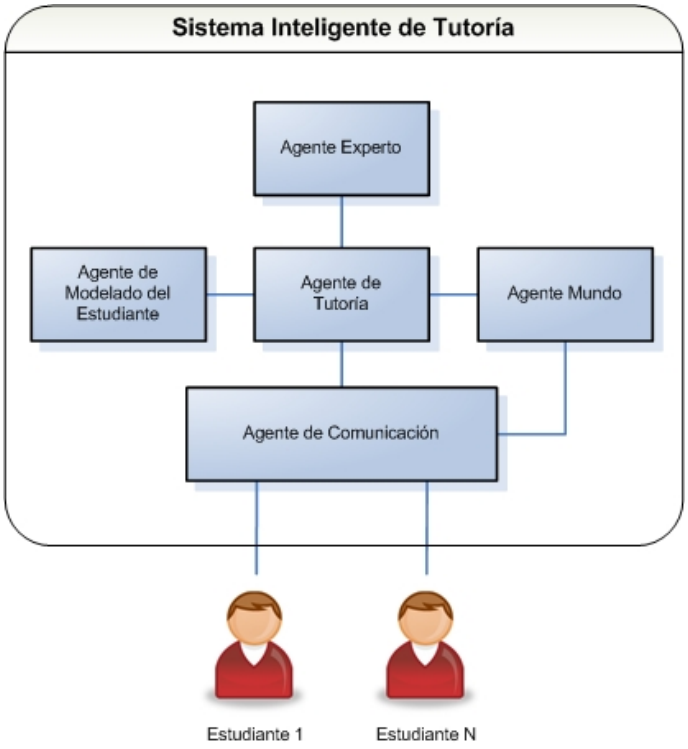

Figura 4.3: Arquitectura ampliada de un SIT con agentes

4.3.1.1. Agente de Comunicación

El agente de comunicación es el encargado de gestionar todo el intercambio de información existente entre el sistema y el exterior.

De esta manera, de cara al exterior del SIT, debe encargarse de recibir las acciones que realizan los estudiantes en el EV y cualquier otro evento que pueda suceder en él y transmitírselos, según corresponda, al agente de tutoría y al agente mundo.

También debe gestionar las conexiones de los distintos EVs para poder informar al agente de tutoría del número de estudiantes conectados.

Finalmente, debe enviar las respuestas del SIT al EV en forma de acciones, explicaciones, respuestas o cualquier otro elemento que pueda ser tratado en el EV.

De cara al SIT, como se ha mencionado, es el encargado de distribuir los eventos del EV al agente correspondiente, ya sea al agente de tutoría, si son acciones o preguntas del estudiante, o al agente mundo, si son cambios en la configuración de los objetos o habitantes presentes en el EV.

En sentido contrario, debe recibir las instrucciones que el agente de tutoría desea comunicar a los estudiantes y enviarlas al alumno correspondiente.

4.3.1.2. Agente de Tutoría

Este agente constituye la pieza central del SIT, y es responsable de muchas de las decisiones que afectan al proceso de enseñanza. Así, en última instancia, este agente es el que decide qué se hace ante las distintas acciones que realicen los estudiantes: dejarles continuar cuando realizan alguna acción incorrecta, darles algún tipo de pista o cambiar el nivel de detalle de las explicaciones y las respuestas a sus preguntas. Esto se realiza adoptando diferentes estrategias de tutoría, estrategias que se pueden apoyar en la información que se posee sobre los alumnos y que está en posesión del agente de modelado del estudiante.

La comunicación entre este agente y los que forman el resto del sistema es bastante amplia, debido a la relevancia de su papel.

Del agente de comunicación recibe las acciones y preguntas de los estudiantes, y le devuelve respuestas a las preguntas, pistas, explicaciones, y eventos que deben suceder en el EV en respuesta a las acciones de los estudiantes.

Del agente experto recibe la forma de resolver las actividades que sirven de entrenamiento al alumno, así como el conocimiento que se le debe transmitir al alumno en forma de respuestas a algunas preguntas, explicaciones y pistas. Se le proporcionan las acciones que realiza el alumno para poder actualizar la solución al ejercicio planteado.

4.3.1.3. Agente de Modelado del Estudiante

El agente de modelado del estudiante es, posiblemente, el agente más complejo de encajar en la arquitectura, porque su función puede variar enormemente tanto en tamaño como en complejidad. En la versión más simple, no hace más que recibir las acciones que realiza el estudiante desde el agente de tutoría y las almacena para mantener una traza de cada sesión. En la versión más compleja, sirve de punto de apoyo de la estrategia de tutoría, de manera que realiza un diagnóstico cognitivo del alumno: mantiene un perfil de su estilo de aprendizaje, su atención, estado de ánimo o el conocimiento que el SIT asume que el estudiante posee, en función de los cuales se pueden seleccionar distintas estrategias de tutoría y distintas respuestas ante una acción del alumno, así como el nivel de detalle de una explicación.

Recibe del agente de tutoría la información necesaria para mantener el modelo del estudiante, como las acciones realizadas, sus movimientos por el

EV o hacia dónde mira, y le proporciona los datos necesarios para tomar decisiones de tutoría, como su nivel o facilidad de aprendizaje.

Además, puesto que posee la información relativa a todos los estudiantes, también debe realizar un modelo del grupo en su conjunto, de manera que se pueda medir su capacidad organizativa, su coordinación o la corrección de la distribución de funciones.

4.3.1.4. Agente Experto

Este agente posee el conocimiento sobre la materia que se quiere enseñar al alumno. En el caso del entrenamiento procedimental, es capaz de plantear situaciones que el alumno debe resolver y de actualizarlas en función de las acciones del estudiante para replantear la solución en caso necesario. Por tanto, debe tener capacidades de planificación y replanificación, que serán aprovechadas por el agente de tutoría para utilizar estrategias más o menos flexibles. Así, se podrán plantear ejercicios con objetivos similares partiendo de situaciones distintas o, incluso, plantear ejercicios en respuesta a una petición del alumno.

Por otro lado, posee el conocimiento necesario para simular el comportamiento de los objetos que aparecen en el EV, de forma que si se requiere la manipulación por parte del estudiante se pueda controlar el comportamiento de los mismos.

Proporciona al agente de tutoría el planteamiento de los ejercicios a resolver, así como la información para determinar si una acción es o no correcta. De él recibe peticiones para crear un nuevo ejercicio, las acciones realizadas por el estudiante para que las valide y peticiones para replantear la solución de un ejercicio si las acciones del estudiante se dirigen por un camino distinto al propuesto por el experto.

4.3.1.5. Agente Mundo

Este agente se encarga de mantener una representación del EV que puedan utilizar los demás agentes. Posee información acerca de los objetos presentes en el EV, como su posición, su orientación y su geometría. También posee conocimiento sobre la utilidad de los distintos objetos, como las acciones que se pueden realizar con ellos y sus condiciones de uso.

Además, maneja información sobre los habitantes del EV, como su posición, la orientación de su mirada o los objetos que poseen.

Finalmente, también conoce la estructura del EV, de manera que es capaz de calcular rutas entre dos puntos distintos del mismo, evitando colisionar con los objetos que se encuentran presentes en él.

Recibe toda la información geométrica a través del agente de comunicación, y le proporciona al agente de tutoría la información necesaria acerca del estado del mundo para que la pueda utilizar en el proceso de tutoría.

4.3.2. Agentes de Segundo Nivel

Estos agentes surgen para hacerse cargo de las responsabilidades de los agentes descritos en la sección anterior, quienes se encargarán de coordinarlos y de canalizar las comunicaciones entre ellos. Así, la existencia de los agentes de primer nivel introduce un nivel de indirección que contribuye a aumentar la flexibilidad de la arquitectura (Bass et al., 2000), ya que facilita la introducción de nuevos agentes subordinados a ellos y la eliminación de alguno de los existentes, atendiendo a la funcionalidad requerida.

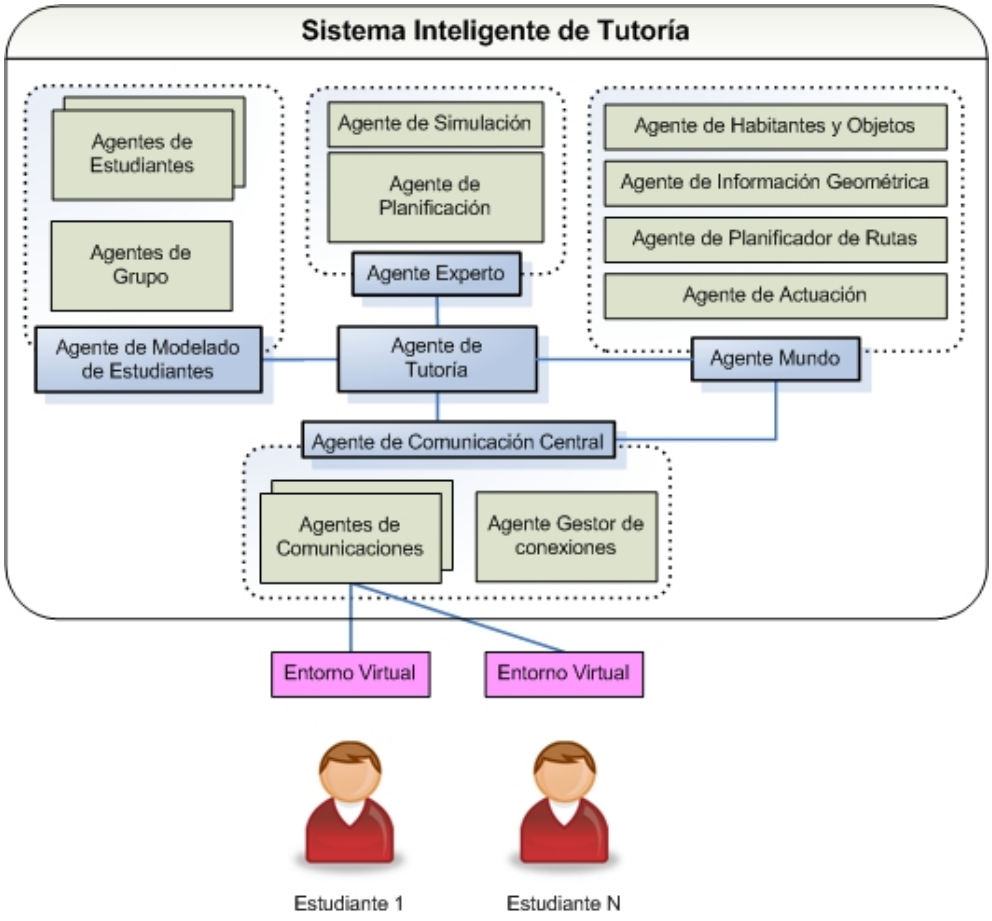

Figura 4.4: Arquitectura basada en agentes

Por otra parte, los agentes de segundo nivel se ocuparán de las responsabilidades de carácter variable y para las cuales se pretende dotar de flexibilidad a la arquitectura.

Además, se ha adoptado un enfoque cooperativo para realizar la planificación y replanificación de las soluciones a los ejercicios. Dicho enfoque permite la incorporación de nuevos agentes con conocimiento distinto al de los agentes existentes, lo cual facilita la adaptación a nuevos dominios. Todos los agentes que participen en la planificación se van a comunicar por medio de una pizarra, lo cual también contribuye a la modificabilidad del sistema, ya que la pizarra permite la incorporación y eliminación de agentes sin que el resto deba sufrir ningún tipo de modificación.

A continuación se describen las responsabilidades de cada uno de los agentes de segundo nivel, así como las de los agentes de primer nivel, tanto las nuevas como las que ya poseían.

4.3.2.1. Agente de Comunicación Central

Este agente es el equivalente al agente de comunicación del nivel anterior. Sin embargo, al añadir como subordinados a los agentes de comunicación y al agente gestor de conexiones, este agente se mantiene sólo por motivos de flexibilidad. Sus responsabilidades quedan prácticamente reducidas a cero, salvo por las recién adquiridas como mediador.

Su papel principal en la arquitectura es, por tanto, el de mantener la modificabilidad. Esto permitirá, en caso de que sea necesario, eliminar los agentes de comunicación sin afectar al resto de los agentes, lo cual puede ser conveniente, por ejemplo, para mantener el SIT funcionando sin EVs conectados a él. También facilitará la inclusión de nuevos agentes, lo que puede ser útil si se precisa manejar algún tipo de información especial relativa, por ejemplo, a los dispositivos de realidad virtual, ya que esta información podría no ser proporcionada directamente por el EV.

Aunque puede constituir un cuello de botella, el hecho de que no realice ningún tipo de procesamiento complejo, puesto que sólo redirige mensajes entre el exterior y el interior del SIT, hace poco probable que vaya a generar problemas de rendimiento.

4.3.2.2. Agentes de Comunicación

Estos agentes se encargan de la comunicación entre el SIT y cada uno de los EVs. En ejecución habrá tantos agentes de comunicación como estudiantes hayan iniciado una sesión de entrenamiento, y cada agente será responsable de recibir la información de su EV correspondiente, traducirla al formato que

maneje el resto de los agentes y enviársela al agente de comunicación central para que éste, a su vez, se la envíe al agente adecuado.

La labor de traducción se le asigna a estos agentes por dos motivos. El primero es que de esa forma se puede realizar simultáneamente la traducción de los mensajes enviados desde distintos EVs, lo cual redunda en un mejor rendimiento de la aplicación (aunque el hecho de tener que hacer la traducción es ya una carga para el SIT). El segundo es que permite cambiar el formato de los mensajes entre el EV y el SIT sin necesidad de modificar ningún otro agente del SIT, lo cual va en beneficio de la modificabilidad del sistema. Por supuesto, sería mejor que la traducción se realizase en el EV, antes de enviar la información, pero ya que uno de los propósitos es poder interconectar diferentes EVs, no se puede asegurar que esto siempre vaya a ser así.

De igual manera, estos agentes se deben encargar de realizar la labor contraria: recibir información del agente de comunicación central, traducirla y enviarla al EV del cual son responsables.

4.3.2.3. Agente Gestor de Conexiones

La labor de este agente es, sobre todo, de soporte. Así, es el encargado de recibir las conexiones entrantes de cada alumno que desea unirse al entrenamiento y crear un agente de comunicación que se encargue del intercambio de información entre el nuevo EV y el SIT.

También se encarga de informar al agente de tutoría, a través del agente de comunicación central, del número de estudiantes que se encuentran conectados, para que se pueda iniciar el entrenamiento cuando haya suficientes estudiantes conectados.

4.3.2.4. Agente de Tutoría

El agente de tutoría es el único que, de momento, no tiene asignado ningún agente subordinado, por lo que conserva todas las responsabilidades que se le asignaron en la sección anterior.

Aunque se ha barajado la posibilidad de incorporar un agente de tutoría individualizado para cada estudiante, esto podría provocar problemas de coordinación y visualización, por lo que, de momento, se ha preferido mantener un único agente de tutoría común para todo el grupo.

También se ha estudiado la posibilidad de incluir agentes que pongan en práctica distintas estrategias de tutoría. Sin embargo, esto presenta varios problemas que no aconsejan seguir ese camino.

A pesar de su aparente utilidad desde el punto de vista pedagógico, requeriría establecer un proceso para decidir qué estrategia de tutoría aplicar

en cada momento. Este proceso de negociación entre distintos agentes de estrategia de tutoría podría conducir a un cambio continuo de estrategia de tutoría, lo cual podría llegar a confundir al alumno.

Desde el punto de vista computacional, la negociación para realizar el cambio de estrategia puede suponer un coste alto si se realiza con demasiada frecuencia. Además, dependiendo de las estrategias de tutoría que deban convivir, puede suponer un incremento significativo de la complejidad del agente de modelado de estudiantes.

En cualquier caso, la arquitectura permitiría dotar al sistema de ambos mecanismos.

4.3.2.5. Agente de Modelado de Estudiantes

Al igual que sucedía con el agente de comunicación central, el agente de modelado de estudiantes corresponde al agente de estudiante descrito en la sección anterior, cuyas responsabilidades se han delegado en sus agentes subordinados, por lo que su papel ahora es el de mediador por motivos de flexibilidad.

Así, se encargará de redirigir la información y las peticiones que le transmita el agente de tutoría al agente adecuado, y enviar las respuestas de estos agentes al de tutoría. En caso necesario deberá encargarse de realizar labores de traducción.

4.3.2.6. Agentes de Estudiante

De igual manera que los agentes de comunicación, los agentes de estudiante se van a encargar de realizar un modelo de cada estudiante conectado al sistema. En ejecución existirá un agente de estudiante por cada alumno conectado al sistema.

El modelo de estudiante puede incluir información acerca de la forma de aprendizaje del alumno, de su personalidad o de su atención. También incluirá las acciones realizadas por el alumno a lo largo de la sesión, así como indicaciones de su corrección o incorrección y cualquier otro tipo de evaluación que pueda ser de interés para el desarrollo de la estrategia de tutoría seleccionada.

La evaluación de la atención del alumno puede necesitar información en poder del agente mundo, como la dirección de la mirada del estudiante. De igual manera, el diagnóstico de por qué el estudiante ha cometido un error necesita información en posesión del agente experto. Por tanto, puede existir flujo de información entre los agentes de estudiante y el agente experto y el agente mundo.

Por el momento se ha decidido que este tráfico de información se realice a través del agente de tutoría, pues estas acciones se realizarán a petición suya. Si llegase a suponer un tráfico excesivo de información y un cuello de botella, habrá que realizar una comunicación directa. Esto supondría una merma en la modificabilidad del sistema, al tiempo que un aumento en el rendimiento de la aplicación, ya que la comunicación no pasaría por intermediarios.

De manera adicional, la evaluación podría realizarse sin esperar la petición del agente de tutoría, por lo que éste tampoco tendría que esperar a que sea elaborada la información que necesite y el rendimiento de la aplicación también se vería incrementado.

4.3.2.7. Agente de Grupo

Este agente es el encargado de realizar la misma labor que los agentes descritos anteriormente pero para el grupo en su conjunto. Al contrario que los anteriores, sin embargo, no utilizará la información proveniente del exterior, sino que hará uso de la que ya ha sido elaborada por los agentes de estudiante.

El modelo a realizar por este agente incluye aspectos de coordinación entre los miembros del grupo, colaboración entre ellos o posibles fallos en algún aspecto concreto del trabajo en equipo.

4.3.2.8. Agente Experto

El agente experto es otro de los que han visto sus responsabilidades reducidas a un papel mediador, en este caso entre el agente de tutoría y el de planificación. Al contrario que en el caso del agente de comunicación central o el de modelado de estudiante, carece de un grupo de agentes a su cargo, por lo que su carga de trabajo no es muy grande, ni en cantidad ni en complejidad. Aunque es un firme candidato a desaparecer de la arquitectura, de momento se ha mantenido por motivos de homogeneidad.

4.3.2.9. Agente de Simulación

El agente de simulación está a cargo del conocimiento del dominio relacionado con los efectos de las acciones del usuario sobre algunos objetos. En concreto, tal como indica su nombre, está encargado de simular el comportamiento de determinados objetos del EV, especialmente cuando un alumno actúa sobre ellos. Por ejemplo, si se quisiese enseñar a un estudiante el procedimiento a seguir para lavar la ropa en una lavadora, el agente de simulación estaría encargado de simular lo que sucede en la lavadora una

vez que ésta se pone en marcha: los distintos estados por los que pasa y los efectos que tienen sobre la ropa.

Este agente es uno de los mejores ejemplos de la necesidad de modificabilidad en la arquitectura, ya que dependiendo del dominio de aplicación puede suceder que este agente no sea necesario, funcione con un sistema basado en reglas, con un autómata finito determinista o por cualquier otro medio.

4.3.2.10. Agente de Planificación

El agente de planificación constituye unos de los puntos críticos de la arquitectura, y es uno de los principales motivos por los que es deseable que el sistema sea altamente modificable.

Este agente tiene uno de los papeles más relevantes dentro de un EVETI, ya que es responsabilidad suya plantear la solución a un ejercicio e ir encontrando soluciones alternativas a medida que las acciones de los estudiantes no se ajustan a las especificadas en el plan inicial.

El motivo de la importancia de este agente es doble. Por un lado, el proceso de planificación y replanificación es bastante laborioso, y constituye uno de los elementos que más afecta al rendimiento del sistema. Por otro lado, el hecho de haber optado por una planificación cooperativa hace que haya más de un agente involucrado en la misma, lo cual puede afectar a la modificabilidad de la arquitectura. Esto es debido tanto a que, dependiendo del dominio, pueden ser necesarias distintas técnicas de planificación y distinto tipo de representación de la información, y también a que la necesidad de comunicación puede dar lugar a acoplamientos no deseados.

Para evitar el problema que supone la planificación y replanificación la solución consiste en utilizar planificadores eficientes en los que la replanificación no suponga una nueva planificación desde cero. Este es uno de los motivos que hace que sea necesario facilitar el cambio de planificador. El otro, como se ha mencionado anteriormente, es el hecho de que distintos dominios de aplicación pueden necesitar distintos tipos de planificación.

Para evitar el acoplamiento entre los distintos agentes que toman parte en la planificación se ha optado por realizar la comunicación a través de una pizarra. De esta forma, el agente de planificación se encarga de realizar la planificación con la información que le proporcionan el resto de los agentes, pero no necesita saber cuántos ni cuáles son. Por otro lado, el resto de agentes que toman parte en la planificación no necesitan saber quién la realiza ni quién más participa. Simplemente, si encuentran en la pizarra alguna situación que se puede satisfacer con el conocimiento que poseen, lo escriben en la pizarra y se olvidan de cualquier otro detalle.

En cuanto al problema de representación de la información, no es algo que se pueda resolver con la arquitectura del sistema, sino que dependerá de una buena implementación de la misma. Por ejemplo, si se implementase utilizando orientación a objetos, utilizar operadores polimórficos ayudaría a evitar este problema.

En la arquitectura que se plantea, la planificación la realiza el agente de planificación junto con el de simulación, el de actuación y el planificador de rutas. Con este enfoque resultaría sencillo hacer que otros agentes también participasen en la planificación.

4.3.2.11. Agente Mundo

Como ocurre con la mayoría de los agentes descritos en la sección anterior el agente mundo queda reducido a labores de intermediario para establecer un punto que dote a la arquitectura de mayor flexibilidad. Al contrario de lo que sucedía con el agente experto, este agente tiene la suficiente entidad como para que no surjan dudas acerca de su necesidad, ya que este agente sí tiene como misión la supervisión de un grupo de agentes.

Por otra parte, el agente mundo es susceptible de recibir nuevos agentes subordinados a él, ya que los avances en modelos de percepción, personalidad y comportamiento para agentes inteligentes hacen bastante probable que haya que incorporar nuevas características al funcionamiento del tutor virtual dentro del mundo.

Este agente va a interactuar, por tanto, con el agente de comunicación central para recibir las actualizaciones sobre el estado del EV, las cuales trasladará a sus agentes subordinados. También se comunica con el agente de tutoría para proporcionar la información que éste necesite para dar explicaciones al alumno o contestar sus preguntas.

4.3.2.12. Agente de Habitantes y Objetos

El agente de habitantes y objetos posee información semántica acerca de los objetos presentes en el EV, de forma que este conocimiento es utilizado para indicarle a los alumnos qué es un objeto o para qué se puede utilizar. Esta información se la proporcionará al agente de tutoría a través del agente mundo.

4.3.2.13. Agente de Información Geométrica

El agente de información geométrica mantiene un mapa geométrico del EV, incluyendo habitaciones, objetos y personajes. Aunque toda esta

información podría recuperarse del EV, resulta más eficiente mantener una réplica en el SIT, la cual se mantendrá actualizada a medida que lleguen cambios procedentes de cualquiera de los EVs que se encuentren conectados al SIT.

Este agente maneja información acerca de la situación y orientación de todos los habitantes y objetos presentes en el EV, así como de los objetos que se encuentran en posesión de cada uno de los habitantes del EV. De esta manera, por ejemplo, proporciona información acerca de dónde se encuentra un objeto en el EV para poder indicárselo al alumno.

4.3.2.14. Agente de Actuación

El agente de actuación es uno de los agentes más importantes de cara al proceso de planificación. Este agente maneja información acerca de las posibles acciones que se pueden realizar con los objetos presentes en el EV, de manera que, con la configuración actual, es el agente que más contribuye a la planificación.

Por otro lado, la información que maneja este agente es de utilidad para indicarle al alumno las consecuencias de realizar una determinada acción dentro del EV.

4.3.2.15. Agente Planificador de Rutas

El agente planificador de rutas se encarga de calcular los posibles caminos que existen para llegar de un punto a otro del EV, evitando chocar con obstáculos. Este cálculo se realiza, fundamentalmente, aplicando un algoritmo de cálculo de trayectorias como A* al mapa del EV realizado por el agente de información geométrica.

Los caminos calculados por este agente se pueden utilizar, por ejemplo, para mover agentes virtuales dentro del EV o, en los casos en los que sea aplicable, para comparar la ruta obtenida con la que ha seguido el estudiante, de manera que se pueda ver si ha seguido el camino más conveniente o no.

4.4. Análisis de la Arquitectura

Como se ha mencionado al comienzo del capítulo, la arquitectura aquí descrita presenta una serie de problemas que la hacen poco adecuada para los fines que se persiguen. En concreto, algunas partes de la arquitectura son excesivamente rígidas para la modificabilidad que se quiere obtener.

En otros casos, las decisiones que se han tomado, aunque razonables en su momento, parecen no conducir hacia los fines deseados.

A continuación se describen los dos principales motivos que han llevado a descartar esta arquitectura, junto con unas reflexiones del autor del trabajo sobre las posibles causas de haber tomado decisiones poco adecuadas.

4.4.1. Efectos de la Planificación en la Arquitectura

Una de las responsabilidades más complejas que soporta la arquitectura descrita en este capítulo es la relativa a la planificación. Uno de los requisitos que se impusieron es que se deseaba que el sistema de planificación implementado fuese muy flexible y admitiese la adición de nuevos operadores de manera dinámica. Por esa razón se decidió utilizar una planificación cooperativa en la que distintos agentes pudiesen aportar operadores relativos a lo que exclusivamente ellos saben hacer. Esto permite la introducción de nuevos agentes, incluso de manera dinámica, que aporten nuevos operadores que ayuden a realizar un plan donde antes no se podía, o a realizar un plan más eficiente que el que se conseguiría sin su ayuda.

En este sentido, se debe decir que la arquitectura presenta un alto grado de modificabilidad, ya que permite la introducción de nuevos agentes y nuevos operadores para realizar la planificación de manera dinámica, sin modificar el resto del sistema.

Sin embargo, esta característica que incluye tanta flexibilidad impone también fuertes restricciones. En primer lugar, involucra a muchos de los agentes del sistema en la planificación. Como cada agente es responsable de sus propios operadores, los cambios que se realicen en el planificador que afecten el formato de los operadores pueden afectar a todos los agentes que formen parte del proceso de planificación.

Por otra parte, el hecho de realizar la planificación usando una pizarra que utilicen todos los agentes se salta en cierta manera la filosofía de utilizar una estructura jerárquica en la que la comunicación entre agentes se realice a través de los agentes supervisores. Uno de los aspectos fundamentales en una arquitectura es que exista la denominada *integridad conceptual* (Bass et al., 2003), que consiste en mantener una homogeneidad de criterios en el diseño de la arquitectura. El hecho de introducir soluciones distintas para cada una de las funciones que debe cumplir la arquitectura redunda en una mayor complejidad de la misma y, a la larga, en una mayor dificultad para entenderla, manejarla y modificarla.

Además, uno de los factores que no se tuvo en cuenta es que en algún momento podría ser necesario sustituir el tipo de planificador empleado en la realización de la arquitectura. El planificador utilizado en primera instancia,

basado en STRIPS, resultó ser bastante sencillo, y no proporcionaba algunas capacidades que se hacía necesario utilizar. Llegado el momento, el problema que surgió es que no fue posible encontrar un planificador que admitiera seguir utilizando la planificación cooperativa como se hacía hasta el momento. El resultado fue que, al cambiar el planificador, hubo que desmontar gran parte del sistema para volver a rediseñarlo, lo cual demostró que la arquitectura no estaba preparada para sufrir esa modificación.

4.4.2. La Estructura Jerárquica

Todavía es muy común hoy en día la realización de diseños mediante la descomposición de un sistema en subsistemas, lo que en muchos casos acaba con un diseño que presenta una estructura jerárquica. En ocasiones, como sucede con la metodología utilizada, esta estructura jerárquica es el objetivo que se busca desde el principio.

Puesto que es lo que recomendaba la metodología seguida, así se hizo, obteniéndose una arquitectura jerárquica con dos niveles. Esta arquitectura, como se ha señalado en el apartado anterior, contaba con un mecanismo alternativo para realizar el proceso de planificación.

En ese punto surgen dos problemas. El primero de ellos es que parte de la comunicación entre los agentes tiene lugar para realizar la planificación, de manera que en cierto modo, no se respeta uno de los criterios que llevan a utilizar la estructura jerárquica. El segundo es que a veces resulta difícil discriminar qué mensajes forman parte del proceso de planificación y cuáles no. Como consecuencia, a veces por comodidad, se tiende a comunicar a los agentes directamente en lugar de respetar la estructura jerárquica.

Esto último no es sino una muestra de que, aunque en apariencia la estructura jerárquica independiza a los agentes subordinados a un agente supervisor de los agentes subordinados a otro, la dependencia se mantiene si existía y no se ha tomado alguna medida para eliminarla, más allá de la descomposición jerárquica.

La interpretación que se hace de esto es que, aunque la arquitectura tradicional de un SIT separa las funciones de tutoría de las del experto o de las de modelado del estudiante, lo que no hace es eliminar la dependencia que existe entre estos módulos. Así, lo que sucede al descomponerlos es que la dependencia se traslada a los módulos que se originan tras la descomposición, pero no se elimina.

La descomposición jerárquica ha originado también otros problemas. Casi cualquier descomposición ofrece resultados distintos en función del criterio que se use para descomponer. Se da también el caso en que un mismo criterio de descomposición puede justificar que un elemento cuelgue indistintamente

de dos o más padres, puesto que puede no caer completamente en el ámbito de ninguno de ellos, y en general las descomposiciones jerárquicas no contemplan la posibilidad de que un submódulo lo sea a la vez de dos módulos distintos.

Es lo que ha pasado, por ejemplo, con el agente de habitantes y objetos, el cual se ha dejado bajo la responsabilidad del Agente Mundo, aunque durante algún tiempo estuvo orbitando alrededor del Agente Experto. Quedó al final bajo la responsabilidad del Agente Mundo porque parecía que tenía algo más que ver con él, pero no hay una frontera clara a este respecto.

Esto saca a la luz otro de los inconvenientes de las descomposiciones jerárquicas, y es que la descomposición se puede hacer cuando se crea el sistema desde cero. Las sucesivas ampliaciones ya no se pueden realizar por descomposición, sino por adición, por lo que nada asegura que el criterio de descomposición vaya a seguir resultando válido.

Como consecuencia surge un nuevo problema: si añadir un nuevo elemento nos obliga a realizar una nueva descomposición, se pueden dar dos situaciones. La primera es que se decida rediseñar toda la arquitectura, lo cual puede tener un coste bastante elevado. La segunda es que se intente introducir el nuevo elemento en la arquitectura existente de la mejor manera posible, lo cual conducirá, casi con toda seguridad, a un proceso de degradación de la arquitectura que acaba llegando al mismo punto que el caso anterior: la necesidad de un nuevo diseño de la arquitectura software del sistema.

4.5. Reflexiones

Existe una tendencia natural a intentar los diseños, más que a evaluarlos, para emitir dictámenes acerca de si son buenos o malos.

A pesar de los defectos que se han podido discernir en la sección anterior, con el análisis realizado no se quiere transmitir la impresión de que la arquitectura es mala, sino que no cumple con todos los requisitos de modificabilidad que se persiguen.

La arquitectura descrita es buena, si se entiende como buena el hecho de que es modificable, aunque sea sólo en unos determinados aspectos, y desde luego también es buena porque dio lugar a un sistema implementado que ofrecía las funcionalidades que de él se requerían. Por lo tanto, lo más que se puede decir es si la arquitectura es modificable en uno u otro aspecto, y en alguno de ellos no lo era tanto como se esperaba.

Pensando en el proceso de diseño de la arquitectura seguido en este capítulo, y sobre todo después de haber obtenido un resultado poco satisfactorio, cabe plantearse si tiene sentido comenzar a diseñar la arquitectura dirigiéndola ya hacia una estructura jerárquica. Intentando

llegar más allá, puesto que el resultado con la arquitectura jerárquica no ha sido muy satisfactorio, cabe plantearse si se debe fijar de antemano la estructura que debe tener la arquitectura antes de diseñarla.

A ojos del autor, la respuesta debe ser no. Sin embargo, lo que ha sucedido en este caso es justo lo contrario, es decir, se ha diseñado la arquitectura, siguiendo las directrices de una metodología, intentando obtener un sistema jerárquico, sin considerar, al menos no mucho, otras opciones. Esto no quiere decir que se atribuya a la metodología el mal resultado obtenido, sino más bien que se han asumido ciertas características sin cuestionarlas hasta que han aparecido los primeros síntomas de que algo no funciona.

Cuenta el premio Nobel de física Richard Feynman (Feynman, 2007) que algunas de sus investigaciones sufrieron contratiempos por aceptar como válidas las afirmaciones de otros expertos sin cuestionarlas o comprobarlas.

Salvando humildemente las distancias con tan insigne personaje, ese mismo problema se puede haber presentado en este caso. Se ha asumido que la estructura jerárquica funciona porque los expertos dicen que funciona, y por ese motivo se ha diseñado una estructura jerárquica, sin cuestionar si efectivamente esos expertos estaban considerando las mismas cuestiones que se tienen en cuenta en este trabajo.

Está muy arraigada la costumbre de realizar diseños jerárquicos, por refinamientos sucesivos. Se nos enseña en las universidades como técnica de programación, y se mantiene en muchos casos como técnica de diseño. Sin embargo, como dice Haythorn (Haythorn, 1994), una cosa es la programación de algoritmos y otra el diseño de sistemas, y lo que puede funcionar en un caso no tiene por qué hacerlo en otro. Lo que falla en muchos casos no es el hecho de descomponer en sí, sino la forma en que lo hacemos, es decir, el criterio de descomposición, como también señala Parnas (Parnas, 1972).

Además, quizá se ha caído en el error de asumir como madura un área como el desarrollo de sistemas orientados a agentes que no tiene tantos años de existencia. Por ello, se ha confiado en la validez de las propuestas del área del desarrollo orientado a agentes sin tener en cuenta otras áreas de la ingeniería de software tradicional, cayendo en algunos de los errores mencionados en (Wooldridge y Jennings, 1998).

La principal conclusión que se saca de la experiencia mostrada en este capítulo es la necesidad de buscar las herramientas que no se han encontrado en el área de agentes en el lugar donde es más probable que se encuentren: las arquitecturas software.

Capítulo 5

Propuesta de Arquitectura Centrada en la Modificabilidad

"Nada es, todo transcurre. Nada perdura, todo cambia"
— Heráclito

5.1. Introducción

El diseño de la arquitectura de un sistema software constituye un aspecto fundamental en el desarrollo del mismo, tanto para conseguir que la aplicación resultante funcione como para lograr que lo haga sin problemas y satisfaciendo los requisitos impuestos sobre ella, tanto funcionales como no funcionales.

Como se ha puesto de manifiesto en el capítulo anterior, para obtener el diseño de la arquitectura no es suficiente con seguir una serie de pasos propuestos por una metodología concreta, ni con situar estratégicamente una serie de módulos de una manera que parezca adecuada.

Para solucionar el problema que se plantea, más que en el área de las metodologías orientadas al diseño de sistemas multiagente, se pueden encontrar técnicas de mayor utilidad en el área de las arquitecturas software. Si bien este área no cuenta con demasiados años de existencia, y tiene una presencia escasa en el área de los sistemas multiagente, no deja de ser cierto que se ha desarrollado un importante trabajo que se basa en la experiencia acumulada desde principios de los años 70, lo que le confiere bastante solidez.

Uno de los mayores problemas de la arquitectura mostrada en el capítulo anterior ha sido dar por sentado que una estructura jerárquica sería suficiente para construir un sistema modular fácilmente modificable.

Llegados a este punto, y dispuestos a modificar o rediseñar la arquitectura anterior, resulta conveniente plantearse si es adecuado seguir utilizando una estructura jerárquica o si, por el contrario, resulta más acertado adoptar un enfoque distinto que, además, saque mayor partido a las características aportadas por los agentes.

Tomando como base el hecho de que la aplicación del *principio de ocultación de información* (Parnas, 1972) será una de las técnicas que proporcione al sistema la modificabilidad buscada, el propio Parnas afirma que la descomposición obtenida de aplicar el principio a un diseño no tiene por qué resultar en una estructura jerárquica. No obstante, sí manifiesta en (Parnas et al., 1984) que esta estructura jerárquica, si existe, resulta de utilidad a la hora de organizar la documentación del diseño, ya que aporta mayor claridad a la hora de ir desvelando al lector los detalles del mismo.

Buscando más allá de los límites de la informática se puede encontrar un influyente trabajo de Herbert A. Simon (Simon, 1969) donde defiende la estructura jerárquica como manera fundamental de organizar sistemas complejos. Sin embargo, un reciente análisis de este trabajo (Agre, 2003) sitúa la concepción de Simon de los sistemas complejos en su marco histórico, y lo muestra como consecuencia natural del pensamiento de la época, poniendo de manifiesto la imposibilidad que existía entonces de alcanzar soluciones hoy en día posibles, como los sistemas auto-organizados (p.e. los mercados), que también son aplicables al campo de la arquitectura de sistemas software. Además, este último artículo afirma que los sistemas auto-organizados son una evolución que necesita de la existencia previa de sistemas jerárquicos para desarrollarse, situación que se cumple en el presente trabajo.

Ésta va a ser la perspectiva que se trate en el presente capítulo, donde se mostrará la arquitectura de la aplicación como un sistema auto-organizado que obtiene un mayor partido de las capacidades de computación distribuida de los agentes y del modelo de organización *peer-to-peer*.

Además, se hará uso de técnicas propuestas por distintos autores, como (Bass et al., 2003), (Álvaro García y Medinilla, 2007), (Hayden et al., 1999), (Buschmann et al., 1996) orientadas a conseguir que los diseños posean mayor modificabilidad.

5.2. Proceso de Diseño de la Arquitectura

Para realizar un desarrollo más formal de la arquitectura del sistema se va a utilizar el conjunto de métodos y técnicas proporcionados por el grupo de arquitecturas software del SEI (*Software Engineering Institute, Carnegie Mellon University*), surgidos en torno al año 1995.

Como principal herramienta para el diseño de la arquitectura software del sistema proponen el método ADD (*Attribute Driven Design*) (Bass et al., 2001; Wojcik et al., 2006). ADD es un método iterativo que se basa en la sucesiva descomposición de los distintos módulos del sistema hasta obtener una arquitectura que permita conseguir los objetivos funcionales y de calidad deseados. Para descomponer cada módulo es necesario seleccionar los escenarios de calidad que le afectan, y descomponerlo utilizando las tácticas y estilos arquitectónicos más indicados para conseguir los atributos de calidad perseguidos, como los definidos en (Bachmann et al., 2007). Donde sea preciso, también se utilizarán patrones de diseño orientados a agentes, como los definidos en (Hayden et al., 1999), ya que algunos de ellos están especialmente orientados a dotar a la arquitectura de mayor flexibilidad. Será necesario definir las interfaces de los módulos resultantes y asignarles responsabilidades para, en caso de ser necesario, proseguir con la descomposición.

ADD está pensado para desarrollar software de corte tradicional, pero no existe literatura que documente su uso para el desarrollo de sistemas multiagente. Por ello, no puede asegurarse su buen funcionamiento para este tipo de software. De hecho, se espera que su aplicación no esté exenta de dificultades en lo que al diseño del SIT se refiere. Sin embargo, el énfasis que se hace en los atributos de calidad de la arquitectura lo convierten en una herramienta valiosa para conseguir una arquitectura con las características de modificabilidad y rendimiento deseadas.

El uso de ADD se complementa con QAW (*Quality Attribute Workshop*) (Barbacci et al., 2003). Básicamente, QAW consiste en la realización de una reunión en la que participan las partes interesadas en el desarrollo de la aplicación, con el objeto de identificar los atributos de calidad y generar los escenarios de calidad. A través de distintas fases, se generan, refinan y priorizan una serie de escenarios que identifican los atributos de calidad deseados para el sistema. QAW no es un método sistemático y objetivo, por lo que los resultados dependen de los participantes en las sesiones. Una vez obtenidos los escenarios de calidad con QAW, se utilizarán como entrada para el diseño de la arquitectura con ADD (Nord et al., 2004).

De los distintos métodos existentes para evaluar arquitecturas software, SAAM (*Software Architecture Analysis Method*), ARID (*Active Reviews for Intermediate Designs*) y ATAM (*Architecture Trade-off Analysis Method*), se ha decidido utilizar este último por tres razones, principalmente. La primera es que ARID es más apropiado para estadios intermedios de la arquitectura, lo cual lo hace menos apropiado para nuestros intereses. La segunda es que ATAM es un método más moderno y evolucionado que SAAM, al cual podría decirse que engloba, aunque SAAM es más específico para evaluar

modificabilidad y eficiencia. La tercera razón es que existen precedentes de evaluación de arquitecturas basadas en agentes utilizando ATAM (Woods y Barbacci, 1999), (Ivezic et al., 2000), (Boucké et al., 2006) lo cual no sucede con los otros dos métodos.

Al igual que QAW, ATAM se basa en la generación de una serie de escenarios que, en este caso, se utilizarán para evaluar la arquitectura. Los escenarios son tanto funcionales como específicos para evaluar los atributos de calidad y, al igual que sucedía con QAW, no sirven para dar una medida objetiva de la calidad de la arquitectura, sino que reflejan los intereses de los evaluadores en el momento de realizar la evaluación. Las salidas del método, por tanto, no son medidas absolutas de calidad, sino una lista de puntos críticos de la arquitectura, un marco de acción para evaluar la arquitectura y una lista de problemas no resueltos.

Por los motivos anteriormente expuestos, los pasos a seguir para el diseño de la arquitectura y su posterior evaluación serán los siguientes:

- Utilización de QAW para generar los escenarios de los atributos de calidad que interesa contemplar.
- Utilización de ADD para definir la arquitectura software.
- Cuando finalice la aplicación de ADD, se evaluará la arquitectura en función de los criterios enumerados en (Wooldridge y Jennings, 1999), de manera que se cuente con unos criterios de evaluación específicos para sistemas basados en agentes.
- Una vez finalizada la descripción de la arquitectura, se evaluará utilizando ATAM, para refinarla en caso necesario.

5.3. Atributos de Calidad

De los seis atributos de calidad que se mencionan en (Bass et al., 2003) como relevantes para el diseño de la arquitectura de un sistema software, hay dos, modificabilidad y eficiencia, que han sido identificados por las personas involucradas en el desarrollo de la presente arquitectura como de especial importancia.

La modificabilidad es el atributo que cobra más importancia, ya que el principal objetivo que se persigue es el desarrollo de una arquitectura que facilite el intercambio de determinados componentes de la misma.

La eficiencia en la ejecución también es de especial importancia, ya que una de las características deseables de un EV es que pueda actuarse dentro

de él de forma similar a como se haría en un entorno real, lo cual implica una velocidad de ejecución lo más cercana posible al tiempo real.

Los otros cuatro atributos de calidad, seguridad, disponibilidad, usabilidad y testabilidad, no deben descartarse, si bien no suponen, en el momento en que se aborda el diseño de la arquitectura, factores que deban tenerse en cuenta a la hora de darle forma a la misma.

De los cuatro, y teniendo en cuenta el tipo de sistema con que se trabaja, podría argumentarse que la usabilidad también debería tenerse en cuenta. Sin embargo, funcionalidades como la posibilidad de cancelar o deshacer una acción entran más en el ámbito de la estrategia de tutoría que en el de la usabilidad, razón por la cual se ha decidido no tenerlos en cuenta como atributos que afecten a la arquitectura del sistema. Otros factores que afectan a la usabilidad, como realizar de modelos de usuario o permitir iniciativas de la aplicación para ayudar a los usuarios, entran también dentro de las funcionalidades requeridas de un SIT, por lo que no se consideran elementos que afecten a la usabilidad del sistema.

A continuación se describen los escenarios de calidad que se han identificado como más relevantes y que se tendrán en cuenta a la hora de diseñar la arquitectura. Se utilizará la notación sugerida en (Bass et al., 2003) tal y como se muestra en la Figura 5.1.

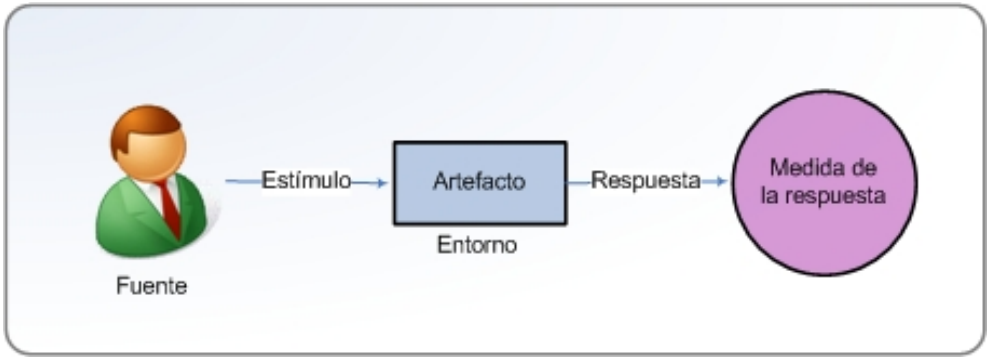

Figura 5.1: Escenario de calidad genérico

5.3.1. Escenarios de Modificabilidad

La modificabilidad del sistema debe considerarse desde dos puntos de vista. Por una parte, como ya se ha mencionado, se pretende diseñar una arquitectura que posibilite la sustitución de algunos de sus componentes por otros que proporcionen funcionalidades similares. En presencia de un estándar, debería ser posible intercambiar componentes sin mayores

consecuencias. No siendo así, es de esperar que sea necesario adaptar las interfaces de comunicación y el formato de los datos que intercambian los distintos módulos, aspecto que se refleja en el escenario de la Figura 5.2. Puesto que, sin más información, resulta difícil dar una medida del tiempo que llevará realizar las modificaciones, se ofrece como medida del impacto de éstas el número de módulos que se verán afectados por las mismas.

Por otra parte, también se desea que, en función de las necesidades de la persona que diseña el curso o de quien realiza el entrenamiento, haya algunas funcionalidades que se puedan añadir o quitar sin tener que modificar el diseño de la arquitectura o el código de la aplicación. Debe ser posible realizar estos cambios a través de un fichero de configuración, una opción de un menú o algún otro medio similar, razón por la cual es un mecanismo que debería poder funcionar bajo demanda sin mayores complicaciones.

Estas necesidades se reflejan en los escenarios de las Figuras 5.2 y 5.3.

Figura 5.2: Escenario general de modificabilidad en diseño

Figura 5.3: Escenario general de modificabilidad en ejecución

Los siguientes apartados describen las modificaciones concretas que se tendrán en cuenta.

5.3.1.1. Escenario de Modificabilidad 1 Sustitución del Entorno Virtual

Uno de los problemas a los que comúnmente deben hacer frente los equipos que desarrollan SITs que funcionan junto a un EV es que ningún miembro del equipo suele ser experto en el desarrollo de EVs. Los EVs desarrollados suelen adolecer de defectos como escenarios mal estructurados, algoritmos poco eficientes o modelos 3D de aspecto bastante pobre. Todo ello hace que tanto el resultado final de la aplicación como la velocidad en la ejecución hagan desmerecer la, por otro lado, excelente labor realizada por estos equipos.

Por lo tanto, una de las necesidades fundamentales en estos casos es la posibilidad de conectar EVs que han sido desarrollados por otros grupos más expertos y que, por lo general, no han sido diseñados *ex-profeso* para ser integrados con los SITs con los que van a funcionar.

Además, a medida que las tecnologías evolucionan y que las necesidades de los SITs se van incrementando, se vuelve a hacer necesario sustituir el EV que se ha utilizado hasta el momento por otro más sofisticado, por lo que ésta es una necesidad que nunca deja de estar presente.

Como resultado, es necesario que, durante el diseño de la aplicación, los diseñadores puedan integrar un EV desarrollado independientemente del SIT. Aparte de ofrecer la posibilidad de acceder a la información que sea necesaria en cada caso, lo normal es que los distintos EVs trabajen con distintos formatos de la información (por ejemplo, criterios para interpretar cuáles son los ejes x, y, z) razón por la cual será habitual que se deba realizar un trabajo de traducción de los formatos de la información al integrar el EV con el SIT.

Figura 5.4: Escenario de sustitución del Entorno Virtual

5.3.1.2. Escenario de Modificabilidad 2 Sustitución de la Estrategia de Tutoría

La estrategia de tutoría es una de las partes fundamentales del SIT y una de las que más evoluciones sufrirán a lo largo de todo el ciclo de vida, pues prácticamente cualquier mejora en los demás elementos del sistema podrá ser aprovechada para modificar la estrategia de tutoría. Por tanto, es bastante probable que cualquier cambio en otras partes del sistema puedan afectar a la estrategia de tutoría y, de forma inversa, que los cambios en la estrategia de tutoría puedan afectar su relación con otras partes del sistema.

Mientras que, como se ha dicho, la estrategia de tutoría debería aprovechar las mejoras en otras partes del sistema para mejorar la tutoría, lo deseable es que la situación no se reproduzca a la inversa, es decir, que estos cambios en la estrategia de tutoría no disparen a su vez cambios en otras partes del sistema, ya que la modificabilidad quedaría seriamente afectada.

Lo que se va a tener en cuenta, por tanto, es que los cambios en la estrategia de tutoría se realicen dentro del marco definido por las funcionalidades ofrecidas por otras partes del sistema.

Lo que se pretende es que la sustitución de la estrategia de tutoría se pueda realizar, en tiempo de diseño, sin necesidad de modificar nada más que los módulos encargados de realizar la traducción de información con los demás módulos del sistema.

Figura 5.5: Escenario de sustitución de la estrategia de tutoría

5.3.1.3. Escenario de Modificabilidad 3 Sustitución del Modelo del Estudiante

Una de las mejores formas de personalizar la tutoría, que también redunda en una mejora de la usabilidad del sistema, es la realización y utilización de un modelo del usuario. En cuanto a la tutoría, es la manera de acercarla a la forma de aprendizaje del estudiante, además de servir para evaluar los conocimientos del mismo.

Así, se puede pasar de almacenar una traza de sus acciones, como medio de evaluarlo, a utilizar esa traza para decidir cómo responder a sus preguntas, si se le ofrece ayuda de manera proactiva o para valorar si está prestando atención a una explicación o si muestra señales de aburrimiento.

Los modelos de estudiante se pueden realizar de maneras completamente distintas, como pueden ser los sistemas basados en reglas o las redes bayesianas (Russell y Norvig, 1995), en función de lo que se desee obtener con el modelado del estudiante y de la eficiencia de las operaciones que se le soliciten (p.e. una red bayesiana suele ser más eficiente que un sistema basado en reglas, pero lo normal es que el modelo del estudiante no sea tan rico en información).

Dependiendo de lo sofisticado de la estrategia de tutoría y del modelo del estudiante, este último puede variar entre ser un elemento completamente pasivo o constituir un componente que tome parte activa en la toma de decisiones de la tutoría. Por ello, un cambio en el modelo del estudiante puede ir de la mano de un cambio en la estrategia de tutoría, lo cual supone un cambio de suficiente calado como para ser objeto de una atención especial.

Por tanto, lo que aquí se considera es una sustitución del modelo del estudiante existente por otro que ofrezca una funcionalidad similar y que vaya acorde con el funcionamiento marcado por la estrategia de tutoría existente.

Figura 5.6: Escenario de sustitución del modelo del estudiante

5.3.1.4. Escenario de Modificabilidad 4 Sustitución del Planificador

Otra forma de enriquecer la tutoría consiste en poder generar ejercicios de manera dinámica y, también, en resolver ejercicios o situaciones planteados por el alumno. Esto requiere disponer de un mecanismo de resolución de problemas en lugar de una batería de problemas resueltos, lo que en el caso de tareas procedimentales se traduce la necesidad de disponer de un planificador que proporcione la secuencia en la que deben realizarse las acciones.

En el caso más sencillo, el planificador puede consistir en la asociación de un problema con su solución. A partir de ahí, se pueden requerir mayores capacidades, como poder planificar un procedimiento lineal a partir de una situación inicial y unos objetivos. Posteriormente, se podrían añadir procedimientos con varias soluciones válidas, cuya diferencia sea, por ejemplo, el orden en que se realizan las acciones. Luego, se puede necesitar que se planifiquen acciones paralelas, imponer restricciones temporales o que el planificador sea capaz de realizar operaciones aritméticas sencillas.

También puede necesitarse un cambio de planificador por motivos de eficiencia. Hay implementaciones de algoritmos de planificación que son más eficientes que otras, y un planificador que sea capaz de realizar replanificaciones, en general, también resultará más eficiente que uno que no lo sea y necesite llevar a cabo una nueva planificación desde cero.

Algunas de las modificaciones descritas no implican cambios en la comunicación entre el planificador y quien use sus servicios, salvo en el formato de la información que se intercambia. Otros cambios, como la planificación de acciones paralelas, sí pueden implicar modificaciones en el tipo de información intercambiada, por lo que se requerirá que los cambios afecten únicamente a quien solicite los servicios del planificador.

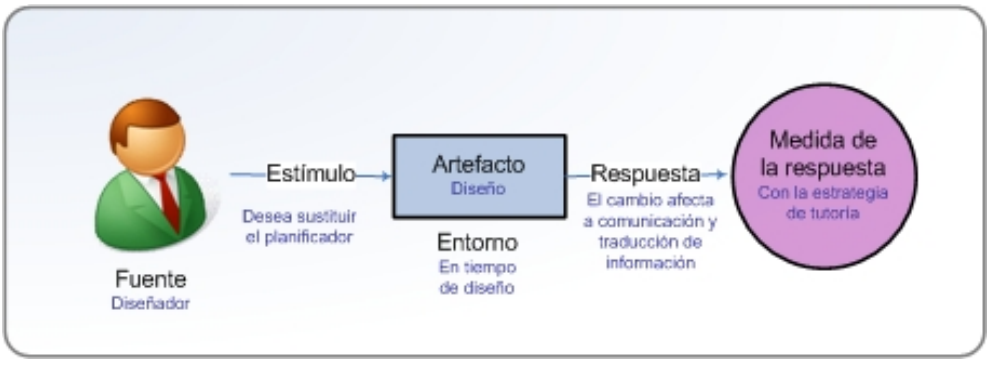

Figura 5.7: Escenario de sustitución del planificador

5.3.1.5. Escenario de Modificabilidad 5 Sustitución del Planificador de Rutas

Un planificador de rutas es un elemento que cobra especial relevancia cuando se utiliza un EV como parte integrante de un sistema de enseñanza. La utilización de un planificador de rutas sirve a varios fines, como el cálculo de un camino entre dos puntos cuando hay que mover un personaje por el escenario, o como medio para evaluar si el alumno está siguiendo la mejor ruta posible entre dos lugares dados del escenario. Esto último es especialmente relevante en determinados escenarios, como una central nuclear, donde uno de los objetivos que se le pueden exigir a un alumno es aprender a moverse por las zonas de la central en las que las radiaciones recibidas sean mínimas, lo cual no siempre implica seguir el camino más corto entre dos puntos.

Los métodos de cálculo de rutas varían en función de las características de los escenarios en los que tiene lugar el entrenamiento, especialmente en función de su tamaño y dinamismo y de las necesidades del contexto. Así, en los casos más simples, con un escenario pequeño, estático en el que siempre se busque el camino más corto, una representación del escenario en forma de grafo sobre el que se ejecuta un algoritmo de Dijkstra (Dijkstra, 1959) suele ser suficiente. En escenarios de mayor tamaño o en los que existan elementos móviles que pueden bloquear un camino que previamente era factible, suele ser necesario optar por otro tipo de algoritmos como el A* (Nilsson, 1971) o los diagramas de Voronoi (Voronoi, 1907).

En cualquiera caso, lo que se le puede pedir a un planificador de rutas es que acepte las coordenadas que maneja el SIT y que proporcione una solución de las mismas características, aunque puede ser necesario modificar la representación del escenario que usa el planificador de rutas.

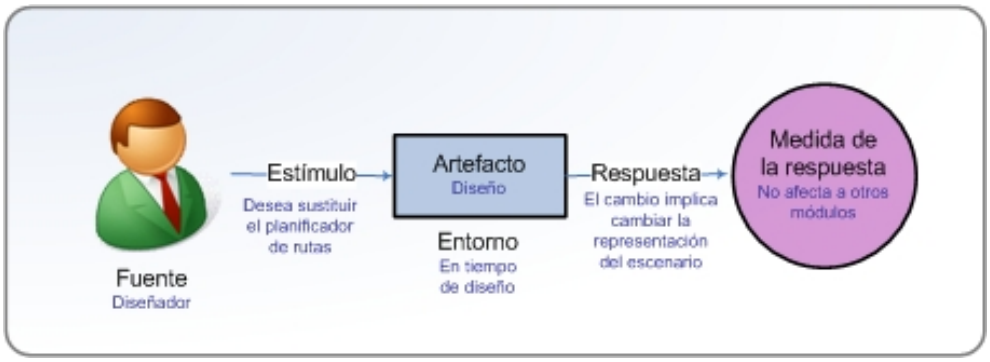

Figura 5.8: Escenario de sustitución del planificador de rutas

5.3.1.6. Escenario de Modificabilidad 6 Sustitución del Simulador

Cuando se realiza un entrenamiento en un entorno complejo, puede ser necesario hacer uso de un elemento que simule las características de dicho entorno, ya sea como respuesta a las acciones realizadas por el estudiante o como fruto de la dinámica habitual en un entorno de esas características.

Así, por ejemplo, en un entorno de entrenamiento médico, será necesario simular la evolución del estado de un paciente, quien irá reaccionando ante las acciones del estudiante, irá empeorando su estado si no se le atiende o si las medidas adoptadas no son correctas, o cambiará su estado como fruto de condiciones impuestas por el SIT para que el estudiante haga frente a determinadas situaciones.

A diferencia de lo que sucede con otros elementos descritos, el simulador es mucho más dependiente del contexto en el que se realice el entrenamiento, por lo que resulta más complicado determinar las características que puede tener un simulador o qué es lo que se puede esperar de él. Además, se puede dar el caso de que el simulador esté disponible de manera separada o que se encuentre integrado ya en un EV.

Otra dificultad añadida estriba en el hecho de que, dependiendo del estado de la simulación en cada momento, puede darse el caso de que no se puedan llevar a cabo las acciones realizadas por los estudiantes, o que la situación que determine modifique el estado del EV e influya en el proceso de planificación.

Tantas son las variables que se presentan en el caso de los simuladores que habrá que intentar reducir al mínimo el número de módulos que dependen de él, de manera que un cambio en el simulador afecte al menor número de módulos que sea posible.

Figura 5.9: Escenario de sustitución del simulador

5.3.1.7. Escenario de Modificabilidad 7 Deshabilitación de la Tutoría

Existen distintas razones que pueden llevar a que los usuarios soliciten la eliminación de las funciones de tutoría, al menos de manera visible.

El caso más claro es quizá la realización de un examen por parte de un alumno, situación en la que el diseñador del curso puede querer evaluar las aptitudes del alumno sin ningún tipo de ayuda como la que prestaría un tutor virtual. Una situación similar podría darse si existen distintos niveles de dificultad para resolver un ejercicio, donde en el nivel más sencillo se prestaría al estudiante toda la ayuda posible y en el más complejo se le dejaría con completa libertad para hacer lo que quisiese. También puede darse el caso de que, previa a la realización de una actividad, se ofrezca al alumno la posibilidad de moverse por el escenario para familiarizarse con él.

Una funcionalidad más elaborada puede ser que el SIT detecte si el alumno rechaza siempre los ofrecimientos de ayuda, lo cual podría llevarle a decidir que no va a ofrecerla más salvo que el alumno así lo requiera.

Puede suceder también que un alumno encuentre excesivamente molestas las interrupciones del tutor para hacerle preguntas o para ofrecerle ayuda, de manera que puede ser el propio alumno quien decida deshabilitar estas funciones, ya sea con este fin o para comprobar él mismo cuál es su nivel de destreza en la resolución de una determinada situación.

Para cualquiera de estas situaciones se hace necesario poder deshabilitar las funciones de tutoría, bien al arrancar la aplicación o con el sistema ya en marcha. Esto debe poder realizarse prácticamente al instante y sin mucho esfuerzo, sin necesidad de modificar el código de la aplicación sino cambiando alguna opción de configuración o solicitándolo a través de un menú o algún otro mecanismo similar.

Figura 5.10: Escenario de deshabilitación de la tutoría

5.3.1.8. Escenario de Modificabilidad 8 Deshabilitación del Cálculo de Trayectorias

La planificación o cálculo de rutas es un elemento de uso común en los EVs, ya tengan estos fines educativos o no. Sin embargo, ni todos los dominios requieren del funcionamiento del cálculo de rutas, ni todas las actividades que se puedan plantear en un dominio tienen por qué requerirlo tampoco.

Así, en un sistema de entrenamiento médico en el que el alumno tenga que realizar el diagnóstico de un paciente, lo normal es que no tenga que desplazarse demasiado y, si lo hace, es probable que esto no tenga mayor relevancia. Del mismo modo, es posible que una actividad sea o comprenda el ensamblaje de un artefacto y tenga lugar encima de una mesa. En este caso, tampoco es necesario que la planificación de rutas se encuentre activa mientras se lleva a cabo la actividad.

Por último, igual que en el apartado anterior, se le puede ofrecer al alumno la posibilidad de explorar un escenario antes de realizar una actividad, lo cual tampoco requeriría del funcionamiento del cálculo de rutas.

A la vista de las situaciones descritas, es deseable que pueda deshabilitarse el cálculo de trayectorias, especialmente en función de las características de la actividad a realizar.

Figura 5.11: Escenario de deshabilitación del cálculo de trayectorias

5.3.2. Escenarios de Eficiencia

Como se mencionaba al comienzo de la sección, en una aplicación de entrenamiento basada en realidad virtual es de especial importancia que la aplicación responda a las acciones del usuario de manera bastante inmediata. De no ser así, la experiencia del usuario puede no resultar satisfactoria y el interés por el entrenamiento se degradará de manera bastante acusada.

Sin embargo, no todas las acciones que realiza el usuario pueden medirse por el mismo rasero, y si bien es cierto que la navegación por el escenario debe realizarse de manera ágil, la respuesta a otras acciones sí admite un cierto, aunque no excesivo, retardo.

Los siguientes escenarios, que siguen la forma general mostrada en la Figura 5.12, recogen los aspectos mencionados.

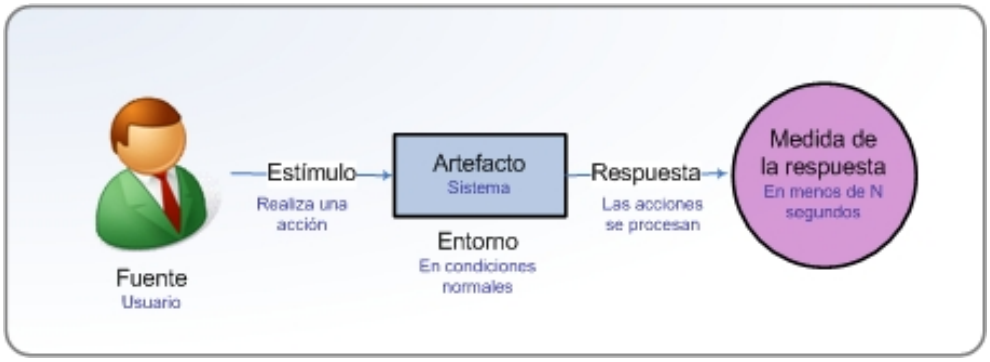

Figura 5.12: Escenario general de eficiencia

5.3.2.1. Escenario de Eficiencia 1 Respuesta ante Acciones

Cuando un alumno tiene que realizar acciones dentro del EV como parte de una actividad, éstas deben procesarse de distintas formas, ya que, por ejemplo, la estrategia de tutoría debe decidir si permite su realización, y pueden causar algún efecto en el simulador. A su vez, habitualmente se debe generar una respuesta, como no permitir su realización, advertir al estudiante de algo o disparar algún evento dentro del EV.

Como el procesamiento de una acción puede ser complejo, existe la posibilidad de que no pueda darse una respuesta en tiempo real, lo cual no es estrictamente necesario, pero el retardo debe ser lo suficientemente pequeño como para que el alumno no piense que no se ha ejecutado la acción, pues esto le llevaría a repetirla cuando en realidad está siendo procesada. Tampoco debe darle la sensación de que la ejecución de la aplicación es lenta, ya que puede desmotivarle para continuar utilizándola.

Por tanto se requerirá de la aplicación que, independientemente de lo que tenga que hacer con motivo de la ejecución de una acción, le dé una respuesta al alumno en menos de un segundo, sin importar que una misma acción pueda disparar más de una respuesta.

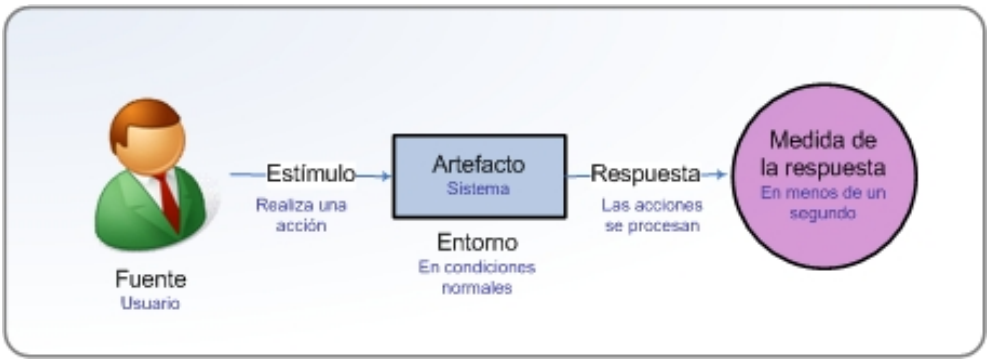

Figura 5.13: Escenario de eficiencia ante una acción

5.3.2.2. Escenario de Eficiencia 2 Respuesta en Navegación

Cuando un usuario se mueve por el EV, es necesario recopilar las coordenadas del avatar por si resulta necesario hacer un seguimiento de sus movimientos por el EV. Sin embargo, esto no debe afectar en absoluto al usuario, pues la percepción de un movimiento fluido es uno de los factores clave a la hora de que los alumnos tengan una buena experiencia con el sistema y continúen utilizándolo para su formación.

Por tanto, se requerirá que el muestreo de coordenadas se lleve a cabo sin disminuir perceptiblemente la velocidad de ejecución del sistema, de forma que el usuario no note que esta acción se está realizando.

Figura 5.14: Escenario de eficiencia en navegación

5.4. Documentación de la Arquitectura

5.4.1. Descripción de la Documentación Necesaria

Según se afirma en (Clements et al., 2002a), la documentación de una arquitectura software es un elemento fundamental para la comunicación, tanto dentro del equipo de desarrollo como con otros individuos que, de una u otra manera, se ven afectados por el desarrollo del sistema, como pueden ser los usuarios o los diseñadores de software que se puede conectar con el sistema en desarrollo. Por este motivo, resulta de especial relevancia elaborar una documentación adecuada a cada uno de los posibles afectados, lo cual implica reflejar los aspectos que necesita conocer cada uno de ellos con un nivel de detalle suficiente, pero no demasiado profundo para no dificultar la comprensión del documento.

Los autores del citado trabajo proponen la documentación de una arquitectura software a través del uso de distintas vistas de la arquitectura, las cuales reflejan los diferentes aspectos que puede ser necesario conocer. Además, proponen una serie de vistas que pueden ser utilizadas, mediante las cuales se puede documentar una amplia gama de arquitecturas. Esto no impide que el arquitecto software genere nuevas vistas o combine algunas de las existentes si lo considera necesario. A continuación se describen brevemente las vistas propuestas.

Las **vistas de módulos** reflejan aspectos estáticos de la arquitectura a través de distintos tipos de descomposiciones en módulos. Por este motivo, resultan útiles para obtener una primera visión de conjunto de la aplicación. Además, la descripción de las responsabilidades de cada módulo es útil para realizar un análisis de trazabilidad de los requisitos. También pueden resultar de utilidad para realizar análisis de impacto, de forma que ayudan a ver qué elementos de la arquitectura se pueden ver afectados por una modificación. Los estilos arquitectónicos incluidos en este tipo de vista son:

- Descomposición: representa la descomposición del sistema en subsistemas, y de éstos a su vez en otros subsistemas, y así sucesivamente. Este estilo se utiliza para dar una visión general del sistema y sus partes, y su uso está especialmente recomendado para empezar a entender el sistema y para la comunicación con los responsables del proyecto.

- Uso: muestra la dependencia de unos módulos respecto de otros en cuanto a su relación de uso. Se dice que un módulo P1 usa a otro módulo P2 cuando la corrección de P1 depende de la corrección de P2. Esto no implica que P1 llame directamente a P2, sino que la dependencia puede ser resultado de la llamada a P2 por parte de un módulo X que a

su vez es llamado por P1. Este estilo resulta de utilidad para planificar desarrollos iterativos, extensiones al sistema y análisis de impacto.

- Generalización: muestra, de una forma similar a la herencia de la orientación a objetos, qué subsistemas extienden la funcionalidad de otros, por lo que es un estilo que suele utilizarse en el diseño de arquitecturas orientadas a objetos.

- Capas: estructura los módulos del sistema en forma de una serie de capas superpuestas, de tal manera que las capas más altas pueden utilizar los servicios proporcionados por las capas más bajas de acuerdo con unas reglas predefinidas (por ejemplo, que sólo pueden utilizarse los servicios de la capa inmediatamente inferior). Es un estilo adecuado para la creación de máquinas virtuales, las cuales facilitan la reutilización y la portabilidad.

Las **vistas de componentes y conectores** reflejan el comportamiento del sistema mediante el intercambio de mensajes entre distintos componentes a través de determinados conectores. Dependiendo de la vista utilizada, los componentes puede corresponder a los módulos utilizados en las vistas de módulos, aunque no tiene que ser necesariamente de esta forma. Debido al aspecto dinámico que incorporan, estas vistas sirven para analizar atributos de calidad como fiabilidad, disponibilidad y rendimiento. Las vistas incluidas en este tipo son:

- Tuberías y filtros: es un tipo de vista que está especialmente indicado cuando existen unos datos que van a ir sufriendo sucesivas transformaciones.

- Cliente-servidor: muestra la interacción de elementos a través de la solicitud de servicios de unos a otros. Los servidores proporcionan los servicios a través de distintas interfaces, y los clientes utilizan los servicios proporcionados por esas interfaces. Este estilo desacopla a los productores y a los consumidores de datos y servicios.

- *Peer-to-peer*: se caracteriza por el intercambio de servicios entre los distintos elementos de la arquitectura; cualquier elemento puede solicitar los servicios ofrecidos por otro. En este estilo es habitual que los componentes dispongan de la información más actualizada que necesitan, lo que facilita la distribución de los componentes, reduce la carga de los mismos cuando actúan como servidores y reduce la necesidad de comunicación para actualizar datos.

- Procesos comunicados: se caracteriza por la interacción de distintos componentes que se ejecutan de manera concurrente. Sirve para comprender la dinámica de sistemas que trabajan en paralelo, por lo que también puede ser utilizado para realizar análisis de eficiencia y fiabilidad. Se suele usar en combinación con otros estilos, como *peer-to-peer* o cliente-servidor.

- Editor-suscriptor: se caracteriza por la existencia de elementos que interactúan a través del anuncio de eventos por parte de unos y la subscripción a dichos eventos por parte de otros. Es un estilo adecuado para desacoplar a los productores y a los consumidores de los eventos. Este desacoplamiento permite que los productores y los consumidores no se conozcan directamente hasta que el sistema se está ejecutando, lo que facilita la modificación tanto de productores como de consumidores. Cuando los componentes se ejecutan en distintos hilos de control, este estilo se puede considerar un refinamiento de los procesos comunicados. Suele presentarse en combinación con el estilo *peer-to-peer*.

- Datos compartidos: este estilo se centra en la representación de datos persistentes a los que acceden distintos subsistemas. Puede servir para desacoplar a los productores y a los consumidores de datos, ya que el acceso al repositorio de datos posibilita que los productores y los consumidores no se conozcan directamente. Si el almacén de datos es pasivo se le da el nombre de repositorio, mientras que si avisa a los consumidores de datos de los cambios que se producen se le da el nombre de pizarra (en este caso, se convierte en un estilo similar al editor-suscriptor, aunque en este último no se da la persistencia de los datos). Es una representación adecuada para sistemas de información y para sistemas basados en conocimiento. Este estilo suele aparecer en combinación con el cliente-servidor.

Las **vistas de asignación**, por su parte, describen la correspondencia entre elementos software y elementos del entorno:

- Despliegue: establece una correspondencia entre los elementos software de la arquitectura y los elementos hardware que soportarán su ejecución, y su utilización es especialmente adecuada para la realización de análisis de rendimiento, seguridad y fiabilidad. Es habitual que se utilice este estilo como única muestra de la arquitectura de un sistema, pero, como se ha visto hasta el momento, un sólo diagrama de este estilo carece de muchos de los detalles que es conveniente conocer sobre la arquitectura de un sistema.

- Implementación: establece la correspondencia entre los elementos software y la infraestructura de desarrollo, siendo adecuada su utilización, por ejemplo, para establecer el sistema de gestión de configuración.

- Asignación de trabajo: se utiliza para indicar a qué miembros del grupo de trabajo les corresponde el desarrollo de los distintos elementos presentes en la arquitectura.

Además de las vistas anteriores, existe **documentación adicional** que también es conveniente tener en consideración, aunque queda a criterio del arquitecto software qué es relevante y qué no lo es, en función de a quién vaya dirigida, del tipo de sistema en construcción o de los atributos de calidad considerados:

- Diagrama de contexto: dentro de la documentación de la arquitectura software, un diagrama de contexto es aquel que muestra la visión de un elemento junto con los elementos con los que éste se relaciona. Un caso de especial relevancia es aquel en el que el elemento considerado es el propio sistema. En este caso, el diagrama de contexto muestra el alcance del sistema, estableciendo qué es lo que queda dentro y qué es lo que queda fuera, así como las interacciones con el exterior. No tiene por qué existir un único diagrama de contexto con estas características, sino que puede existir uno por cada vista que proporcione una perspectiva diferente de la relación del sistema con el exterior. En cualquier caso, es recomendable que quien lea la documentación se encuentre como primer elemento de la misma con un diagrama de este estilo que le ayude a conocer poco a poco los detalles de la aplicación.

- Decisiones de diseño: puesto que el diseño de la arquitectura es un proceso iterativo, en muchas ocasiones se contemplan soluciones que posteriormente se desechan por distintos motivos. Suele ser conveniente documentar las razones por las que se ha optado por una solución, así como las soluciones descartadas y los motivos para descartarlas, de manera que en posteriores revisiones de la arquitectura se cuente con esa experiencia como criterio para seguir descartando una solución u optar por ella en función de cambios que hayan podido suceder.

- Mapeo de vistas: una de las tareas importantes a la hora de documentar la arquitectura consiste en decidir qué vistas se utilizan. Una vez decidido, es conveniente documentar la relación existente entre las distintas vistas para que no dé la sensación de que se refieren a

sistemas completamente distintos. Una forma de hacerlo consiste en combinar varias vistas en una sola, aunque muchas veces esta no es la solución más aconsejable, ya que pueden aparecer diagramas demasiado complejos. En esos casos lo más conveniente es mantener las vistas por separado y elaborar una tabla que muestre qué elementos de una vista se corresponden con los elementos de otra de las vistas.

- Guía de variabilidad: esta guía se escribe para documentar aspectos del sistema que pueden cambiar en función de distintos factores. Estos cambios pueden ser de dos tipos: variabilidad y dinamismo. La variabilidad se da cuando cuando algún elemento de la arquitectura puede sustituirse por otro. Este caso ocurre, por ejemplo, cuando todavía no se ha tomado una decisión respecto a esa parte de la arquitectura pero ya se han explorado distintas opciones, cuando se planea construir una familia de sistemas y esa parte de la arquitectura cambia en cada miembro de la familia o cuando se está diseñando un *framework* que contiene puntos en los que puede extenderse. En cualquiera de los casos, es necesario definir cuáles son los puntos de variación, cuáles son los elementos afectados por el cambio y en qué momento se produce el cambio (diseño, compilación, ejecución, etc.). El dinamismo, en cambio, es una propiedad del sistema en ejecución. En general tiene que ver con la creación y eliminación de distintos componentes en función de los servicios requeridos por los usuarios. También se produce dinamismo cuando los componentes del sistema pueden cambiar dinámicamente de procesador para, por ejemplo, mejorar el rendimiento de la aplicación.

- Interfaces: la interfaz de un elemento es el medio por el cual ese elemento interactúa con su entorno, considerando la interacción como cualquier cosa que hace un elemento que puede afectar a otro. La información acerca de una interfaz depende de la vista donde se documente dicha interfaz. En general, las interfaces son bidireccionales, por lo que es importante documentar no sólo lo que un elemento ofrece al exterior, sino también lo que ese elemento necesita del exterior. Además, dentro de una misma vista, un elemento puede tener más de una interfaz para, por ejemplo, restringir los servicios a los que otros elementos pueden acceder. Además, la existencia de múltiples interfaces facilita la evolución de un elemento.

- Análisis: en este apartado se documenta cualquier tipo de análisis que se realice sobre la arquitectura, como los resultados de una evaluación de la seguridad o del impacto de realizar alguna modificación.

5.4.2. Selección de la Documentación a Generar

Los autores de (Clements et al., 2002a) proponen, a partir de la documentación descrita, un pequeño proceso para decidir qué documentos resultan de utilidad y cuáles no.

En primer lugar, sugieren la creación de una tabla como la mostrada en la Tabla 5.1, donde se muestran los tipos de documentación que se pueden generar y los perfiles afectados por el sistema considerado. En las casillas de la tabla se representa la documentación que es necesario generar para cada perfil y el nivel de detalle adecuado para cada uno de ellos.

Sin embargo, como se puede apreciar, la cantidad de documentación que resulta de considerar el contenido de la tabla sin más posee un tamaño considerable. Por este motivo, los autores del citado trabajo sugieren combinar y eliminar distintas vistas, proporcionando unas guías adicionales para filtrar los resultados obtenidos a partir de la tabla anterior, de manera que la documentación generada no sea excesiva y continúe resultando de utilidad. Así, por ejemplo, indican que en sistemas de tamaño pequeño o medio, como el que nos ocupa, pueden no documentarse las vistas de implementación y asignación de trabajo, pues con toda probabilidad se pueden solapar con la de descomposición. Además, dependiendo del tipo de sistema, habrá vistas que no resulten de utilidad, por lo que no será necesario generar la documentación correspondiente a las mismas.

Como tercer paso se propone realizar una priorización de la documentación según su importancia, en función de restricciones de tiempo o presupuestarias. Puesto que esta situación no es aplicable en el presente trabajo, se ha omitido la realización de este paso.

Como resultado de la realización de este proceso, la documentación que se va a generar es la que se indica a continuación. Es importante señalar que a medida que se diseña y refina la arquitectura pueden aparecer nuevas necesidades en lo que a documentación se refiere, por lo que lo que aquí se presenta es el resultado final:

- Vista de descomposición: con esta vista se mostrarán, de manera progresiva, los elementos de los que se compone el sistema. Será de especial importancia porque en ella se fijarán los principales criterios de descomposición del sistema en distintos módulos, por lo que gran parte de la modificabilidad del sistema dependerá de lo adecuado de las decisiones adoptadas para realizar la descomposición.

- Vista de uso: con esta vista se mostrarán las dependencias existentes entre los elementos de la vista anterior. Esta vista resulta de especial utilidad para analizar la modificabilidad del sistema, por

		Participantes en el proyecto						
		Jefe de proyecto	Arquitectos	Equipo desarrollo	Diseñador externo	Equipo mantenim.	Analista	Nuevo miembro
Vistas de Módulo	Descomposición	◀	■	■		■	■	○
	Generalización	◀	■	■		■	■	○
	Uso		■	■		■	◀	○
	Capas	◀	■	■		■	■	○
Vistas C & C	Tuberías y filtros		■	■		■	◀	○
	Datos Compartidos		■	■		■	◀	○
	Cliente-Servidor		■	■		■	◀	○
	Editor-Suscriptor		■	■		■	◀	○
	Peer-to-peer		■	■		■	◀	○
	Procesos Comunicados		■	■		■	◀	○
Vistas de Asignación	Despliegue	■	■	◀		◀	■	○
	Implementación		◀	◀		◀		○
	Asignación de trabajo	■	◀					○
Otra Documentación	Interfaces		■	■	■	■	■	○
	Diagrama Contexto	•	■	■	•	■	■	○
	Mapeo de Vistas		■	■		■		○
	Guía Variabilidad		■	■		■	◀	○
	Análisis		■				■	○
	Decisiones de diseño	◀	■	◀		■	◀	○

■ = información detallada, ◀ = algunos detalles, • = descripción, ○ = cualquier nivel de detalle

Tabla 5.1: Nivel de detalle de la documentación de la arquitectura en función de los miembros del equipo de trabajo

lo que, suponiendo una vista bien documentada, la aparición de pocas dependencias será una buena señal de cara a la mencionada característica.

- Vista editor-suscriptor: el uso de editores y suscriptores es una de las técnicas que mencionan los autores de (Bass et al., 2003) como adecuadas para conseguir aplicaciones modificables, y es una de las que van a utilizarse para el diseño de la arquitectura software que nos ocupa. Por ello, deberá ser documentada convenientemente a través de esta vista.

- Vista *peer-to-peer*: como se explicará más adelante, la aproximación utilizada en esta arquitectura, a diferencia de la anterior, no va a ser jerárquica, sino que cada agente solicitará los servicios de otros directamente, sin utilizar intermediarios. De esta manera, todos los agentes tendrán la misma categoría, por lo que se ha considerado que esta vista es la más adecuada para reflejar esta característica.

- Vista de despliegue: esta vista se utilizará para proponer una posible instalación del sistema resultante de utilizar la arquitectura propuesta, teniendo en cuenta las características de las aplicaciones que se le piensan dar al sistema resultante.

5.4.3. Plantilla de Documentación

Para documentar cada una de estas vistas se dividirá la documentación en paquetes. Esta división en paquetes puede realizarse atendiendo tanto al tamaño de la parte del sistema a documentar como al nivel de profundidad en el que nos adentremos en la arquitectura. La organización de la documentación de cada vista o paquete será la que se indica a continuación:

1. Presentación de la vista: muestra los elementos existentes en la parte de la vista representada por este paquete y las relaciones existentes entre ellos. Puede hacerse de manera gráfica (es lo más habitual), en forma de tabla o mediante una descripción.

2. Catálogo de elementos: contiene, al menos, los elementos mostrados en la presentación de la vista y otros que se hayan podido omitir. Más específicamente, debe contener:

   a) Elementos y sus propiedades: enumera cada uno de los elementos de la vista o paquete y describe sus propiedades.

   b) Relaciones y sus propiedades: cada vista representa una relación distinta entre los elementos que contiene. Si la presentación de la vista no contiene todas las relaciones o si existen excepciones a lo mostrado en la presentación de la vista, en esta sección se recoge la información pertinente.
   c) Interfaces: en esta sección se documenta tanto lo que proporciona un elemento como lo que necesita del entorno.
   d) Comportamientos: aquí se documentan interacciones complejas o comportamientos útiles de cara al análisis del sistema.

3. Diagrama de contexto: muestra cómo se relaciona el elemento descrito con los elementos con los que debe interactuar.

4. Guía de variabilidad: muestra cómo puede tener lugar cualquier forma de variación o dinamismo previstas sobre el elemento descrito.

5. Información sobre la arquitectura: recoge información acerca de las decisiones de diseño que se han tomado para que la arquitectura haya dado el resultado que se muestra. El objetivo principal es justificar por qué se han tomado determinadas decisiones y mostrar que se ha hecho de manera razonada.

   a) Decisiones de diseño: se describen las decisiones de diseño adoptadas así como las opciones que se han descartado y por qué se han descartado, lo cual evita que en el futuro vuelvan a considerarse las opciones descartadas salvo que cambien las circunstancias.
   b) Análisis de resultados: recoge las resultados de las evaluaciones que se realicen sobre la arquitectura, como por ejemplo una lista de cambios necesarios para una determinada modificación.
   c) Suposiciones: cualquier circunstancia que se haya asumido al realizar el diseño, como la disponibilidad de un determinado sistema ajeno al que se está diseñando.

6. Otra información: cualquier otra información de naturaleza no arquitectónica.

7. Vistas relacionadas: en general, la vista o paquete del que se deriva la vista o paquete actual (padre), los que se derivan de él (hijos) o los que se derivan del mismo que él (hermanos).

En cada una de las vistas sólo se documentarán, de entre los puntos anteriores, aquellos que se consideren aplicables o necesarios.

5.5. Diseño de la Arquitectura

A continuación se presentan las distintas vistas de la arquitectura del sistema atendiendo a la selección de documentación realizada en la sección 5.4.2 y con el formato de la plantilla descrita en la sección 5.4.3. En primer lugar, se ven los paquetes de la vista de descomposición, seguidos por los de la vista de uso, los de la vista editor-suscriptor, la vista *peer-to-peer* y, finalmente, la vista de despliegue.

Todos los diagramas que se muestran, salvo indicación expresa en otro sentido, se han realizado utilizando la notación UML.

5.5.1. Vista de Descomposición

En esta vista se muestra, de forma progresiva, la descomposición del sistema en subsistemas y de éstos en módulos. No se reproduce el comportamiento de los módulos, el cual se podrá ver en las vistas editor-suscriptor y *peer-to-peer*.

5.5.1.1. Vista de Descomposición Paquete 1: Sistema

A. Presentación de la vista

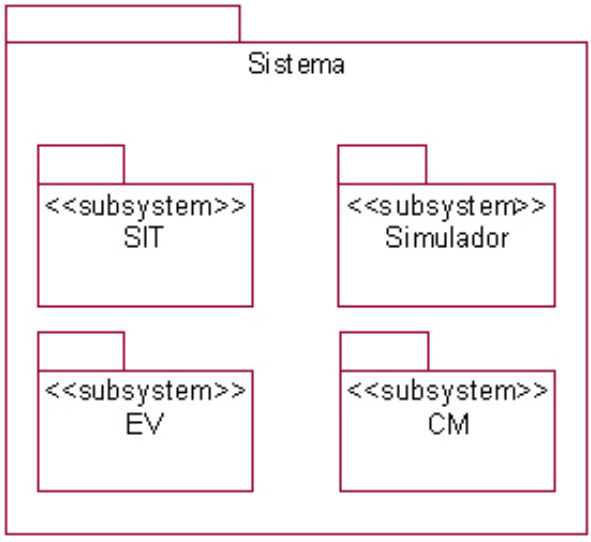

Figura 5.15: Descomposición del sistema

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Entorno Virtual	El EV constituye la principal forma de interacción del usuario con la aplicación. Proporciona una interfaz tridimensional que reproduce el entorno real en el que se realiza el entrenamiento, con las licencias pertinentes debidas a restricciones técnicas o a ayudas al proceso de entrenamiento. A través de la navegación por el escenario y la manipulación de los objetos que en él aparecen, el estudiante intentará realizar los ejercicios que se le planteen.
Simulador	Reproduce el comportamiento de distintos elementos presentes en un EV, ya sea como respuesta a las acciones de los usuarios o como resultado de la evolución o funcionamiento normal de cada elemento en cuestión.
Sistema Inteligente de Tutoría	En este subsistema reside todo lo relativo al proceso de tutoría. Es el encargado de registrar y procesar las acciones del usuario, decidir si son correctas y dar una respuesta adecuada a las mismas. También se encarga de procesar las preguntas de los estudiantes y proporcionarles una respuesta adecuada en función de los parámetros manejados por la estrategia de tutoría. Es dentro de este subsistema donde se realizan la planificación y el cálculo de rutas.
Centro de Mensajes	Es el encargado de manejar la comunicación entre los otros subsistemas, aunque admite un número indeterminado de ellos. Funciona con un mecanismo de suscripción. Cada subsistema se suscribe a los tipo de mensajes que quiere recibir, y los mensajes enviados por cada subsistema sólo los reciben aquellos que están suscritos a esos mensajes. Admite suscripciones a tipos de mensajes concretos o a mensajes enviados por clientes específicos. El Centro de Mensajes no conoce a priori los tipos de mensajes que puede recibir ni de quién, por lo que admite todas las suscripciones que reciba.

Es importante señalar que el sistema, como tal, no existe como elemento, por lo que todos los elementos descritos son visibles por otros elementos con las restricciones impuestas por la utilización del centro de mensajes.

B.2. Relaciones y sus propiedades

La relación que dirige la descomposición de esta vista es *forma-parte-de*. No hay excepciones ni añadidos a lo que define esta relación.

B.3. Interfaces

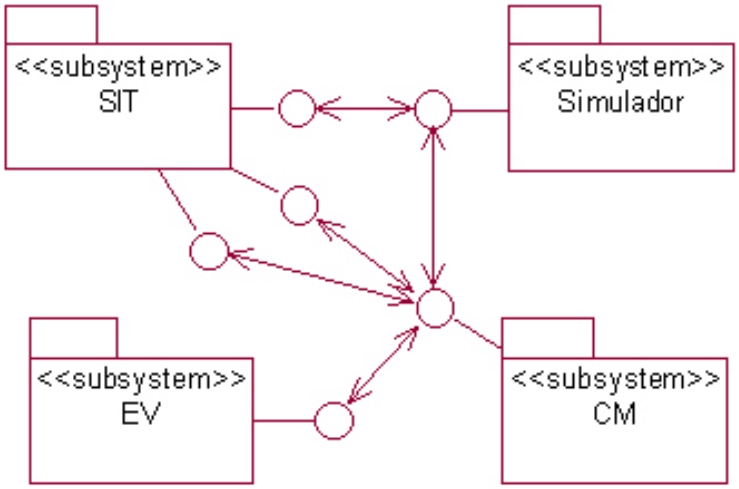

Figura 5.16: Interfaces de los módulos del sistema

Entorno Virtual	Recibe comandos para mover un avatar por el escenario, así como otros para realizar acciones dentro del mismo. Proporciona las posiciones y orientaciones actuales de los objetos presentes en un escenario, así como las acciones realizadas por los usuarios.
Simulador	Recibe las acciones que afectan al estado de la simulación. Proporciona los resultados de las acciones y permite consultar el estado de los elementos que forman parte de la simulación.

Sistema Inteligente de Tutoría	Recibe las notificaciones de conexión de estudiantes y, para cada uno de ellos, sus acciones, preguntas y movimientos. Proporciona distintas respuestas a las acciones de los estudiantes en función de los parámetros manejados por la estrategia de tutoría. También facilita respuestas a sus preguntas y proporciona pistas para ayudar a los estudiantes a resolver los ejercicios.
Centro de Mensajes	Recibe peticiones de suscripción y desuscripción a distintos tipos de mensajes. Proporciona a los suscriptores los mensajes recibidos de otros subsistemas conectados a él, siempre y cuando sean de alguno de los tipos a los que se han suscrito. La manera correcta de utilizar el Centro de Mensajes es suscribirse a los mensajes que se desee recibir y desuscribirse una vez no se desee recibirlos más.

Todas las interfaces se definen de forma más concreta en el paquete correspondiente a cada módulo.

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

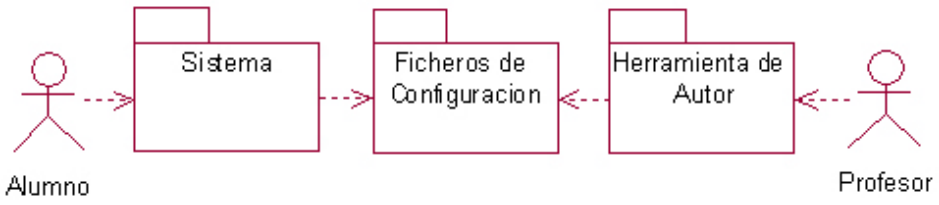

Figura 5.17: Diagrama de contexto del sistema

D. Guía de variabilidad

La conexión del simulador al sistema es susceptible de variar en función de las características del mismo. Para mayor información, ver la guía de

variabilidad en la sección 5.5.1.4. Existirá un único SIT, un único simulador y un único Centro de Mensajes. Sin embargo, habrá tantos EVs como estudiantes estén realizando el entrenamiento.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

Para realizar esta descomposición se ha tenido en cuenta que algunos de los elementos mostrados, en concreto EV y Simulador, forman parte de los elementos susceptibles de sufrir sustituciones, por lo que se ha decidido que es mejor separarlos del resto de módulos del sistema desde el principio, ya que constituyen, de por sí, aplicaciones con entidad propia. El tercer elemento con el que se deben integrar, el SIT, se descompondrá posteriormente.

Para facilitar la independencia entre ellos y también su integración se ha pensado en utilizar un esquema editor-suscriptor (ver sección 5.5.3). Por esta razón aparece el Centro de Mensajes, en el cual los distintos elementos se suscribirán a los mensajes que quieran recibir. De esta manera, los productores de información no son conscientes de quiénes son los consumidores, lo cual facilita la sustitución del EV y el simulador e incluso su conexión y desconexión en función de las necesidades del entrenamiento. Como facilidad adicional, también posibilita la conexión de otros elementos, como una consola, que ayuden en el proceso de desarrollo y depuración del sistema, lo cual, pese a no ser uno de los objetivos buscados, mejora la *testabilidad* de la aplicación.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

Para la realización de esta descomposición se supone que se dispone de una red de banda ancha, de forma que la comunicación por la red no supone un cuello de botella debido a la velocidad de la misma. Además, el número y tamaño de los mensajes que se intercambien no serán muy elevados, por lo que no se espera que el Centro de Mensajes sea un cuello de botella en las comunicaciones.

La descomposición también supone que los tres subsistemas que se conectan al centro de mensajes pueden funcionar sin alguno de los otros, o sin todos ellos, aunque la ausencia de las entradas provenientes de ellos suponga un funcionamiento más simple.

Puesto que la comunicación se realiza con un mecanismo de suscripción, todos los subsistemas asumen que existe al menos un productor de la información a la que se suscriben. De no ser así, los subsistemas deben tener alguna forma de funcionamiento en ausencia de esa información.

Puesto que, de momento, la seguridad no es un problema, no se requiere suscripción al Centro de Mensajes para poder enviar mensajes, sino sól para recibirlos.

F. Otra información

No aplicable.

G. Vistas relacionadas

- Padre: ninguna
- Hermanas: ninguna
- Hijas: ver secciones 5.5.1.2, 5.5.1.3, 5.5.1.4, 5.5.1.5

5.5.1.2. Vista de Descomposición Paquete 2: Entorno Virtual

A. Presentación de la vista

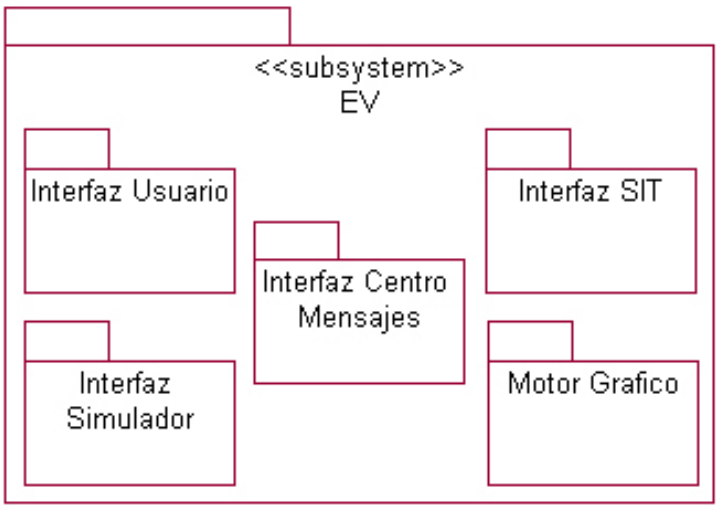

Figura 5.18: Descomposición del EV

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Motor gráfico	Es el módulo que se responsabiliza de cargar los escenarios 3D y mostrarlos por pantalla, así como de ir actualizando la escena mostrada en función de las entradas procedentes de cualquier actor que interactúe con el EV.
Interfaz de Usuario	Este módulo se encarga de trasformar las entradas proporcionadas por el usuario en llamadas a métodos del motor gráfico que transformen la escena. Debe soportar las distintas posibilidades de interacción con el usuario a través de los dispositivos que se determinen, como joystick, teclado, ratón, sensor de movimiento o reconocedor de voz.
Interfaz con el SIT	Tiene como responsabilidad la traducción de los mensajes enviados desde el SIT a llamadas a métodos del motor gráfico y viceversa.
Interfaz con el Simulador	Se encarga de transformar en llamadas a métodos del motor gráfico los mensajes recibidos desde el simulador y viceversa.
Interfaz con el Centro de Mensajes	Se encarga de realizar el encaminamiento y la traducción de mensajes entre el centro de mensajes y los demás módulos del EV.

B.2. Relaciones y sus propiedades

La relación que dirige la descomposición de esta vista es *forma-parte-de*. No hay excepciones ni añadidos a lo que define esta relación.

B.3. Interfaces

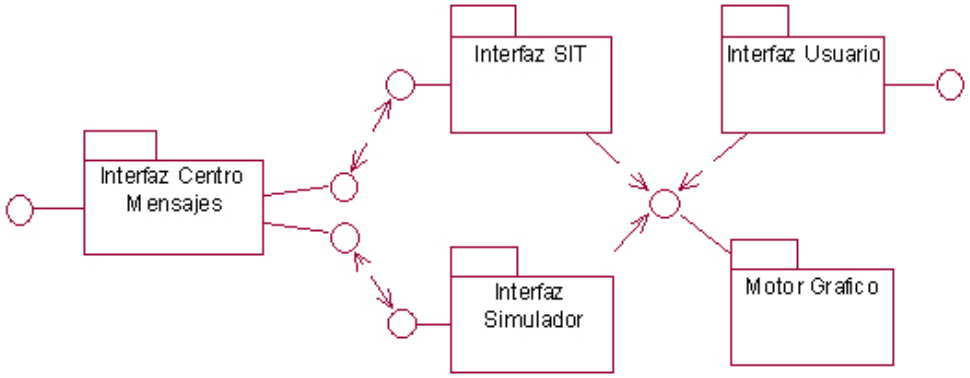

Figura 5.19: Interfaces de los módulos del EV

Motor gráfico	Proporciona servicios para crear y destruir objetos, cambiar sus propiedades, como posición, orientación o color, e interactuar con ellos. También proporciona servicios para reproducir animaciones y la posibilidad de consultar los nuevos valores de las propiedades de los elementos. Sólo es visible desde dentro del EV.
Interfaz con el usuario	Recibe las órdenes enviadas por los dispositivos de interacción y las traduce en llamadas a los servicios proporcionados por el motor gráfico, así como en mensajes hacia los demás subsistemas que forman la arquitectura. Puesto que sirve de comunicación con el usuario, esta interfaz es visible desde el exterior del subsistema.
Interfaz con el SIT	Recibe los mensajes enviados por el SIT y los traduce en llamadas a los servicios proporcionados por el motor gráfico. Sólo es visible desde dentro del EV.
Interfaz con el simulador	Recibe los mensajes enviados por el simulador y los traduce en llamadas a los servicios proporcionados por el motor gráfico. Sólo es visible desde dentro del EV.

Interfaz con el Centro de Mensajes	Proporciona tres interfaces de comunicación. La primera permite al Centro de Mensajes enviarle los mensajes a los que está suscrito el EV, mientras que las otras permiten a los demás módulos del subsistema comunicarle los datos que deben ser enviados a los demás subsistemas de la aplicación. Por tanto, la primera interfaz es visible sólo desde el exterior del subsistema, mientras que las otras sólo lo son desde el interior.

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

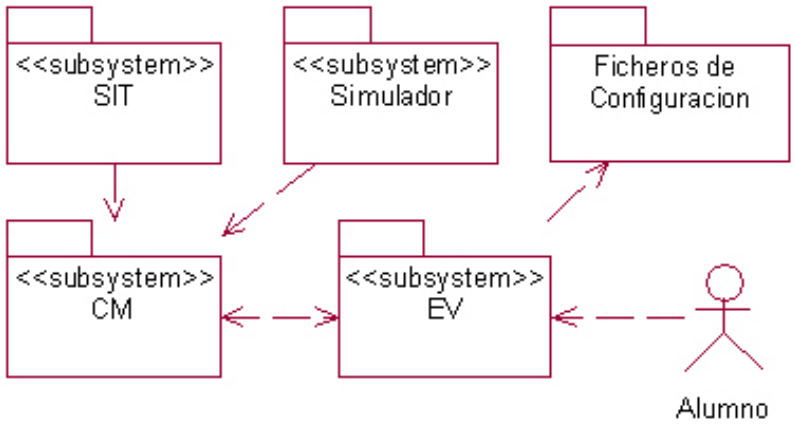

Figura 5.20: Diagrama de contexto del EV

D. Guía de variabilidad

La descomposición variaría si el simulador se encuentra integrado con el EV. Para más información, ver la sección 5.5.1.4.

Dependiendo del simulador utilizado, existe la posibilidad de integrar los módulos de comunicación con el SIT y con el simulador en uno solo.

La posible variabilidad de los dispositivos de interacción entre el usuario y el EV se debe tener en cuenta al realizar el diseño de la interfaz con el usuario.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

Para realizar la división en módulos se ha utilizado el Principio de Separación de Interfaces (ISP, *Interface Segregation Principle*) (Martin, 1999). Este principio establece que es preferible utilizar interfaces separadas para distintos tipos de clientes que una gran interfaz genérica que sirva a todos los clientes.

De esta manera, se han identificado como tipos de clientes potencialmente distintos al usuario, al SIT y al simulador, por lo que se ha creado un paquete específico para cada uno de ellos que realice las funciones de interfaz con el EV. Estas interfaces serán las que realicen las labores de traducción entre las salidas producidas por cada tipo de cliente y las entradas aceptadas por el motor gráfico.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

No aplicable.

F. Otra información

No aplicable.

G. Vistas relacionadas

- Padre: 5.5.1.1
- Hermanas: ver secciones 5.5.1.3, 5.5.1.4, 5.5.1.5
- Hijas: ninguna

5.5.1.3. Vista de Descomposición Paquete 3: Centro de Mensajes

A. Presentación de la vista

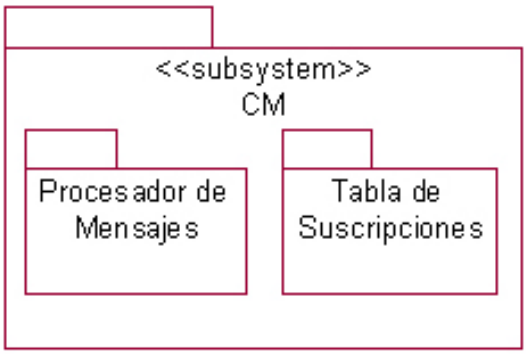

Figura 5.21: Descomposición del Centro de Mensajes

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Tabla de Suscripciones	En esta tabla se mantienen las suscripciones que los clientes del Centro de Mensajes han realizado. Estas suscripciones se organizan tanto por tipo de mensaje al que se suscribe como por el remitente de los mismos, de manera que puedan realizarse ambos tipos de suscripciones.
Procesamiento de Mensajes	Este módulo implementará la lógica de funcionamiento del centro de mensajes. Por una parte, rellena la información que contiene la Tabla de Suscripciones, y por otra, consulta la tabla para saber a quién reenviar los mensajes recibidos en el Centro de Mensajes.

B.2. Relaciones y sus propiedades

La relación que dirige la descomposición de esta vista es *forma-parte-de*. No hay excepciones ni añadidos a lo que define esta relación.

B.3. Interfaces

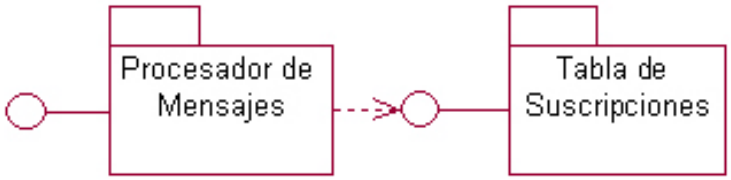

Figura 5.22: Interfaces del Centro de Mensajes

Tabla de Suscripciones	Proporciona funciones para registrar nuevas suscripciones, para consultar las suscripciones existentes y para borrar suscripciones. Sólo es visible desde dentro del Centro de Mensajes.
Procesamiento de Mensajes	Proporciona servicios para suscribirse a mensajes por tipo y por remitente. Espera que los clientes conectados al Centro de Mensajes proporcionen servicios para enviarles los mensajes a los que se han suscrito.

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

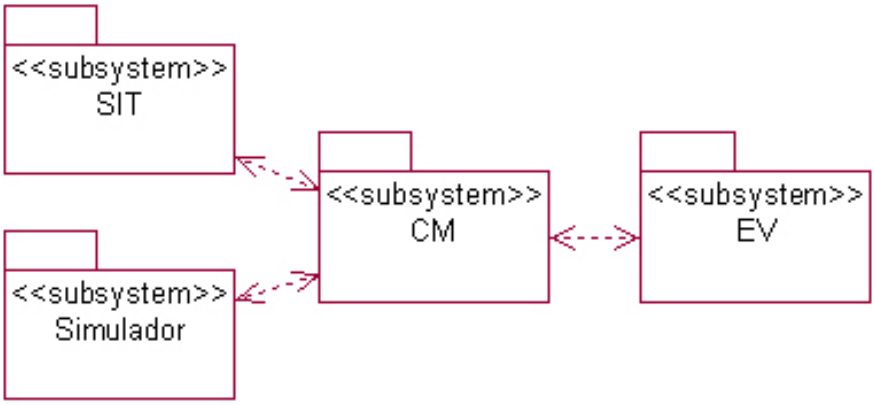

Figura 5.23: Diagrama de contexto del Centro de Mensajes

D. Guía de variabilidad

No aplicable.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

Se ha decidido hacer uso de la tabla de suscripciones por dos razones. La primera es evitar que las comunicaciones queden fijadas en el código, con lo que también se evita modificarlo para cambiar los detalles relativos a los clientes. La segunda es que se posibilita la modificación del procesamiento de los mensajes sin afectar a las suscripciones.

Es posible que sea necesario cambiar la estructura de la tabla de suscripciones si se cambia la estructura de los mensajes o si se desea modificar el tipo de suscripciones.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

Se ha supuesto que los clientes del Centro de Mensajes no tienen por qué funcionar siempre correctamente. De esta manera, si un cliente se

desconecta sin notificarlo ni borrar las suscripciones, el Centro de Mensajes lo realizará de manera automática cuando detecte que el cliente no está activo.

F. Otra información

No aplicable.

G. Vistas relacionadas

- Padre: 5.5.1.1
- Hermanas: ver secciones 5.5.1.2, 5.5.1.4, 5.5.1.5
- Hijas: ninguna

5.5.1.4. Vista de Descomposición Paquete 4: Simulador

A. Presentación de la vista

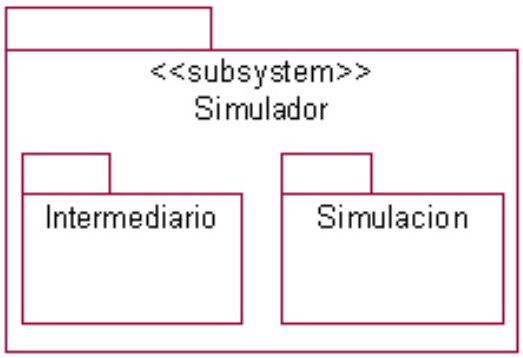

Figura 5.24: Descomposición del Simulador

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Simulación	Este módulo es el que realmente ejecuta la simulación. Su responsabilidad varía en función del dominio de entrenamiento, pero básicamente consiste en reproducir el comportamiento de objetos o habitantes del entorno de entrenamiento que no están bajo control directo de ningún usuario o de otro subsistema de la aplicación.
Intermediario	Puesto que la simulación puede no ser sencilla de modificar en caso de que esté ya implementada, este módulo es responsable de interactuar con la simulación y el resto de subsistemas que se tengan que comunicar con ella. De esta manera, oculta a la simulación los detalles de comunicación con el resto de la aplicación. En caso de ser necesario, también se ocupa de la traducción de mensajes e información entre la simulación y el resto de elementos de la aplicación.

B.2. Relaciones y sus propiedades

La relación que dirige la descomposición de esta vista es *forma-parte-de*. No hay excepciones ni añadidos a lo que define esta relación.

B.3. Interfaces

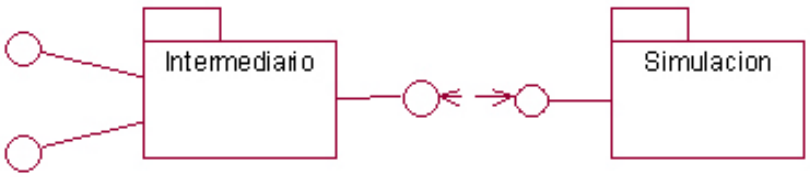

Figura 5.25: Interfaces del Simulador

Simulación	Proporciona servicios para realizar acciones sobre elementos de la simulación, así como para consultar el estado o propiedades de los elementos de la misma. Sólo resulta visible desde dentro del subsistema.
Intermediario	Proporciona tres interfaces: una para comunicarse con el SIT, otra para la comunicación con el Centro de Mensajes y una tercera para recibir notificaciones del simulador. La interfaz con el centro de mensajes ofrece la posibilidad de recibir mensajes de éste, mientras que la interfaz con el SIT recibe directamente las acciones o consultas a realizar sobre la simulación.

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

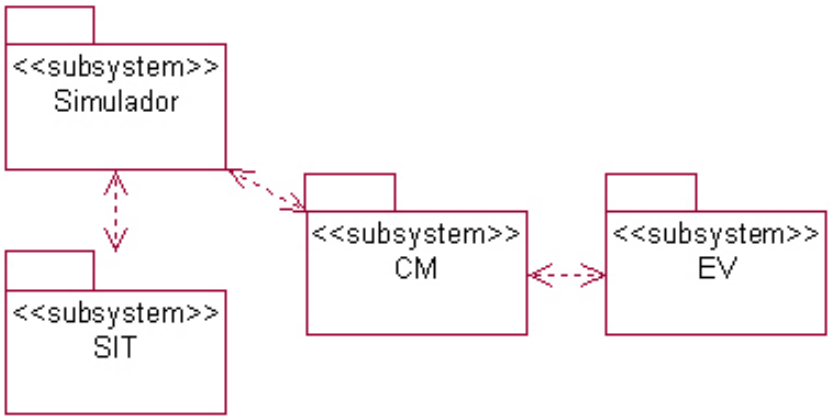

Figura 5.26: Diagrama de contexto del Simulador

D. Guía de variabilidad

El simulador es un componente que presenta una amplia gama de posibilidades a la hora de conectarlo al sistema, dependiendo de si ya está o

no implementado y de cómo lo está. Las situaciones que se han identificado son:

- Simulador implementado e integrado en el EV: es una situación relativamente habitual y posiblemente la más fácil de resolver, siempre y cuando el conjunto proporcione un mecanismo de comunicación tanto con el EV como con el simulador. La comunicación con el SIT se realizará a través del Centro de Mensajes, por lo que el SIT no tendrá conocimiento de la estructura del simulador ni del EV.

- Simulador implementado e independiente: en este caso se presentan varias posibilidades, debidas a la naturaleza del simulador y del dominio en el que se realice el entrenamiento.

  - Si el simulador realiza acciones que afectan tanto al funcionamiento del SIT como a la visualización en el EV, lo más adecuado puede resultar que el simulador se comunique con el retos de subsistemas a través del Centro de Mensajes.
  - Si el número de eventos que debe notificar el simulador es elevado, será mejor que la comunicación tenga lugar sin hacer uso del Centro de Mensajes para mantener así una buena velocidad de ejecución de la aplicación.
  - Se puede dar el caso de que el simulador realice muchos cambios que deban reflejarse en el EV, pero que no afecten mucho al SIT. En esta situación, es preferible que se comunique directamente con el EV, posiblemente también sin hacer uso del Centro de Mensajes por la misma razón que en el caso anterior. La comunicación que afecte al SIT sí se podrá realizar a través del Centro de Mensajes.

- Simulador no implementado: si el simulador es sencillo, se puede implementar de manera que se comunique directamente con el SIT, de forma que no sea más complejo el mecanismo de comunicación que el propio simulador. En otro caso, nos podemos remitir a alguna de las situaciones anteriores.

También se puede comunicar el simulador directamente con el SIT en caso de que lo eventos del simulador no afecten a ningún otro subsistema, sino tan sólo a las decisiones que deba tomar el SIT durante la tutoría.

Teniendo en cuenta que es deseable mantener un alto grado de coherencia en el diseño de la arquitectura, resulta difícil encontrar una solución aceptable para todos los casos, por lo que resultará más conveniente considerar la integración del simulador como un punto de variación y adoptar distintas soluciones en cada caso.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

Las simulaciones manejadas actualmente en el grupo de investigación en el que se ha elaborado el presente trabajo no realizan excesivas actualizaciones en el estado de los objetos que se muestran en el EV. De igual modo, en el EV no se llevan a cabo demasiadas acciones que afecten al estado del simulador. Por esta razón, parece adecuado mantener la integridad conceptual del diseño arquitectónico y realizar la comunicación a través del Centro de Mensajes.

En cuanto al simulador en sí, su funcionamiento debe mantenerse independiente del hecho de que la comunicación se realice o no a través del centro de mensajes, y también de con quién se realice. Por esta razón se ha decidido incluir un intermediario entre la simulación y el resto de elementos. Este intermediario será quien se ocupe de los detalles de comunicación, de manera que si hay que modificar algún aspecto relativo a ella baste con modificar el intermediario, pero no la simulación.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

Para la solución adoptada se ha supuesto que el simulador no está implementado, es de complejidad no muy elevada y no realiza demasiadas actualizaciones en el estado del EV.

F. Otra información

Uno de los motivos para decidir cómo comunicar el simulador con el resto de elementos de la aplicación radica en el hecho de que se utilice JADE como plataforma para implementar los agentes. JADE implementa el mecanismo de envío de mensajes entre unos agentes y otros de forma tal que todos los agentes se ven afectados por todos los mensajes que se envían (no los reciben, pero se enteran de que se han enviado). Si el simulador envía muchos mensajes, todos los agentes se verían afectados, por lo que el sistema sufriría una disminución en la velocidad de ejecución. Por eso resulta preferible que el simulador se comunique directamente con el agente de simulación. Por esta misma razón, es preferible que no todos los mensajes pasen a través de dicho agente, ya que si no son mensajes relevantes para él (ni para otros agentes) se ralentiza el sistema sin obtener ningún beneficio. Esta es la razón para que

la simulación se comunique con el EV directamente a través del Centro de Mensajes en lugar de aprovechar que ya se comunica con el SIT.

G. Vistas relacionadas

- Padre: 5.5.1.1
- Hermanas: ver secciones 5.5.1.2, 5.5.1.3, 5.5.1.5
- Hijas: ninguna

5.5.1.5. Vista de Descomposición Paquete 5: Sistema Inteligente de Tutoría

A. Presentación de la vista

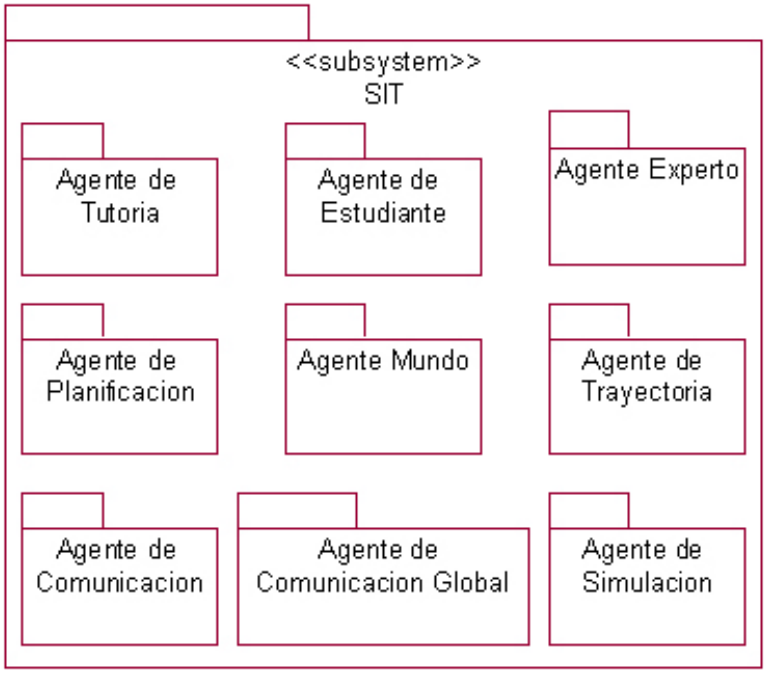

Figura 5.27: Descomposición del SIT

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Agente de Comunicación	Se encarga de la comunicación específica del SIT con un estudiante concreto. De esta manera, recibe, a través del Centro de Mensajes, todos los mensajes provenientes de un único EV, y envía todos los mensajes del SIT que son específicos para ese mismo EV.
Agente de Comunicación Global	Este agente es responsable de una parte de la comunicación del SIT con el resto de subsistemas de la aplicación. En concreto, se encarga de las comunicaciones relativas al inicio de sesión de los alumnos, así como de las comunicaciones que no son específicas de estos, como las acciones que el agente de simulación tenga que enviar al EV. También se encarga de la creación de los agentes de comunicación cuando se conectan nuevos estudiantes.
Agente de Simulación	Es el encargado de manejar en el SIT el conocimiento relativo a la simulación. De esta manera, si un alumno realiza una acción que contiene alguna precondición relativa al estado de la simulación, el Agente de Simulación es quien maneja la información para saber si la condición se cumple o no. De igual manera, si la acción realizada sobre la simulación tiene consecuencias visibles en el EV, el Agente de Simulación será el encargado de emitir la orden de ejecución de la mencionada acción. Este agente realiza labores de comunicación entre el SIT y el Simulador.

Agente Experto	Es el responsable de controlar la ejecución de las acciones realizadas por los estudiantes. Cada acción realizada por un estudiante tiene unas precondiciones, que indican si se puede ejecutar o no independientemente de su corrección, y unas postcondiciones, que indican los efectos de la ejecución de la acción. El Agente Experto está encargado de realizar la validación de las precondiciones que no pueda validar el planificador, solicitando a otros agentes la información que pueda ser necesaria para ello. Una vez ejecutada la acción, es el encargado de enviar las postcondiciones de las acciones para que los agentes interesados actualicen su información al respecto y para que el EV muestre los resultados a los alumnos.
Agente Mundo	Este agente se encarga de mantener actualizada la información del estado del entorno de entrenamiento. De esta manera, todos los demás agentes pueden consultar la información actualizada relativa al estado del mundo sin necesidad de tener que pedirla a los EVs. También es el encargado de responder preguntas acerca de la relación entre distintos objetos, como por ejemplo si están cerca o si uno está encima de otro.
Agente de Planificación	Es el encargado de planificar la ejecución de los ejercicios propuestos a los estudiantes para su entrenamiento. Es capaz de realizar un plan, actualizar el estado del plan a medida que los estudiantes ejecutan acciones y replanificar la ejecución de un ejercicio en caso de que las acciones de los estudiantes se desvíen del plan vigente y no permitan llegar a una solución siguiendo dicho plan.
Agente de Trayectoria	Es el agente responsable de registrar las coordenadas de los movimientos de los estudiantes por el EV y, en caso de requerirlo la actividad objeto de entrenamiento, de comprobar la idoneidad de la trayectoria seguida por los estudiantes en su desplazamiento por el EV y de informar si un estudiante ha llegado a un lugar relevante para la actividad entrenada.

Agente de Estudiante	Este agente es el encargado de realizar el modelo del estudiante en función de las acciones que va llevando a cabo durante el entrenamiento. De esta manera, registra las acciones que el estudiante ejecuta dentro del EV junto con su evaluación, las trayectorias seguidas y su evaluación, las preguntas realizadas y cualquier otra información relevante para el modelo utilizado. Asimismo, proporciona la información contenida en este modelo a los agentes que la soliciten.
Agente de Tutoría	El Agente de Tutoría constituye la pieza angular del SIT, ya que, de una u otra manera, todos los demás agentes trabajan para que el Agente de Tutoría pueda realizar su labor. Esta consiste en ayudar al estudiante en el proceso de entrenamiento. Dentro de sus responsabilidades se encuentra plantear el procedimiento en el que se debe entrenar el estudiante y, una vez iniciado el entrenamiento, monitorizar las acciones que realice. El Agente de Tutoría manejará una o varias estrategias de tutoría, las cuales serán utilizadas para decidir qué hacer ante las acciones de los estudiantes. También será el responsable de contestar a las preguntas del estudiante y de ofrecerle pistas cuando lo considere oportuno. Otra de las tareas que se encuentran a su cargo es evaluar las acciones del estudiante, lo que contribuirá a la realización del modelo del estudiante.

B.2. Relaciones y sus propiedades

La relación que dirige la descomposición de esta vista es *forma-parte-de*. No hay excepciones ni añadidos a lo que define esta relación.

B.3. Interfaces

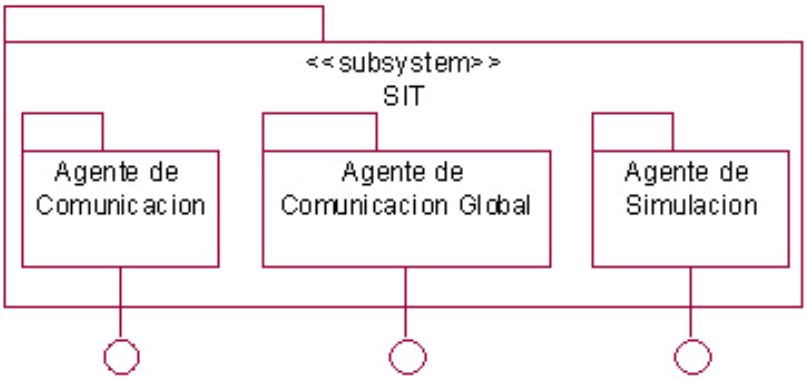

Figura 5.28: Interfaces del SIT

En la imagen se muestran únicamente las interfaces que ofrecen los agentes hacia el exterior del SIT. Las interfaces entre los agentes se muestran en mayor detalle en las vistas de edición-suscripción 5.5.3.

Agente de Comunicación	El agente de comunicación es uno de los puntos de unión entre el SIT y el resto de los subsistemas de la aplicación. De esta manera, ofrece una interfaz para recibir mensajes provenientes del Centro de Mensajes. Este agente no asume la existencia de ningún otro agente, por lo que no envía la información recibida a ningún agente que no se la haya solicitado previamente. Para ello, proporciona a los demás agentes del SIT otra interfaz con servicios para suscribirse a la recepción de mensajes con la información que les interese proveniente del exterior del SIT. También proporciona servicios para el envío de información a otros subsistemas de la aplicación.
Agente de Comunicación Global	Al igual que el anterior, es un punto de comunicación entre el SIT y los demás subsistemas de la aplicación, por lo que proporciona una interfaz para recibir mensajes del Centro de Mensajes. La información recibida se la envía únicamente a los agentes que se suscriban a ella. Además, proporciona servicios para enviar mensajes a otros subsistemas de la aplicación.

Agente de Simulación	Este agente es otra de las puertas de comunicación del SIT con el exterior, en este caso con la simulación. Proporciona servicios para que los agentes del SIT puedan consultar aspectos relativos al estado de la simulación y para que ejecuten acciones dentro de la misma. También proporciona una interfaz para que la simulación pueda enviar cambios relevantes para el SIT. Asume la existencia de un simulador, pero no la de otros agentes.
Agente Experto	Proporciona una interfaz para que se le solicite la validación de las precondiciones de las acciones realizadas por los estudiantes. Necesita que existan agentes que le proporcionen la información necesaria para efectuar la validación de las precondiciones, así como un agente que actúe como intermediario para enviar los resultados de las acciones a los demás subsistemas de la aplicación.
Agente Mundo	Proporciona servicios para consultar y actualizar la información del entorno de entrenamiento. También proporciona servicios para responder preguntas acerca de los objetos de los que mantiene información.
Agente de Planificación	Proporciona servicios para solicitar la realización de un plan, así como de replanificaciones. También permite la aplicación de acciones sobre un plan y la solicitud del plan actual y de la siguiente acción a realizar.
Agente de Trayectoria	Proporciona una interfaz para consultar si un estudiante está siguiendo la trayectoria adecuada o si ha llegado a un lugar relevante. Necesita que algún agente le proporcione las coordenadas con las que debe trabajar.
Agente de Estudiante	Proporciona servicios para consultar y actualizar la información contenida en el modelo del estudiante. Necesita que otros agentes le proporcionen la información con la que elaborar el modelo.

Agente de Tutoría	Proporciona información acerca de las acciones de los estudiantes, así como las correspondientes decisiones de tutoría. Necesita que se le proporcionen las acciones de los estudiantes y las del plan del ejercicio a realizar, así como servicios para validar las acciones de los estudiantes y para consultar datos relativos a los estudiantes.

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

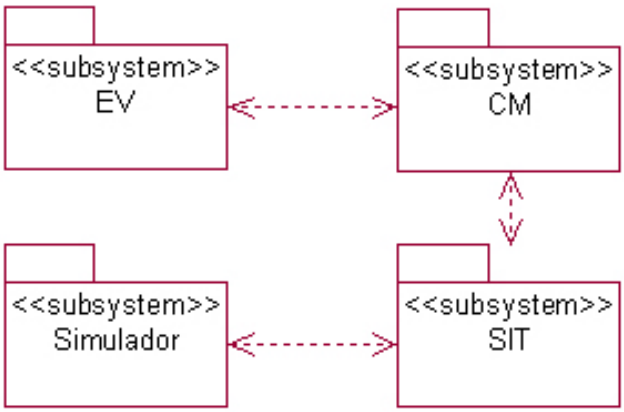

Figura 5.29: Diagrama de contexto del SIT

D. Guía de variabilidad

Como propiedades de dinamismo de este subsistema, cabe destacar que existirán tantos Agentes de Comunicación y tantos Agentes de Estudiante como estudiantes se encuentren conectados al sistema. Ambos tipos de agentes se irán creando a medida que se vayan conectando nuevos estudiantes.

Para información relativa a variabilidad, consultar la sección específica de cada uno de los agentes.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

Se ha decidido crear Agentes de Comunicación y Agentes de Estudiante específicos para cada estudiante por dos motivos. El primero es que así es posible tratar las acciones de los estudiantes en paralelo, de manera que no sea necesario que las acciones de un estudiante tengan que esperar a ser procesadas si no es imprescindible. Por otro lado, cabe la posibilidad de que, debido a un incremento del número de agentes, sea necesario distribuirlos en varios procesadores. Con agentes específicos para cada estudiante resulta posible distribuir los agentes en función de los estudiantes a los que atienden.

Existen otros agentes, como el de Tutoría o el de Planificación, que realizan acciones comunes para todos los estudiantes, por lo que se ha decidido no desdoblarlos. Hacerloo complicaría las labores de diseño debido al alto grado de coordinación que es necesario alcanzar tanto en las labores de planificación como de tutoría, donde es necesario tener en cuenta las acciones de todos los alumnos que toman parte en una sesión de entrenamiento.

Como ya se ha mencionado al describir el Agente de Tutoría, éste es la pieza central del SIT, razón por la cual puede existir la tendencia a asignarle mucha responsabilidad. Como se verá en las próximas secciones, las distintas responsabilidades de los agentes se han separado en distintos comportamientos que deberán mantenerse independientes para, llegado el caso, poder modificar la asignación de responsabilidades e, incluso, introducir agentes nuevos que se hagan cargo de ellas. Este ha sido el caso de agente experto, que ha surgido en parte de la necesidad de reducir la carga de responsabilidades del agente de tutoría, lo que también permite que éste atienda la acción de un alumno mientras se verifica que puede ejecutarse la acción realizada por otro.

Para realizar la descomposición se han tenido en cuenta varios factores. El más influyente ha sido la necesidad de que ciertos componentes deban ser sustituibles (o eliminables), lo que ocasiona que tengan que contemplarse como componentes separados. Este es el caso de los agentes de planificación, estudiante, trayectoria y tutoría.

Para identificar el resto de módulos se ha aplicado el **principio de ocultación de información** (Parnas, 1972) para obtener un diseño más modificable. Existen distintas interpretaciones acerca de la aplicación de este principio. A juicio del autor del presente trabajo, en ocasiones se hace una interpretación excesivamente simplista del mismo, pues se argumenta que consiste en la ocultación de datos o de decisiones de diseño dentro de un módulo, o en la agrupación de elementos que "comparten un secreto" dentro de un mismo módulo. Más que eso, el principio de ocultación de información

implica la utilización de la abstracción como mecanismo de diseño, lo que sirve para enfrentarse a la incertidumbre provocada por requisitos cambiantes o desconocidos. Para ello, el módulo que implementa una responsabilidad es el encargado de tomar las decisiones que conlleva esa responsabilidad, lo que incluye decidir qué información hace falta y qué se hace con ella.

La utilización de agentes proporciona enormes ventajas a este respecto, ya que la principal forma de comunicación entre ellos, el paso de mensajes, desacopla a los emisores y a los receptores de los mismos, posibilitando el intercambio de unos agentes por otros sin que el agente que solicita un servicio perciba el cambio. También ayuda a ello el hecho de que un agente no tiene por qué saber qué agente va a prestarle un servicio hasta que el sistema está en ejecución, a través de un servicio de búsqueda en unas páginas amarillas. Todo esto se debe ver complementado, no obstante, por una manera adecuada de solicitar los servicios, para lo que las interfaces de comunicación entre los agentes deben estar diseñadas adecuadamente.

Se puede encontrar más información en las vistas de descomposición de cada agente, en la vista editor-suscriptor y en las vistas *peer-to-peer*.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

Se ha supuesto que la implementación del SIT se va a realizar utilizando la plataforma JADE, la cual proporciona una serie de servicios y mecanismos que no se muestran en los diagramas, como pueden ser los servicios de páginas blancas y páginas amarillas. De igual manera, también proporciona la ejecución en paralelo de los agentes y mecanismos de comunicación y sincronización, razón por la cual no es necesario incorporarlos entre los módulos en que se descompone el SIT.

F. Otra información

No aplicable

G. Vistas relacionadas

- Padre: 5.5.1.1
- Hermanas: ver secciones 5.5.1.2, 5.5.1.3, 5.5.1.4
- Hijas: ver secciones 5.5.1.6 a 5.5.1.14.

5.5.1.6. Vista de Descomposición
Paquete 6: Agente de Comunicación Global

A. Presentación de la vista

Figura 5.30: Descomposición del Agente de Comunicación Global

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Agente de Comunicación Global	Este elemento, aunque lleva el mismo nombre que el módulo que lo contiene, constituye en sí lo que es el agente, el cual tiene asociados una serie de comportamientos que llevan a cabo la mayor parte de las responsabilidades que le corresponden a éste. La responsabilidad principal de este elemento consiste en la creación y puesta en marcha inicial de los comportamientos, así como ejercer de representante de los demás elementos ante el resto de los agentes, enviando y recibiendo mensajes. También se encarga de crear a los Agentes de Comunicación cuando se reciben conexiones de nuevos estudiantes. Para ello debe registrarse en el Centro de Mensajes para recibir los mensajes de nuevas conexiones.

Comunicación con el Centro de Mensajes	Este elemento es un comportamiento del Agente de Comunicación Global que está encargado de toda la comunicación con el Centro de Mensajes. Cuando el agente recibe un mensaje de otro agente lo envía al Centro de Mensajes, mientras que si el mensaje proviene del Centro de Mensajes, entonces se lo envía a los agentes que estén suscritos a ese tipo de mensaje.
Gestión de Suscripciones	Este comportamiento se encarga de recibir las peticiones de suscripción de otros agentes a los mensajes provenientes del Centro de Mensajes (y, por tanto, de otros subsistemas de la aplicación) y de apuntarlas en la Tabla de Suscripciones.
Tabla de Suscripciones	Este elemento recoge las suscripciones de otros agentes a mensajes provenientes del exterior del SIT.

B.2. Relaciones y sus propiedades

La relación que dirige la descomposición de esta vista es *forma-parte-de*. No hay excepciones ni añadidos a lo que define esta relación.

B.3. Interfaces

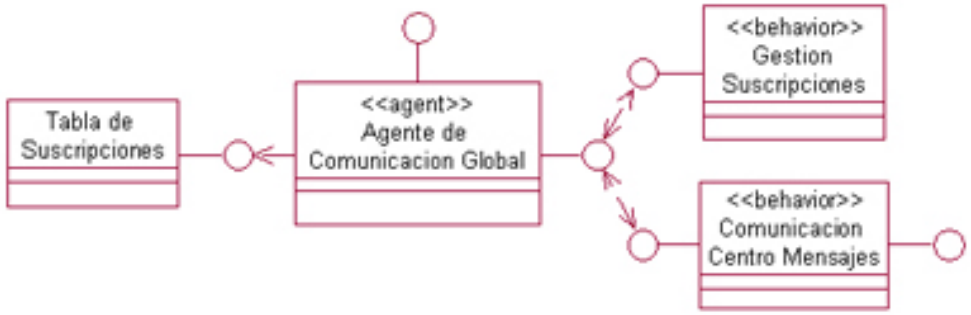

Figura 5.31: Interfaces del Agente de Comunicación Global

Agente de Comunicación Global	Proporciona interfaces para enviar y recibir mensajes de otros agentes. No asume la existencia de ningún otro agente.
Comunicación con el Centro de Mensajes	Proporciona servicios para mandar y recibir mensajes del Centro de Mensajes. Asume la existencia del Centro de Mensajes.
Gestión de Suscripciones	Proporciona servicios para suscribirse a mensajes por remitente y por tipo de contenido. También proporciona servicios para borrar suscripciones. Necesita la existencia de un elemento que almacene las suscripciones.
Tabla de Suscripciones	Proporciona servicios para añadir, borrar y modificar suscripciones.

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

Figura 5.32: Diagrama de contexto del Agente de Comunicación Global

D. Guía de variabilidad

No aplicable.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

Puesto que este agente no necesita entender el contenido de los mensajes, se ha decidido dotarlo únicamente de dos comportamientos. Cada uno de ellos se encarga de la comunicación con un tipo de elemento diferente: agentes y Centro de Mensajes.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

No aplicable.

F. Otra información

No aplicable.

G. Vistas relacionadas

- Padre: 5.5.1.5
- Hermanas: ver secciones 5.5.1.7, 5.5.1.8, 5.5.1.9, 5.5.1.10, 5.5.1.11, 5.5.1.12, 5.5.1.13, 5.5.1.14
- Hijas: ninguna.

5.5.1.7. Vista de Descomposición
Paquete 7: Agente de Comunicación

A. Presentación de la vista

Figura 5.33: Descomposición del Agente de Comunicación

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Agente de Comunicación	Este elemento, aunque lleva el mismo nombre que el módulo que lo contiene, constituye en sí lo que es el agente, el cual tiene asociados una serie de comportamientos que llevan a cabo la mayor parte de las responsabilidades que le corresponden a éste. La responsabilidad principal de este elemento consiste en la creación y puesta en marcha inicial de los comportamientos, así como ejercer de representante de los demás elementos ante el resto de los agentes, enviando y recibiendo mensajes. Además, debe registrarse en el Centro de Mensajes para recibir los mensajes provenientes del estudiante que causó su creación.

Comunicación con el Centro de Mensajes	Este elemento es un comportamiento del Agente de Comunicación que está encargado de toda la comunicación con el Centro de Mensajes. Cuando el agente recibe un mensaje de otro agente lo envía al Centro de Mensajes, mientras que si el mensaje proviene del Centro de Mensajes, entonces se lo envía a los agentes que estén suscritos a ese tipo de mensaje.
Gestión de Suscripciones	Este comportamiento se encarga de recibir las peticiones de suscripción de otros agentes a los mensajes provenientes del Centro de Mensajes (y, por tanto, de un estudiante concreto) y de apuntarlas en la Tabla de Suscripciones.
Tabla de Suscripciones	Este elemento recoge las suscripciones de otros agentes a mensajes provenientes del estudiante al que representa.

B.2. Relaciones y sus propiedades

La relación que dirige la descomposición de esta vista es *forma-parte-de*. No hay excepciones ni añadidos a lo que define esta relación.

B.3. Interfaces

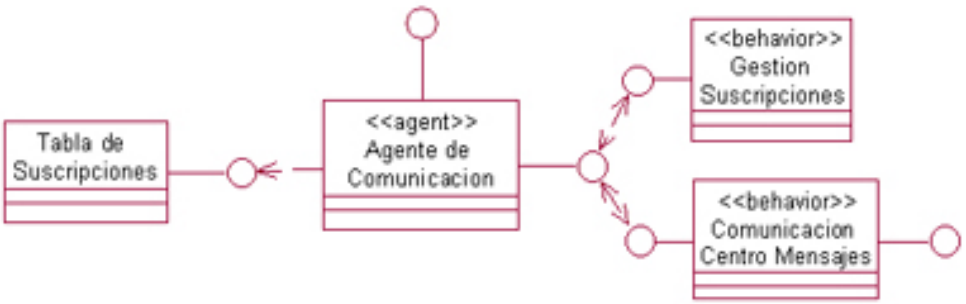

Figura 5.34: Interfaces del Agente de Comunicación

Agente de Comunicación	Proporciona interfaces para enviar y recibir mensajes de otros agentes. No asume la existencia de ningún otro agente.
Comunicación con el Centro de Mensajes	Proporciona servicios para mandar y recibir mensajes del Centro de Mensajes. Asume la existencia del Centro de Mensajes.
Gestión de Suscripciones	Proporciona servicios para suscribirse a mensajes por remitente y por tipo de contenido. También proporciona servicios para borrar suscripciones. Necesita la existencia de un elemento que almacene las suscripciones.
Tabla de Suscripciones	Proporciona servicios para añadir, borrar y modificar suscripciones.

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

Figura 5.35: Diagrama de contexto del Agente de Comunicación

D. Guía de variabilidad

No aplicable.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

Puesto que este agente no necesita entender el contenido de los mensajes, se ha decidido dotarlo únicamente de dos comportamientos, cada uno de los cuales se encarga de la comunicación con un tipo de elemento diferente: agentes y el Centro de Mensajes.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

No aplicable.

F. Otra información

No aplicable.

G. Vistas relacionadas

- Padre: 5.5.1.5
- Hermanas: ver secciones 5.5.1.6, 5.5.1.8, 5.5.1.9, 5.5.1.10, 5.5.1.11, 5.5.1.12, 5.5.1.13, 5.5.1.14
- Hijas: ninguna.

5.5.1.8. Vista de Descomposición Paquete 8: Agente de Simulación

A. Presentación de la vista

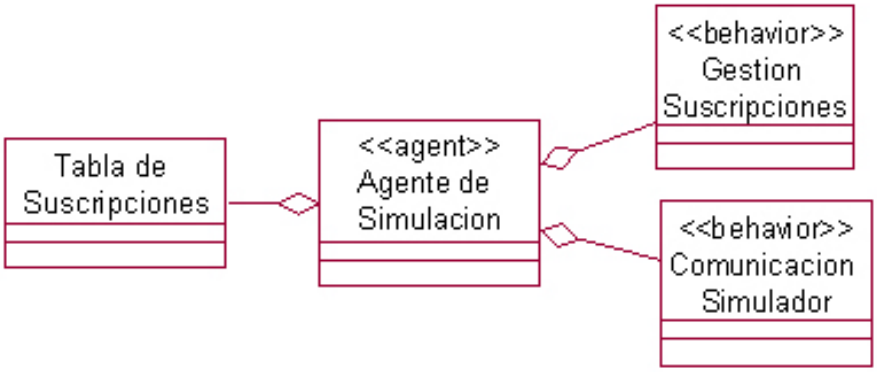

Figura 5.36: Descomposición del Agente de Simulación

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Agente de Simulación	Este elemento, aunque lleva el mismo nombre que el módulo que lo contiene, constituye en sí lo que es el agente, el cual tiene asociados una serie de comportamientos que llevan a cabo la mayor parte de las responsabilidades que le corresponden a éste. La responsabilidad principal de este elemento consiste en la creación y puesta en marcha inicial de los comportamientos, así como ejercer de representante de los demás elementos ante el resto de los agentes, enviando y recibiendo mensajes.

Comunicación con el Simulador	La responsabilidad de este comportamiento consiste en llevar a cabo la comunicación con el Simulador. De esta manera, cuando el agente reciba un mensaje para consultar el estado del Simulador o para enviarle una acción, este comportamiento será el encargado de ponerse en contacto con el simulador y devolver al agente correspondiente la respuesta obtenida, en caso de ser necesario.
Gestión de Suscripciones	Este comportamiento se encarga de recibir las peticiones de suscripción de otros agentes a los mensajes provenientes del Simulador y de apuntarlas en la Tabla de Suscripciones.
Tabla de Suscripciones	Este elemento recoge las suscripciones de otros agentes a mensajes provenientes del simulador.

B.2. Relaciones y sus propiedades

La relación que dirige la descomposición de esta vista es *forma-parte-de*. No hay excepciones ni añadidos a lo que define esta relación.

B.3. Interfaces

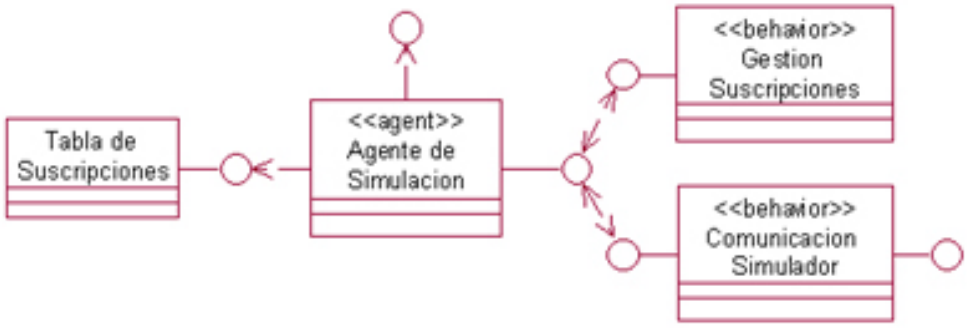

Figura 5.37: Interfaces del Agente de Simulación

Agente de Simulación	Proporciona interfaces para enviar y recibir mensajes de otros agentes. No asume la existencia de ningún otro agente.
Comunicación con el Simulador	Proporciona servicios para mandar y recibir mensajes del Simulador. Asume la existencia del Simulador.
Gestión de Suscripciones	Proporciona servicios para suscribirse a mensajes por remitente y por tipo de contenido. También proporciona servicios para borrar suscripciones. Necesita la existencia de un elemento que almacene las suscripciones.
Tabla de Suscripciones	Proporciona servicios para añadir, borrar y modificar suscripciones.

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

Figura 5.38: Diagrama de contexto del Agente de Simulación

D. Guía de variabilidad

El comportamiento de comunicación con el simulador es dependiente del simulador que se esté utilizando. De esta forma, cuando se desee cambiar el simulador, habrá que sustituir también este comportamiento por otro equivalente que posibilite la comunicación con el nuevo simulador.

Como se muestra en la vista correspondiente al Simulador(5.5.1.4), éste posee un módulo que realiza las funciones de intermediario entre la simulación

y el SIT. Llegado el momento de sustituir la simulación por otra, es posible que sólo sea necesario realizar cambios en el intermediario, y no en el comportamiento de comunicación con el simulador del agente de simulación. Sin embargo, como se ha mencionado, este aspecto es dependiente del simulador con el que se trabaje. Al no haber realizado un análisis exhaustivo de todos los tipos de simuladores que pueden utilizarse, no se puede afirmar de manera categórica que no vaya a ser necesario realizar cambios en el comportamiento del agente.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

El Agente de simulación, en su conjunto, es un agente que tiene como responsabilidad comunicar el simulador con el resto de los agentes, llevando a cabo una labor que bien puede identificarse con el patrón *Wrapper* descrito en (Hayden et al., 1999). Es por este motivo que las labores del agente en sí quedan reducidas a la traducción de información y a la comunicación con el Simulador y con los demás agentes. De cara al SIT, el Agente de Simulación realiza todas las tareas que lleva a cabo el Simulador, aunque, como se ha dicho, en realidad sólo constituye una forma de comunicar el simulador con el resto de los agentes.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

La plataforma de agentes que se va a utilizar para implementar la arquitectura es JADE, que funciona sobre el lenguaje de programación Java. Se ha supuesto que será posible comunicar la plataforma JADE con el lenguaje de programación utilizado para implementar el simulador, ya sea directamente o bien, por ejemplo, mediante la utilización de sockets.

F. Otra información

No aplicable.

G. Vistas relacionadas

- Padre: 5.5.1.5
- Hermanas: ver secciones 5.5.1.6, 5.5.1.7, 5.5.1.9, 5.5.1.10, 5.5.1.11, 5.5.1.12, 5.5.1.13, 5.5.1.14
- Hijas: ninguna.

5.5.1.9. Vista de Descomposición Paquete 9: Agente Experto

A. Presentación de la vista

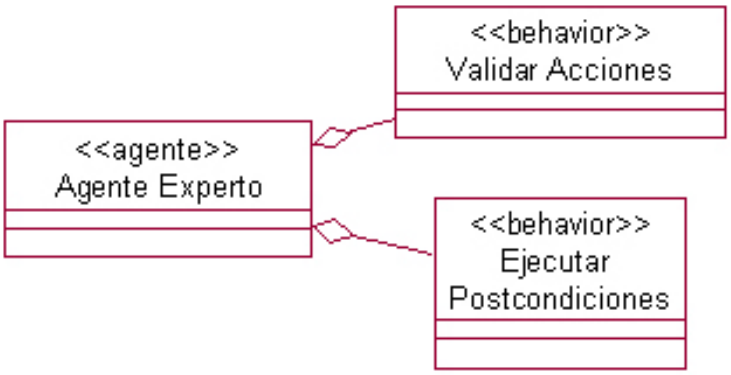

Figura 5.39: Descomposición del Agente Experto

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Agente Experto	Este elemento, aunque lleva el mismo nombre que el módulo que lo contiene, constituye en sí lo que es el agente, el cual tiene asociados una serie de comportamientos que llevan a cabo la mayor parte de las responsabilidades que le corresponden a éste. La responsabilidad principal de este elemento consiste en la creación y puesta en marcha inicial de los comportamientos, así como ejercer de representante de los demás elementos ante el resto de los agentes, enviando y recibiendo mensajes.
Validar Acciones	La responsabilidad de este comportamiento consiste en realizar la validación de las precondiciones de una acción. En caso necesario, deberá decidir si necesita información que le pueda proporcionar otro agente para realizar la validación.
Ejecutar Post-condiciones	La responsabilidad de este comportamiento es la de decidir qué postcondiciones de una acción son directamente relevantes para un estudiante o para otros agentes, y enviarle a cada uno las postcondiciones que le puedan interesar.

B.2. Relaciones y sus propiedades

La relación que dirige la descomposición de esta vista es *forma-parte-de*. No hay excepciones ni añadidos a lo que define esta relación.

B.3. Interfaces

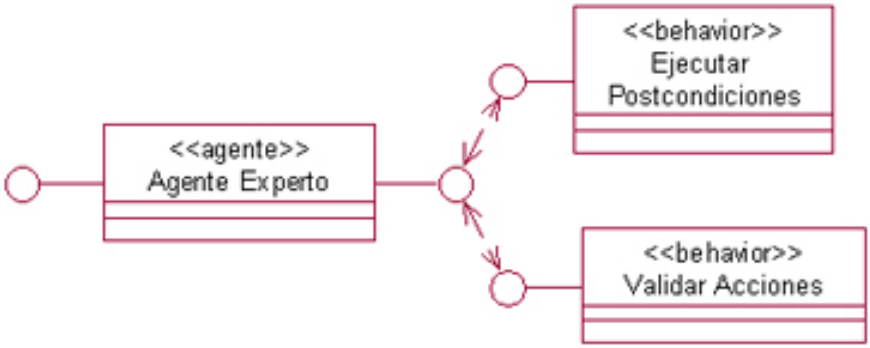

Figura 5.40: Interfaces del Agente Experto

Agente Experto	Proporciona interfaces para enviar y recibir mensajes de otros agentes. No asume la existencia de ningún otro agente.
Validar Acciones	Proporciona servicios para que le soliciten la validación de acciones enteras o de precondiciones. No asume la existencia de otros agentes, aunque puede necesitarlos para realizar la validación de alguna precondición.
Ejecutar Postcondiciones	Proporciona servicios para recibir acciones o postcondiciones que tratar. No asume la existencia de ningún agente.

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

Figura 5.41: Diagrama de contexto del Agente Experto

D. Guía de variabilidad

No aplicable.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

Puesto que las dos tareas principales de este agente consisten en la validación de las precondiciones de una acción y la ejecución de las postcondiciones de la misma, se ha tomado la decisión de realizar cada una de estas acciones en un comportamiento distinto.

La validación de las precondiciones depende de la información que manejan el Agente de Simulación y el Agente Mundo, por lo que, aunque por el momento se ha descartado, se contempla la posibilidad de separar la validación en dos comportamientos separados. De esta manera, entre otros aspectos, se posibilita la validación de ambos tipos de precondiciones en paralelo. No se ha hecho uso de esta opción porque, de momento, el número de precondiciones de una acción se prevé que sea pequeño, y la validación de cada precondición es, en principio, una operación rápida.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

No aplicable.

F. Otra información

No aplicable.

G. Vistas relacionadas

- Padre: 5.5.1.5
- Hermanas: ver secciones 5.5.1.6, 5.5.1.7, 5.5.1.8, 5.5.1.10, 5.5.1.11, 5.5.1.12, 5.5.1.13, 5.5.1.14
- Hijas: ninguna.

5.5.1.10. Vista de Descomposición Paquete 10: Agente Mundo

A. Presentación de la vista

Figura 5.42: Descomposición del Agente Mundo

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Agente Mundo	Este elemento, aunque lleva el mismo nombre que el módulo que lo contiene, constituye en sí lo que es el agente, el cual tiene asociados una serie de comportamientos que llevan a cabo la mayor parte de las responsabilidades que le corresponden a éste. La responsabilidad principal de este elemento consiste en la creación y puesta en marcha inicial de los comportamientos, así como ejercer de representante de los demás elementos ante el resto de los agentes, enviando y recibiendo mensajes.
Actualizar Elemento del Mundo	La responsabilidad de este comportamiento consiste en mantener actualizada la información que maneja el SIT acerca de los elementos presentes en el mundo virtual. Para ello, se encarga de recibir la información proporcionada por otros agentes y la estructura y almacena en la Ontología del Mundo.
Responder Consulta	La responsabilidad de este comportamiento es la de responder a las preguntas realizadas por otros agentes acerca de los elementos presentes en el mundo virtual. Estas preguntas pueden ser simples, como la consulta de la posición de un objeto, o más elaboradas, como la relación existente entre dos elementos del mundo. Las respuestas que sea capaz de ofrecer dependerán de la complejidad de la ontología que se maneja, así como del motor de inferencia que opere sobre ella.
Interfaz con la Ontología del Mundo	Este elemento se encarga de facilitar el intercambio de información entre el Agente Mundo y la Ontología del Mundo, así como de realizar las traducciones correspondientes entre uno y otro.

B.2. Relaciones y sus propiedades

La relación que dirige la descomposición de esta vista es *forma-parte-de*. No hay excepciones ni añadidos a lo que define esta relación.

B.3. Interfaces

Figura 5.43: Interfaces del Agente Mundo

Agente Mundo	Proporciona interfaces para enviar y recibir mensajes de otros agentes. No asume la existencia de ningún otro agente.
Actualizar Elemento del Mundo	Proporciona servicios para recibir información de actualización de otros agentes. Asume la existencia de algún elemento que almacene la información recibida.
Responder Consulta	Proporciona servicios para responder a las preguntas realizadas por otros agentes. Asume la existencia de un repositorio que almacena la información necesaria para responder las preguntas.
Interfaz con la Ontología del Mundo	Proporciona servicios para acceder a la Ontología del Mundo, tanto para realizar consultas sobre ella como para actualizar la información que contiene. Asume la existencia de la ontología.

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

Figura 5.44: Diagrama de contexto del Agente Mundo

D. Guía de variabilidad

No aplicable.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

Se ha decidido utilizar una ontología para almacenar la información del mundo en lugar de una base de datos fundamentalmente debido a que con la ontología resulta más sencillo realizar inferencias y razonamientos a partir de la información que contiene, mediante la utilización de un motor de inferencia. Además, también facilita la creación de nuevas relaciones entre los elementos de la ontología, lo cual da mayor flexibilidad a la hora de ofrecer a los alumnos respuestas más completas a sus preguntas sobre objetos del entorno.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

No aplicable.

F. Otra información

El diseño de la ontología se realizará con Protegé, y las inferencias se harán utilizando Jena. El diseño de la ontología y su utilización estarán restringidos a lo que permitan hacer ambas herramientas.

G. Vistas relacionadas

- Padre: 5.5.1.5
- Hermanas: ver secciones 5.5.1.6, 5.5.1.7, 5.5.1.8, 5.5.1.9, 5.5.1.11, 5.5.1.12, 5.5.1.13, 5.5.1.14
- Hijas: ninguna.

5.5.1.11. Vista de Descomposición Paquete 11: Agente de Planificación

A. Presentación de la vista

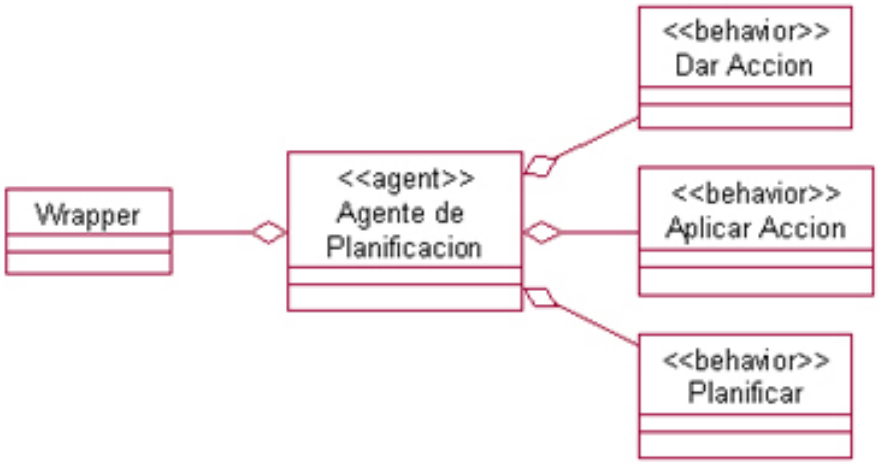

Figura 5.45: Descomposición del Agente de Planificación

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Agente de Planificación	Este elemento, aunque lleva el mismo nombre que el módulo que lo contiene, constituye en sí lo que es el agente, el cual tiene asociados una serie de comportamientos que llevan a cabo la mayor parte de las responsabilidades que le corresponden a éste. La responsabilidad principal de este elemento consiste en la creación y puesta en marcha inicial de los comportamientos, así como ejercer de representante de los demás elementos ante el resto de los agentes, enviando y recibiendo mensajes.
Dar Acción	La responsabilidad de este comportamiento consiste en entregar la siguiente acción que se debe ejecutar de acuerdo con el plan actualmente vigente.
Aplicar Acción	Este comportamiento se encarga de aplicar, sobre el estado actual del planificador, la acción que se le envíe, comprobando si es posible realizar la mencionada acción de acuerdo con sus precondiciones y el estado del plan.
Planificar	Este comportamiento se encarga de realizar un nuevo plan cuando se lo solicite algún otro agente, partiendo de una situación inicial para un problema dado o de la situación actual del plan en curso.
Wrapper	Este elemento se encarga de comunicar al Agente de Planificación con el planificador, realizando las labores de traducción necesarias entre ambos.

B.2. Relaciones y sus propiedades

La relación que dirige la descomposición de esta vista es *forma-parte-de*. No hay excepciones ni añadidos a lo que define esta relación.

B.3. Interfaces

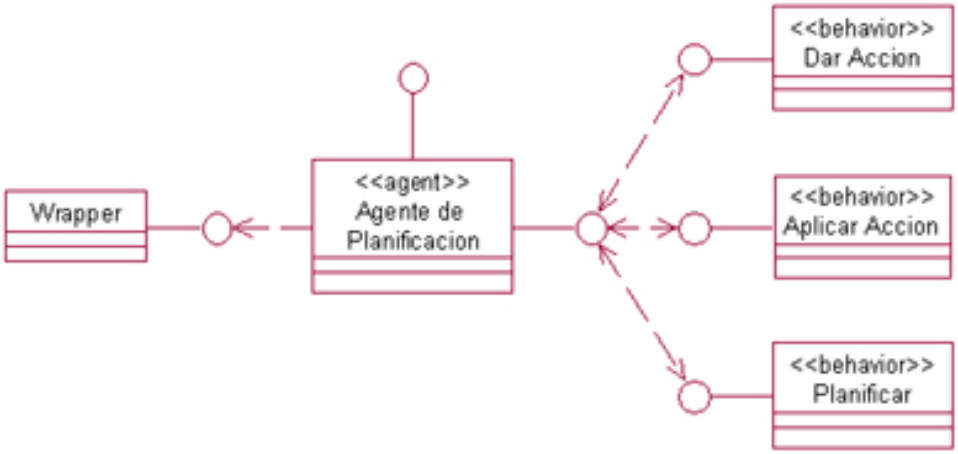

Figura 5.46: Interfaces del Agente de Planificación

Agente de Planificación	Proporciona interfaces para enviar y recibir mensajes de otros agentes. No asume la existencia de ningún otro agente.
Dar Acción	Proporciona servicios para que otros agentes soliciten la siguiente acción del plan. Asume la existencia de un elemento que elabora el mencionado plan.
Aplicar Acción	Proporciona servicios para que otros agentes soliciten la aplicación de una acción sobre el plan en curso. Asume la existencia de un elemento que elabora el mencionado plan.
Planificar	Proporciona servicios para que otros agentes soliciten la elaboración de un nuevo plan. Supone la existencia de un elemento que elabora el mencionado plan.
Wrapper	Proporciona servicios para que el Agente de Planificación se comunique con el planificador. Asume la existencia de su correspondiente planificador.

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

Figura 5.47: Diagrama de contexto del Agente de Planificación

D. Guía de variabilidad

El código del planificador que resuelve un problema debe ser cargado en tiempo de ejecución (ver apartado de "Otra información").

E. Información sobre la arquitectura

E.1. Decisiones de diseño

El aspecto más destacable de esta vista es el elemento denominado *Wrapper*. Como su propio nombre indica, es un envoltorio que proporciona un mecanismo de comunicación del agente con el planificador, al tiempo que oculta las peculiaridades del planificador al Agente de Planificación.

El propósito de utilizar el *Wrapper* consiste en facilitar el cambio del planificador por otro distinto intentando que, en la medida de lo posible, este cambio no afecte al Agente de Planificación y, mucho menos, al resto de los agentes. De esta manera, para cada planificador se desarrollaría un *Wrapper* específico que implementaría una interfaz común para todos estos envoltorios.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

No aplicable.

F. Otra información

La implementación del planificador se va a realizar utilizando JShop2, que es una implementación de un planificador Shop2 sobre el lenguaje Java. JShop2 tiene la peculiaridad de que, para mejorar la eficiencia del planificador, genera un código específico para cada plan realizado, lo que obliga a cargar este código en tiempo de ejecución.

G. Vistas relacionadas

- Padre: 5.5.1.5
- Hermanas: ver secciones 5.5.1.6, 5.5.1.7, 5.5.1.8, 5.5.1.9, 5.5.1.10, 5.5.1.12, 5.5.1.13, 5.5.1.14
- Hijas: ninguna.

5.5.1.12. Vista de Descomposición Paquete 12: Agente de Trayectoria

A. Presentación de la vista

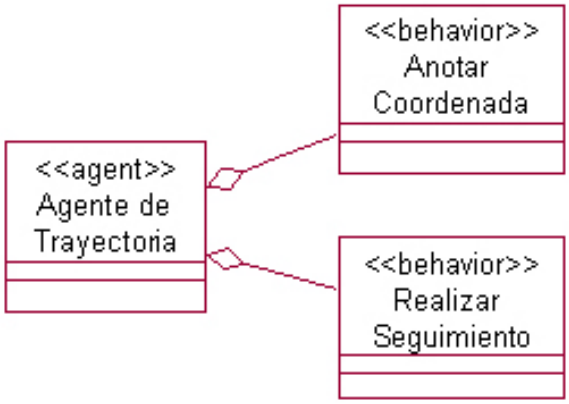

Figura 5.48: Descomposición del Agente de Trayectoria

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Agente de Trayectoria	Este elemento, aunque lleva el mismo nombre que el módulo que lo contiene, constituye en sí lo que es el agente, el cual tiene asociados una serie de comportamientos que llevan a cabo la mayor parte de las responsabilidades que le corresponden a éste. La responsabilidad principal de este elemento consiste en la creación y puesta en marcha inicial de los comportamientos, así como ejercer de representante de los demás elementos ante el resto de los agentes, enviando y recibiendo mensajes.
Anotar Coordenada	La responsabilidad de este comportamiento consiste en recibir las posiciones de un estudiante en el EV e ir guardándolas para que puedan ser usadas para realizar el seguimiento de la trayectoria del alumno.
Realizar Seguimiento	Este comportamiento está encargado de calcular la trayectoria entre dos puntos y compararla con la que realiza el estudiante.

B.2. Relaciones y sus propiedades

La relación que dirige la descomposición de esta vista es *forma-parte-de*. No hay excepciones ni añadidos a lo que define esta relación.

B.3. Interfaces

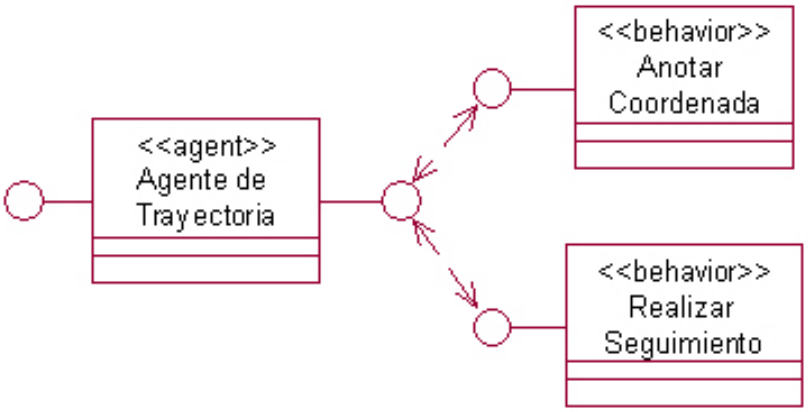

Figura 5.49: Interfaces del Agente de Trayectoria

Agente de Trayectoria	Proporciona interfaces para enviar y recibir mensajes de otros agentes. Asume la existencia de un agente que le proporciona las coordenadas con las que trabaja.
Anotar Coordenada	Proporciona un servicio para recibir las coordenadas de los desplazamientos del estudiante.
Realizar Seguimiento	Proporciona servicios para iniciar y detener el seguimiento de los desplazamientos de un estudiante, así como para facilitar el resultado de dicho seguimiento.

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

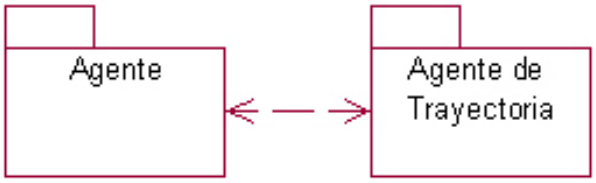

Figura 5.50: Diagrama de contexto del Agente de Trayectoria

D. Guía de variabilidad

El seguimiento de la trayectoria de los estudiantes es algo que no se va a realizar continuamente, sino sólo cuando el estudiante deba moverse. Por este motivo, el comportamiento de seguimiento debe poder activarse y desactivarse a voluntad.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

Se ha decidido separar en dos comportamientos la anotación de coordenadas y el seguimiento por si se desea deshabilitar este último pero manteniendo el registro de los desplazamientos del estudiante.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

Como ya se ha mencionado anteriormente, el seguimiento de la trayectoria del estudiante es uno de los elementos que puede hacer que la velocidad de ejecución de la aplicación disminuya. Por lo tanto, se ha supuesto que el algoritmo de seguimiento de la trayectoria del estudiante puede ejecutarse en tiempo real. Si es preciso, deberá disminuirse la precisión del algoritmo, trabajando con menos coordenadas y flexibilizando los criterios utilizados para decidir si la trayectoria del alumno se ajusta a una trayectoria ideal.

F. Otra información

No aplicable.

G. Vistas relacionadas

- Padre: 5.5.1.5
- Hermanas: ver secciones 5.5.1.6, 5.5.1.7, 5.5.1.8, 5.5.1.9, 5.5.1.10, 5.5.1.11, 5.5.1.13, 5.5.1.14
- Hijas: ninguna.

5.5.1.13. Vista de Descomposición Paquete 13: Agente de Estudiante

A. Presentación de la vista

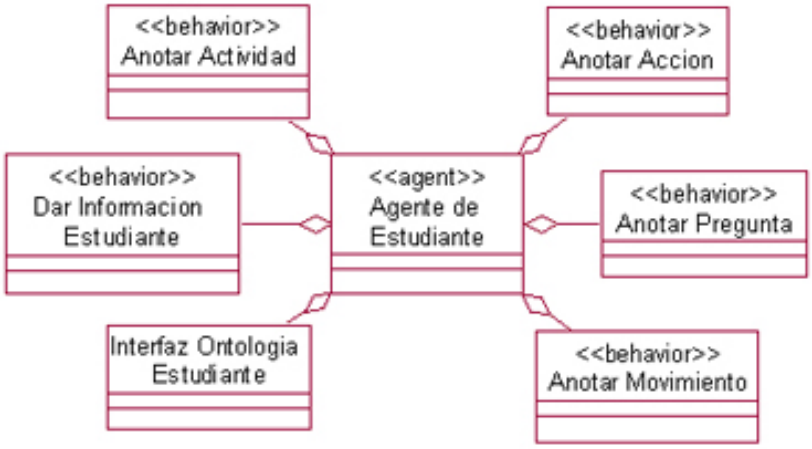

Figura 5.51: Descomposición del Agente de Estudiante

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Agente de Estudiante	Este elemento, aunque lleva el mismo nombre que el módulo que lo contiene, constituye en sí lo que es el agente, el cual tiene asociados una serie de comportamientos que llevan a cabo la mayor parte de las responsabilidades que le corresponden a éste. La responsabilidad principal de este elemento consiste en la creación y puesta en marcha inicial de los comportamientos, así como ejercer de representante de los demás elementos ante el resto de los agentes, enviando y recibiendo mensajes.
Anotar Acción	Este comportamiento recibe las acciones realizadas por un estudiante durante la realización de una actividad, junto con la evaluación de su corrección. Esta acción es almacenada y utilizada para completar el modelo del estudiante
Anotar Pregunta	La responsabilidad de este comportamiento consiste en almacenar las preguntas realizadas por un estudiante, junto con la evaluación de su pertinencia, así como en usar de esta información para completar el modelo del estudiante.
Anotar Movimiento	Este comportamiento se encarga de recibir información de los desplazamientos del estudiante, junto con su evaluación, para almacenarlo y utilizarlo en la realización del modelo del estudiante.
Anotar Actividad	Este comportamiento tiene como responsabilidad el almacenamiento y procesamiento de la evaluación de una actividad en su conjunto para completar la realización del modelo del estudiante.

Dar Información del Estudiante	La responsabilidad de este comportamiento consiste en proporcionar la información del estudiante contenida hasta el momento en el modelo del estudiante. La información que proporciona está relacionada con las acciones, preguntas y movimientos del estudiante, así como todo lo que se pueda inferir durante la realización del modelo del estudiante, como la capacidad de aprendizaje, la eficiencia en la realización de los ejercicios o el grado de autonomía del estudiante.
Interfaz con la Ontología del Estudiante	Este elemento se encarga de facilitar el intercambio de información entre el Agente de Estudiante y la Ontología del Estudiante, así como de realizar las traducciones correspondientes entre uno y otro.

B.2. Relaciones y sus propiedades

La relación que dirige la descomposición de esta vista es *forma-parte-de*. No hay excepciones ni añadidos a lo que define esta relación.

B.3. Interfaces

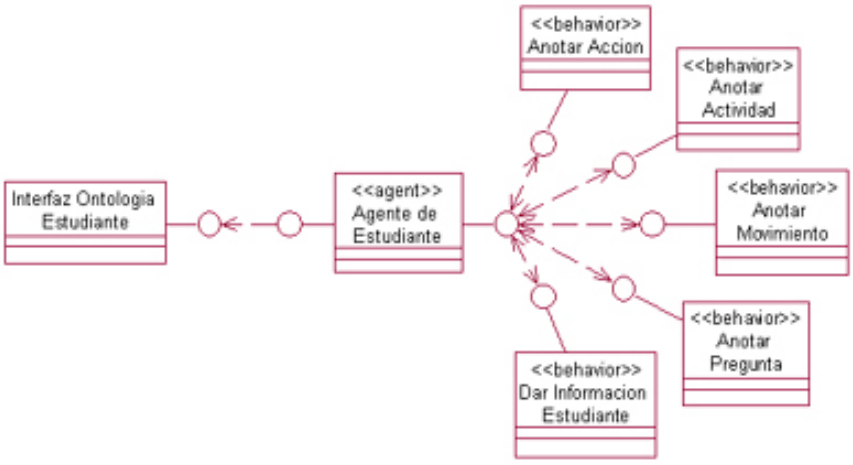

Figura 5.52: Interfaces del Agente de Estudiante

Agente de Estudiante	Proporciona interfaces para enviar y recibir mensajes de otros agentes. Asume la existencia de uno o varios agentes que le proporcionan la información de acciones, preguntas, movimientos y actividades que necesita para la realización del modelo del estudiante.
Anotar Acción	Proporciona servicios para recibir y analizar información acerca de las acciones de los estudiantes. Asume la existencia de algún elemento en el que almacenar la información recibida o inferida.
Anotar Pregunta	Proporciona servicios para recibir y analizar información acerca de las preguntas de los estudiantes. Asume la existencia de algún elemento en el que almacenar la información recibida o inferida.
Anotar Movimiento	Proporciona servicios para recibir y analizar información acerca de los movimientos de los estudiantes. Asume la existencia de algún elemento en el que almacenar la información recibida o inferida.
Anotar Actividad	Proporciona servicios para recibir y analizar información acerca de las actividades de los estudiantes. Asume la existencia de algún elemento en el que almacenar la información recibida o inferida.
Dar Información del Estudiante	Proporciona servicios para consultar la información contenida en el modelo del estudiante. Asume la existencia de algún elemento que almacena la información que debe proporcionar.
Interfaz con la Ontología del Estudiante	Proporciona servicios para acceder a la Ontología del Estudiante, tanto para realizar consultas sobre ella como para almacenar la información que contiene. Asume la existencia de la ontología.

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

Figura 5.53: Diagrama de contexto del Agente de Estudiante

D. Guía de variabilidad

Existirá un agente de estudiante por cada estudiante conectado al sistema, de modo que cada agente mantendrá por separado el modelo de cada uno de los estudiantes.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

Aunque el movimiento se puede considerar como una acción dentro del EV, se ha decidido considerarlo de manera separada porque la evaluación que hay que realizar sobre el mismo es diferente a la del resto de las acciones, y los agentes involucrados pueden ser distintos.

Se ha considerado la posibilidad de que el modelado de todos los estudiantes lo realizase un único agente, aunque finalmente se ha optado por que cada estudiante tenga su propio agente de estudiante. De esta manera, entre otros beneficios, se espera facilitar la distribución de los agentes en distintas máquinas en caso de que sea necesario tomar esta medida. También se simplifica el diseño del modelo del estudiante y se facilita el traslado del perfil del estudiante entre distintos cursos e, incluso, entre distintas plataformas.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

No aplicable.

F. Otra información

No aplicable.

G. Vistas relacionadas

- Padre: 5.5.1.5
- Hermanas: ver secciones 5.5.1.6, 5.5.1.7, 5.5.1.8, 5.5.1.9, 5.5.1.10, 5.5.1.11, 5.5.1.12, 5.5.1.14
- Hijas: ninguna.

5.5.1.14. Vista de Descomposición Paquete 14: Agente de Tutoría

A. Presentación de la vista

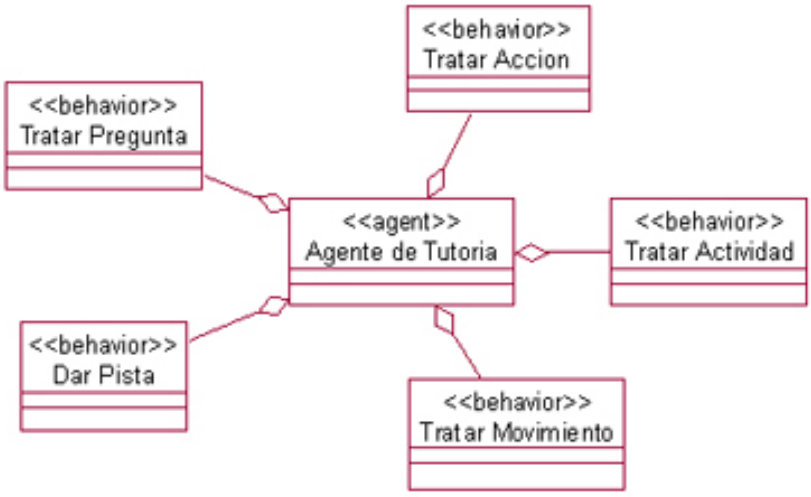

Figura 5.54: Descomposición del Agente de Tutoría

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Agente de Tutoría	Este elemento constituye el agente en sí, el cual tiene asociados unos comportamientos que llevan a cabo la mayor parte de las responsabilidades que le corresponden. La responsabilidad principal de este elemento consiste en la creación y puesta en marcha de los comportamientos, así como ejercer de representante de los demás elementos ante el resto de los agentes, enviando y recibiendo mensajes.
Tratar Acción	Este comportamiento se encarga de recibir y evaluar las acciones realizadas por el estudiante para resolver una actividad. Debe comprobar si es una de las acciones que pueden llevarse a cabo para seguir con el ejercicio o, si no, si es posible realizarla. En caso de ser así, debe decidir, en función de la estrategia de tutoría, si permite la realización de la acción o si le indica al alumno que la acción no es posible.
Tratar Actividad	La responsabilidad de este comportamiento consiste en evaluar en su conjunto una actividad realizada por un alumno tras su finalización, decidiendo si se ha completado de manera satisfactoria o no.
Tratar Pregunta	Cuando un alumno realiza una pregunta, este comportamiento es el encargado de evaluar su adecuación y decidir si se responde o no, así como el nivel de detalle de la respuesta.
Tratar Movimiento	Este comportamiento está encargado de evaluar la adecuación de la trayectoria de un estudiante.
Dar Pista	Este comportamiento está encargado de monitorizar el comportamiento del alumno. Si se cumplen las condiciones marcadas en la estrategia de tutoría en uso, se detectará que el alumno no sabe cómo continuar y se le ofrecerán pistas con distinto nivel de detalle para ayudarle a decidir cómo tiene que continuar la ejecución la actividad.

B.2. Relaciones y sus propiedades

La relación que dirige la descomposición de esta vista es *forma-parte-de*. No hay excepciones ni añadidos a lo que define esta relación.

B.3. Interfaces

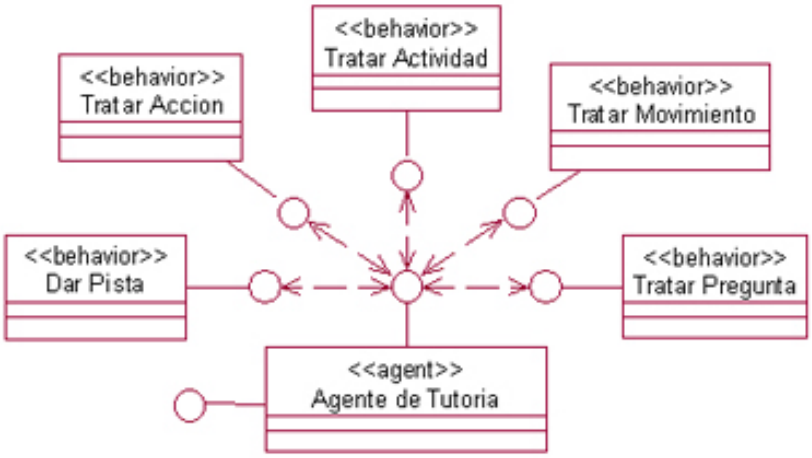

Figura 5.55: Interfaces del Agente de Tutoría

Agente de Tutoría	Proporciona interfaces para enviar y recibir mensajes de otros agentes. Asume la existencia de otros agentes que le proporcionan información con la que trabajar.
Tratar Acción	Proporciona servicios para recibir las acciones realizadas por el estudiante. Asume la existencia de agentes que le ayudan a verificar la corrección o aplicabilidad de la acción ejecutada.
Tratar Actividad	Proporciona servicios para recibir notificaciones del comienzo y finalización de las actividades. Asume la existencia de algún agente que almacena información sobre lo acontecido en la realización de una actividad.
Tratar Pregunta	Proporciona servicios para recibir las preguntas realizadas por el estudiante. Asume la existencia de un agente que proporciona la respuesta en función de los parámetros presentes en la estrategia de tutoría.

Tratar Movimiento	Proporciona servicios para recibir los movimientos del estudiante. Asume la existencia de un agente que analiza la trayectoria seguida y proporciona una valoración para poder realizar la evaluación.
Dar Pista	Asume la existencia de algún agente que registra las acciones del estudiante.

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

Figura 5.56: Diagrama de contexto del Agente de Tutoría

D. Guía de variabilidad

Algunos de los comportamientos descritos pueden no estar presentes si así lo determinan la estrategia de tutoría, el estudiante o la actividad a ejecutar. Así, el comportamiento de tratar movimientos no se ejecutará en actividades que no requieran un desplazamiento por el EV. El comportamiento de dar pistas no se ejecutará si la estrategia de tutoría así lo determina o si el estudiante solicita que se supriman las pistas. También es posible que se supriman las respuestas ante las preguntas realizadas por el estudiante.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

La identificación de los comportamientos se ha realizado tratando de tener en cuenta las propiedades de dinamismo definidas en el apartado anterior.

Existe la posibilidad de activar y desactivar, ya sea de forma dinámica o no, muchas de las funcionalidades proporcionadas por el tutor. Disponer de cada una de esas funcionalidades en un comportamiento separado del agente facilita la labor de poder activarlas y desactivarlas de manera independiente, sin más que activar o desactivar el comportamiento que se encarga de realizarlas. Además de los aspectos mencionados con las pistas o los movimientos, se permite que no se respondan preguntas si, por ejemplo, no se ha definido esa funcionalidad en un dominio o para un problema determinado. De igual forma, se puede deshabilitar el tratamiento de acciones en caso de que exista un modo no supervisado en el que el estudiante pueda explorar el EV.

Por otro lado, dadas las características y funcionalidades de este agente, depende en gran medida de otros agentes presentes en el entorno, que deben ayudarle a validar las acciones del estudiante, almacenar la traza de las acciones que realiza, registrar y analizar sus movimientos o proporcionarle la respuesta a sus preguntas. De esta forma, la estrategia de tutoría puede proporcionar a los comportamientos del agente acciones por omisión ante la inexistencia de alguno de los agentes necesarios para llevar sus responsabilidades a buen fin. Así, por ejemplo, puede no permitirse la ejecución de una acción que no sea estrictamente la que corresponda según el plan o, por el contrario, puede permitirse la ejecución de cualquier acción que sea físicamente posible dentro del EV.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

No aplicable.

F. Otra información

No aplicable.

G. Vistas relacionadas

- Padre: 5.5.1.5
- Hermanas: ver secciones 5.5.1.6, 5.5.1.7, 5.5.1.8, 5.5.1.9, 5.5.1.10, 5.5.1.11, 5.5.1.12, 5.5.1.13
- Hijas: ninguna.

5.5.2. Vista de Uso

La vista de uso muestra la dependencia entre distintos elementos de la arquitectura en lo que concierne a la relación de uso que existe entre ellos. Esta relación muestra qué elementos hacen uso de otros, entendiendo que P_1 usa a P_2 si la corrección de P_1 depende del correcto funcionamiento de P_2.

En un sistema como el que nos ocupa, esta definición es de poca utilidad, porque resulta complicado definir cuál es el funcionamiento correcto de muchos de los elementos que se manejan. Por este motivo, se va a interpretar la relación de uso de manera algo más amplia, considerando que un elemento usa a otro si necesita de su existencia para funcionar.

Debe aclararse que no es necesario que un elemento utilice directamente a otro para que se establezca una relación de uso, sino que la relación entre ambos puede existir de manera indirecta, a través de un tercero.

Por omisión, cualquier comportamiento necesita que exista un agente al cual pertenece y que se encarga de ponerlo en marcha y de enviar y recibir los mensajes que ese comportamiento pueda necesitar. Por este motivo, se ha decidido no incluir esta relación de uso en ninguno de los paquetes, por ser inherente a agentes y comportamientos.

5.5.2.1. Vista de Uso
Paquete 1: Sistema Inteligente de Tutoría

A. Presentación de la vista

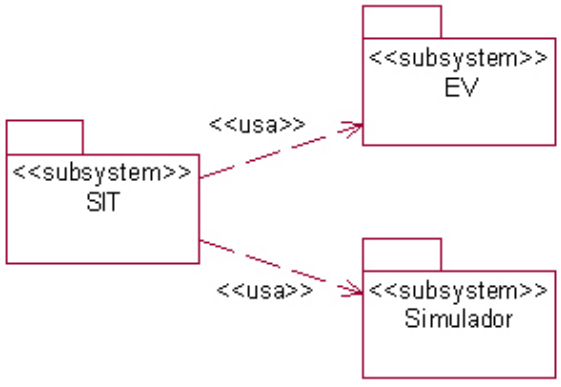

Figura 5.57: Módulos usados por el Sistema Inteligente de Tutoría

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Los elementos presentados en esta vista son los mismos que se pueden encontrar en la vista de descomposición del sistema (sección 5.5.1.1).

B.2. Relaciones y sus propiedades

La relación que se establece entre los módulos de esta vista es *usa*. Un elemento usa a otro si el correcto funcionamiento del primero depende del correcto funcionamiento del segundo. No es suficiente con que el primero realice una llamada al segundo, ni tampoco necesario.

B.3. Interfaces

Las interfaces entre los distintos módulos de esta vista son las descritas en la vista de descomposición del sistema (sección 5.5.1.1).

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

El contexto de este paquete es el mostrado en la vista de descomposición del sistema (sección 5.5.1.1).

D. Guía de variabilidad

No aplicable.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

Aunque la comunicación entre los distintos elementos del sistema se produce a través del Centro de Mensajes, éste no es más que un medio para llevar a cabo dicha comunicación, por lo que la relación de uso efectiva se produce entre el Sistema Inteligente de Tutoría y los dos módulos restantes: Entorno Virtual y Simulador.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

Como se ha mencionado anteriormente, se ha supuesto que tanto el Simulador como el Entorno Virtual tienen al SIT como un posible usuario más, si bien no necesitan de él para funcionar, razón por la cual la relación de uso se produce en un solo sentido.

F. Otra información

No aplicable.

G. Vistas relacionadas

- Ver sección 5.5.1.1.

5.5.2.2. Vista de Uso
Paquete 2: Agente de Tutoría

A. Presentación de la vista

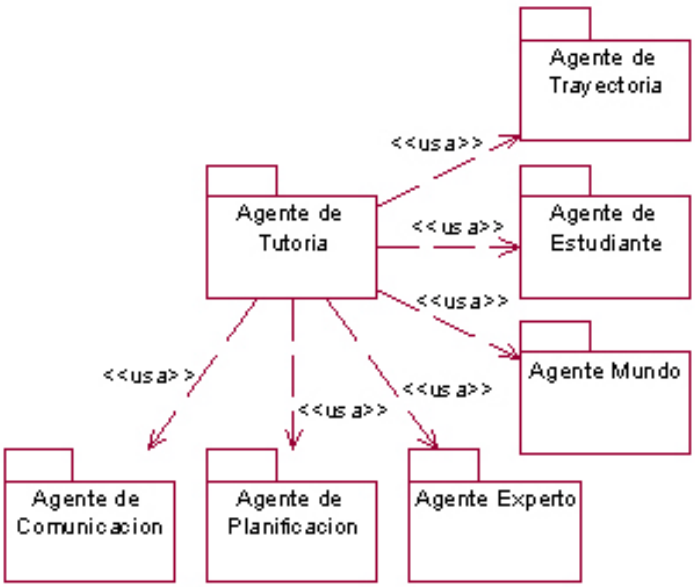

Figura 5.58: Módulos usados por el Agente de Tutoría

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Los elementos presentados en esta vista son los mismos que pueden encontrarse en la vista de descomposición del Sistema Inteligente de Tutoría (sección 5.5.1.5).

B.2. Relaciones y sus propiedades

La relación que se establece entre los módulos de esta vista es *usa*. Un elemento usa a otro si el correcto funcionamiento del primero depende del correcto funcionamiento del segundo. No es suficiente con que el primero realice una llamada al segundo, ni tampoco necesario.

Como decisión de diseño, se ha optado por que los agentes realicen acciones por omisión o ignoren ciertos datos en caso de no existir quien los proporcione, lo que posibilita que la relación de uso se reduzca al mínimo. No obstante, se van a mostrar las relaciones de uso como serían sin las acciones por omisión. De esta manera se da una visión real de las necesidades de los agentes para que el sistema funcione con todas sus características.

B.3. Interfaces

Las interfaces entre los distintos módulos de esta vista son las descritas en la vista de descomposición correspondiente (sección 5.5.1.5).

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

El contexto de este paquete es el mostrado en la vista de descomposición del Sistema Inteligente de Tutoría (sección 5.5.1.5).

D. Guía de variabilidad

No aplicable.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

En los escenarios de modificabilidad planteados para la construcción del sistema se establece la necesidad de que ciertas funcionalidades puedan

deshabilitarse o incluso no existir. Por este motivo, aunque la relación de uso existe cuando las distintas funcionalidades están activas, el agente de tutoría no debe necesitar la presencia de los demás agentes si no va a precisar de sus servicios. Por ello, la relación entre agentes debe establecerse en tiempo de ejecución y sólo a partir del momento en que el agente de tutoría necesite los servicios de otro. Además, este conocimiento no debe establecerse *a priori*, sino a través de un mecanismo como las páginas amarillas, que permiten cambiar de manera dinámica a un agente que preste un determinado servicio por otro. Por otro lado, aunque la relación se establece entre agentes, deben ser los comportamientos del agente de tutoría los que, de manera individual, soliciten establecer contacto con los agentes que necesiten.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

No aplicable.

F. Otra información

No aplicable.

G. Vistas relacionadas

- Ver secciones 5.5.1.5, 5.5.1.14.

5.5.2.3. Vista de Uso
Paquete 3: Agente de Estudiante

A. Presentación de la vista

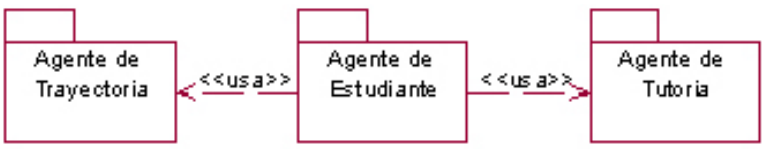

Figura 5.59: Módulos usados por el Agente de Estudiante

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Los elementos presentados en esta vista son los mismos que pueden encontrarse en la vista de descomposición del Sistema Inteligente de Tutoría (sección 5.5.1.5).

B.2. Relaciones y sus propiedades

La relación que se establece entre los módulos de esta vista es *usa*. Un elemento usa a otro si el correcto funcionamiento del primero depende del correcto funcionamiento del segundo. No es suficiente con que el primero realice una llamada al segundo, ni tampoco necesario.

Como decisión de diseño, se ha optado por que los agentes realicen acciones por omisión o ignoren ciertos datos en caso de no existir quien los proporcione, lo que posibilita que la relación de uso se reduzca al mínimo. No obstante, se van a mostrar las relaciones de uso como serían sin las acciones por omisión. De esta manera se da una visión real de las necesidades de los agentes para que el sistema funcione con todas sus características.

B.3. Interfaces

Las interfaces entre los distintos módulos de esta vista son las descritas en la vista de descomposición correspondiente (sección 5.5.1.5).

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

El contexto de este paquete es el mostrado en la vista de descomposición del Sistema Inteligente de Tutoría (sección 5.5.1.5).

D. Guía de variabilidad

No aplicable.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

El agente de estudiante resulta fundamental para el buen funcionamiento de un sistema inteligente de tutoría, ya que es en gran medida quien posibilita

la adaptación del entrenamiento a las necesidades del estudiante. Por ese motivo, su presencia suele darse por descontada, y es habitual que todos los datos relativos a los alumnos se le envíen directamente. Sin embargo, manteniendo la filosofía de diseño del sistema, y contemplando la posibilidad de que distintos modelos de estudiante necesiten distinta información para su elaboración, se ha optado por que sea el agente de estudiante, por indicación de sus comportamientos, quien solicite la información necesaria para elaborar el modelo del estudiante que se encuentre a su cargo.

Se ha decidido que la información se reciba a través del agente de tutoría y del de trayectoria (previa localización a través de las páginas amarillas) en lugar de hacerlo directamente a través del de comunicación. El motivo es que así se mantiene la consistencia en el tratamiento tanto de las acciones como de los desplazamientos del estudiante, y porque es responsabilidad de ambos agentes evaluar la corrección de las acciones del estudiante.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

No aplicable.

F. Otra información

No aplicable.

G. Vistas relacionadas

- Ver secciones 5.5.1.5, 5.5.1.13.

5.5.2.4. Vista de Uso
Paquete 4: Agente de Trayectoria

A. Presentación de la vista

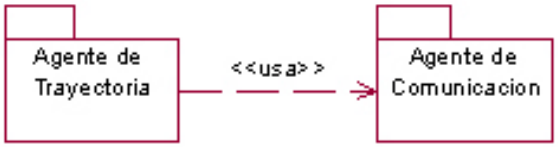

Figura 5.60: Módulos usados por el Agente de Trayectoria

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Los elementos presentados en esta vista son los mismos que pueden encontrarse en la vista de descomposición del Sistema Inteligente de Tutoría (sección 5.5.1.5).

B.2. Relaciones y sus propiedades

La relación que se establece entre los módulos de esta vista es *usa*. Un elemento usa a otro si el correcto funcionamiento del primero depende del correcto funcionamiento del segundo. No es suficiente con que el primero realice una llamada al segundo, ni tampoco necesario.

Como decisión de diseño, se ha optado por que los agentes realicen acciones por omisión o ignoren ciertos datos en caso de no existir quien los proporcione, lo que posibilita que la relación de uso se reduzca al mínimo. No obstante, se van a mostrar las relaciones de uso como serían sin las acciones por omisión. De esta manera se da una visión real de las necesidades de los agentes para que el sistema funcione con todas sus características.

B.3. Interfaces

Las interfaces entre los distintos módulos de esta vista son las descritas en la vista de descomposición correspondiente (sección 5.5.1.5).

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

El contexto de este paquete es el mostrado en la vista de descomposición del Sistema Inteligente de Tutoría (sección 5.5.1.5).

D. Guía de variabilidad

No aplicable.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

Al igual que con otros agentes, se ha optado por que sea el agente que necesita una información quien la solicite, en lugar de que sea el agente que

la proporciona quien se la ofrezca. Por este motivo, la relación de uso es del agente de trayectoria hacia el de comunicación, y no al contrario, debido a que es éste quien solicita las posiciones del estudiante para poder realizar el seguimiento de la trayectoria. Para ello utiliza el mecanismo de edición-suscripción que se describe en la sección 5.5.3.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

No aplicable.

F. Otra información

No aplicable.

G. Vistas relacionadas

- Ver secciones 5.5.1.5, 5.5.1.12.

5.5.2.5. Vista de Uso
Paquete 5: Agente Mundo

A. Presentación de la vista

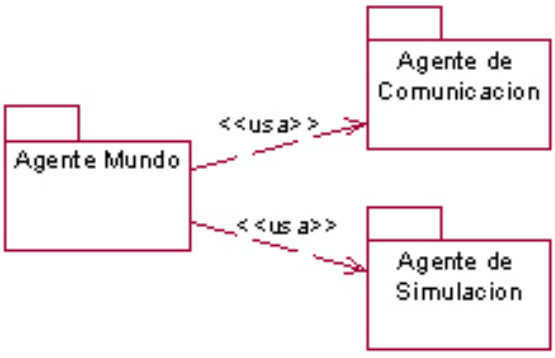

Figura 5.61: Módulos usados por el Agente Mundo

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Los elementos presentados en esta vista son los mismos que pueden encontrarse en la vista de descomposición del Sistema Inteligente de Tutoría (sección 5.5.1.5).

B.2. Relaciones y sus propiedades

La relación que se establece entre los módulos de esta vista es *usa*. Un elemento usa a otro si el correcto funcionamiento del primero depende del correcto funcionamiento del segundo. No es suficiente con que el primero realice una llamada al segundo, ni tampoco necesario.

Como decisión de diseño, se ha optado por que los agentes realicen acciones por omisión o ignoren ciertos datos en caso de no existir quien los proporcione, lo que posibilita que la relación de uso se reduzca al mínimo. No obstante, se van a mostrar las relaciones de uso como serían sin las acciones por omisión. De esta manera se da una visión real de las necesidades de los agentes para que el sistema funcione con todas sus características.

B.3. Interfaces

Las interfaces entre los distintos módulos de esta vista son las descritas en la vista de descomposición correspondiente (sección 5.5.1.5).

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

El contexto de este paquete es el mostrado en la vista de descomposición del Sistema Inteligente de Tutoría (sección 5.5.1.5).

D. Guía de variabilidad

No aplicable.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

La principal responsabilidad del agente mundo consiste en mantener un modelo actualizado del estado del entorno de entrenamiento, de forma

que el resto de los agentes puedan solicitarle esta información cuando la necesiten. De tal forma, las fuentes de cambio en el estado del mundo son los sucesos que puedan ocurrir dentro del propio entorno de entrenamiento, producidos, por ejemplo, por acciones de los estudiantes, y los cambios en elementos controlados por la simulación. La forma de obtener información sobre estos cambios es, por tanto, a través del agente de simulación y de los de comunicación.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

No aplicable.

F. Otra información

No aplicable.

G. Vistas relacionadas

- Ver secciones 5.5.1.5, 5.5.1.10.

5.5.2.6. Vista de Uso
Paquete 6: Agente Experto

A. Presentación de la vista

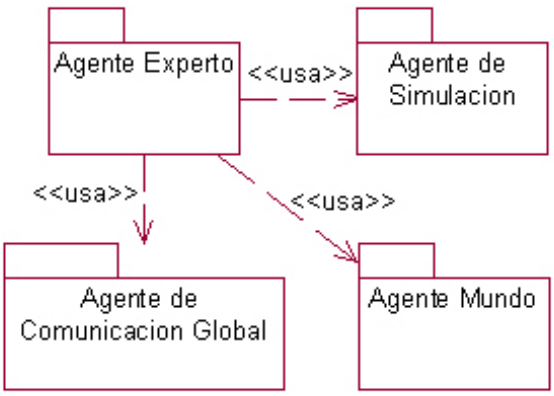

Figura 5.62: Módulos usados por el Agente Experto

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Los elementos presentados en esta vista son los mismos que pueden encontrarse en la vista de descomposición del Sistema Inteligente de Tutoría (sección 5.5.1.5).

B.2. Relaciones y sus propiedades

La relación que se establece entre los módulos de esta vista es *usa*. Un elemento usa a otro si el correcto funcionamiento del primero depende del correcto funcionamiento del segundo. No es suficiente con que el primero realice una llamada al segundo, ni tampoco necesario.

Como decisión de diseño, se ha optado por que los agentes realicen acciones por omisión o ignoren ciertos datos en caso de no existir quien los proporcione, lo que posibilita que la relación de uso se reduzca al mínimo. No obstante, se van a mostrar las relaciones de uso como serían sin las acciones por omisión. De esta manera se da una visión real de las necesidades de los agentes para que el sistema funcione con todas sus características.

B.3. Interfaces

Las interfaces entre los distintos módulos de esta vista son las descritas en la vista de descomposición correspondiente (sección 5.5.1.5).

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

El contexto de este paquete es el mostrado en la vista de descomposición del Sistema Inteligente de Tutoría (sección 5.5.1.5).

D. Guía de variabilidad

No aplicable.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

Las responsabilidades del agente experto están relacionadas con las acciones realizadas por los estudiantes. Por una parte, debe comprobar que

la acción que el estudiante intenta realizar puede llevarse a cabo. Para ello, debe comprobar tanto que las condiciones del mundo permiten ejecutar la acción, como que el estado de la simulación también lo hace posible. Para realizar cada una de estas verificaciones necesita los servicios proporcionados por el agente mundo y el agente de simulación.

Por otra parte, una vez que se ha considerado que la acción puede llevarse a cabo, si dicha acción tiene efectos que se le deban mostrar a los alumnos, será responsabilidad del agente experto enviar las notificaciones pertinentes, lo que hará a través del agente de comunicación global.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

No aplicable.

F. Otra información

No aplicable.

G. Vistas relacionadas

- Ver secciones 5.5.1.5, 5.5.1.9.

5.5.2.7. Vista de Uso
Paquete 7: Agente de Simulación

A. Presentación de la vista

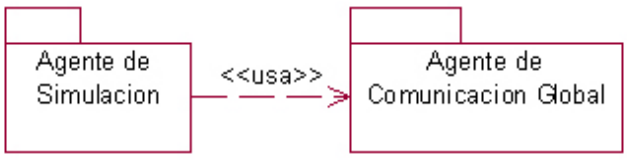

Figura 5.63: Módulos usados por el Agente de Simulación

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Los elementos presentados en esta vista son los mismos que pueden encontrarse en la vista de descomposición del Sistema Inteligente de Tutoría (sección 5.5.1.5).

B.2. Relaciones y sus propiedades

La relación que se establece entre los módulos de esta vista es *usa*. Un elemento usa a otro si el correcto funcionamiento del primero depende del correcto funcionamiento del segundo. No es suficiente con que el primero realice una llamada al segundo, ni tampoco necesario.

Como decisión de diseño, se ha optado por que los agentes realicen acciones por omisión o ignoren ciertos datos en caso de no existir quien los proporcione, lo que posibilita que la relación de uso se reduzca al mínimo. No obstante, se van a mostrar las relaciones de uso como serían sin las acciones por omisión. De esta manera se da una visión real de las necesidades de los agentes para que el sistema funcione con todas sus características.

B.3. Interfaces

Las interfaces entre los distintos módulos de esta vista son las descritas en la vista de descomposición correspondiente (sección 5.5.1.5).

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

El contexto de este paquete es el mostrado en la vista de descomposición del Sistema Inteligente de Tutoría (sección 5.5.1.5).

D. Guía de variabilidad

No aplicable.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

La responsabilidad del agente de simulación consiste en notificar los cambios que se producen en la simulación para que, en caso necesario, se

muestren en el EV. Para ello, envía la notificación de estos cambios a los alumnos a través del agente de comunicación global.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

No aplicable.

F. Otra información

No aplicable.

G. Vistas relacionadas

- Ver secciones 5.5.1.5, 5.5.1.8.

5.5.3. Vista Editor-Suscriptor

Entre las tácticas mencionadas en (Bass et al., 2003) para aumentar la modificabilidad de un sistema software se encuentra la utilización de un comportamiento del tipo editor-subscriptor. Esto es así principalmente por dos razones. En primer lugar, se consigue que los productores y consumidores de información se conozcan en tiempo de ejecución, pero no antes, lo cual permite una modificación más sencilla de los diseños. Por otro lado, también se desacoplan los productores y consumidores de información, que desconocen su identidad y, por tanto, detalles que puedan comprometer a uno de ellos cuando el otro es modificado - salvo, por supuesto, los relativos al servicio al que se suscriben.

Para documentar esta vista se supone la existencia de un servicio de páginas amarillas, donde los productores van a anunciar los servicios que proporcionan y donde los consumidores buscarán los servicios que necesiten para llevar a cabo las tareas que tienen encomendadas. Esta suposición está soportada por la existencia del mencionado servicio dentro del estándar FIPA para la arquitectura de plataformas multiagente (IEEE-FIPA, 2002a).

Todos los agentes del sistema van a anunciar en las páginas amarillas los servicios - o información - que pueden proporcionar a otros agentes. Sin embargo, estos anuncios no siempre van a dar lugar a suscripciones. En los casos en los que un agente necesite conocer un cambio siempre que se produzca sí tendrá lugar la mencionada subscripción. Por el contrario, cuando un agente esté interesado en un servicio sólo en determinadas ocasiones, lo que

hará en lugar de suscribirse y descartarlo cuando no lo necesite será solicitarlo expresamente cuando así lo desee. En cualquiera de los dos casos, el agente que realiza el anuncio desconoce quiénes son los consumidores ni si, a priori, van a suscribirse al servicio o no. Por su parte, los consumidores no conocen expresamente al agente que realiza el anuncio, sino que lo hacen a través de lo que aparece en las páginas amarillas, por lo que éste agente podría cambiar.

Dadas las características del sistema objeto de estudio, se asume que el tipo de los agentes permanece inalterado durante la ejecución, y los proveedores de los servicios tampoco van a variar durante la misma. Aunque no supondría una dificultad, con esta restricción se evita tener que comprobar continuamente si el agente al que solicitar un servicio ha cambiado, lo cual redunda en una pequeña mejora en la eficiencia de la aplicación resultante.

5.5.3.1. Vista Editor-Suscriptor Paquete 1: Sistema

A. Presentación de la vista

Una de las necesidades que se busca satisfacer con esta arquitectura es poder conectar y desconectar los diferentes subsistemas que la forman sin que los demás elementos de la arquitectura se vean afectados.

Por ello, se ha pensado que una buena solución consiste en la utilización de un mecanismo de edición-suscripción que funcione a través de un centro de mensajes, que constituye el único elemento conocido por los demás. Su función principal consiste en esperar notificaciones de otras aplicaciones que desean recibir determinado tipo de mensajes y reenviar los mensajes recibidos a los destinatarios que se hayan suscrito a ellos. Así, si alguno de los elementos del sistema no es necesario para la realización de una actividad porque ninguno de sus servicios es requerido por los demás subsistemas, se podrá detener su ejecución sin que el resto note ninguna diferencia. Si a esto unimos el hecho de que el Centro de Mensajes ignora todos los detalles relativos al resto de aplicaciones, puesto que lo único que hace es enviar y recibir mensajes sin preocuparse de quién los envía, quién los recibe o qué contienen, se puede esperar un grado de acoplamiento bastante bajo entre los distintos subsistemas que conforman el sistema de entrenamiento.

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Cada uno de los elementos que conforman el sistema proporcionan los servicios indicados en la siguiente tabla.

Elemento	**Servicio**	**Propósito**
Centro de Mensajes	Suscribirse a mensaje	Proporciona la posibilidad de que cualquier sistema se conecte con el Centro de Mensajes para recibir mensajes enviados por otras aplicaciones conectadas a él. Aunque el uso habitual es suscribirse para recibir uno o varios tipos de mensajes en función del tipo de contenido, también es posible realizar suscripciones en función del emisor del mensaje.
	Borrar suscripción a mensaje	Cuando una aplicación no desea recibir más mensajes de un tipo al que se ha suscrito previamente, puede utilizar este servicio para que dejen de serle enviados. Así, si una aplicación se va a cerrar, el comportamiento esperado es que borre todas las suscripciones que realizó. Si no se procede de esta manera, cuando el Centro de Mensajes detecte errores en los envíos procederá a borrar las suscripciones correspondientes.
Sistema Inteligente de Tutoría	Registrar estudiante	El propósito de este servicio es indicar al sistema de tutoría la existencia de un nuevo estudiante, para que pueda integrarlo dentro de una actividad y monitorizar sus acciones.
	Registrar acción	El propósito de este servicio es monitorizar las acciones del estudiante para poder realizar su seguimiento. Este servicio se presta por el hecho de que el estudiante se haya registrado en el sistema. La respuesta ante una acción está en función de la estrategia de tutoría en uso.
	Registrar posición	El propósito de este servicio es hacer un seguimiento de los desplazamientos del estudiante por el entorno de entrenamiento. Este servicio se presta por el hecho de que el estudiante se haya registrado en el sistema, aunque puede no estar activo si la actividad objeto de entrenamiento no requiere desplazamientos. La respuesta variará en función de la estrategia de tutoría.

	Contestar pregunta	El propósito de este servicio es que el estudiante pueda solicitar ayuda cuando lo necesite. Este servicio se presta por el hecho de que el estudiante se haya registrado en el sistema. La respuesta variará en función de la estrategia de tutoría.
Simulador	Realizar acción	El propósito de este servicio es que otras aplicaciones puedan solicitar al simulador que se realice una determinada acción dentro de la simulación.
	Cambiar posición	El propósito de este servicio es que una aplicación externa pueda solicitarle al simulador el cambio en la posición de un elemento de la simulación.
	Consultar estado	El propósito de este servicio es que una aplicación pueda consultar el estado de un elemento de la simulación.
	Notificar cambio	Este servicio está pensado para que el simulador notifique cambios en elementos de la simulación cada vez que se produzcan. Este servicio sí necesita suscripción.
Entorno Virtual	Realizar acción	Este servicio permite que una aplicación externa solicite al EV que un elemento del mismo, habitualmente un avatar, realice una determinada acción dentro del entorno.
	Cambiar posición	Este servicio permite que una aplicación solicite al EV el cambio en la posición de un objeto o personaje que se encuentra dentro de éste.
	Mostrar mensaje	Este servicio permite que se solicite al EV que reproduzca un mensaje dentro del mismo, lo que posibilita, por ejemplo, recibir las contestaciones del SIT a las preguntas de los alumnos.

En la siguiente tabla se muestra, para cada elemento del sistema, quién le proporciona los servicios que necesita y para qué los utiliza.

Elemento	**Servicio**	**Propósito**
Centro de Mensajes		
		El propósito del centro de mensajes, como su nombre indica, es distribuir los mensajes a los que se suscriben el resto de elementos. El conocimiento que tiene acerca de ellos no va más allá de la capacidad de enviarles los mensajes a los que se han suscrito, por lo que no utiliza servicios de ninguna otra aplicación que pueda formar parte del sistema de entrenamiento.
Sistema Inteligente de Tutoría		
Centro de Mensajes	Suscribirse a mensaje	Mediante este servicio, el SIT se suscribe a la recepción de los mensajes de los tipos que le interesen y que sean enviados a través del Centro de Mensajes.
	Borrar suscripción a mensaje	Mediante este servicio, el SIT notifica que no quiere continuar recibiendo un determinado tipo de mensaje que le llega a través del Centro de Mensajes.
Simulador	Realizar acción	Cuando el SIT necesita ejecutar una acción que puede tener efectos sobre la simulación, provoca que se ejecute este servicio de manera que el estado de la simulación se actualice convenientemente.
	Cambiar posición	Cuando el SIT necesita modificar la posición de un elemento presente en la simulación, provoca la ejecución de este servicio de manera que el estado de la simulación se actualice en consecuencia.
	Consultar estado	Cuando el SIT necesita conocer el estado de un elemento de la simulación, hace uso de este servicio para solicitar la información que precise conocer.

	Notificar cambio	Si el SIT necesita conocer de forma continuada los cambios que se van produciendo en determinados elementos de la simulación, solicita que este servicio se los vaya notificando cuando sucedan.
Entorno Virtual	Realizar acción	Cuando el SIT necesita ejecutar una acción que puede tener efectos sobre el EV, provoca que se ejecute este servicio de manera que el estado del mismo se actualice convenientemente.
	Cambiar posición	Cuando el SIT necesita modificar la posición de un elemento presente en el EV, provoca que se ejecute este servicio de manera que el estado del mismo se actualice en consecuencia.
	Mostrar mensaje	Cuando el SIT necesita enviar mensajes a un alumno que se está entrenando a través del uso del EV, provoca la ejecución de este servicio de manera que el mensaje le pueda ser comunicado al alumno de la manera más adecuada dentro de las capacidades del EV.
Simulador		
Centro de Mensajes	Suscribirse a mensaje	Mediante este servicio, el Simulador se suscribe a la recepción de los mensajes de los tipos que le interesen y que sean enviados a través del Centro de Mensajes.
	Borrar suscripción a mensaje	Mediante este servicio, el SIT notifica que no quiere continuar recibiendo un determinado tipo de mensaje que le llega a través del Centro de Mensajes.
Sistema Inteligente de Tutoría	Registrar acción	Cuando el simulador realiza una acción que puede tener efecto en el proceso de tutoría, provoca la ejecución de este servicio para que el SIT tome en consideración lo que ha sucedido en la simulación.

	Registrar posición	Cuando el simulador cambia la posición de un objeto que puede tener efecto en el proceso de tutoría, provoca la ejecución de este servicio para que el SIT tome en consideración lo que ha sucedido en la simulación.
Entorno Virtual	Realizar acción	Cuando el Simulador necesita ejecutar una acción que puede tener efectos sobre el EV, provoca que se ejecute este servicio de manera que el estado del mismo se actualice convenientemente.
	Cambiar posición	Cuando el Simulador necesita modificar la posición de un elemento presente en el EV, provoca que se ejecute este servicio de manera que el estado del mismo se actualice en consecuencia.
Entorno Virtual		
Centro de Mensajes	Suscribirse a mensaje	Mediante este servicio, el EV se suscribe a la recepción de los mensajes de los tipos que le interesen y que sean enviados a través del Centro de Mensajes.
	Borrar suscripción a mensaje	Mediante este servicio, el SIT notifica que no quiere continuar recibiendo un determinado tipo de mensaje que le llega a través del Centro de Mensajes.
Simulador	Realizar acción	Cuando el EV necesita ejecutar una acción que puede tener efectos sobre la simulación, provoca que se ejecute este servicio de manera que el estado de la simulación se actualice convenientemente.
	Cambiar posición	Cuando el EV necesita modificar la posición de un elemento presente en la simulación, provoca que se ejecute este servicio de manera que el estado de la simulación se actualice en consecuencia.
	Notificar cambio	Si el EV necesita conocer de forma continuada los cambios que se van produciendo en determinados elementos de la simulación, solicita que este servicio se los vaya notificando.

Sistema Inteligente de Tutoría	Registrar estudiante	Cuando un estudiante inicia el EV y desea tomar parte en un entrenamiento, se le comunica al SIT para que tome las acciones pertinentes para que el estudiante pueda llevar a cabo el mencionado entrenamiento.
	Registrar acción	Cuando el usuario del EV realiza una acción que puede tener efecto en el proceso de tutoría, provoca que se ejecute este servicio para que el SIT tome en consideración lo que ha sucedido en la simulación.
	Registrar posición	Cuando el simulador cambia la posición de un objeto que puede tener efecto en el proceso de tutoría, provoca que se ejecute este servicio para que el SIT tome en consideración lo que ha sucedido en la simulación.
	Contestar pregunta	Cuando el estudiante que utiliza el EV realiza una pregunta, el EV provoca la ejecución de este servicio del SIT para que proporcione al estudiante la respuesta adecuada.

B.2. Relaciones y sus propiedades

La relación existente entre los elementos del sistema, característica de esta vista, es de uso del Centro de Mensajes como *bus* que distribuye mensajes entre el resto de componentes.

B.3. Interfaces

Las interfaces que aparecen en esta vista tienen relación con el uso del Centro de Mensajes como mecanismo de intercambio de información entre el resto de los subsistemas de la arquitectura.

De esta manera, el Centro de Mensajes proporciona un servicio para recibir mensajes de los restantes subsistemas, y espera de ellos que sean capaces de recibir los mensajes que les envíe.

Para que un subsistema pueda recibir mensajes, primero debe suscribirse a los tipos de mensajes que desea recibir. Para borrar una suscripción, primero debe haberse suscrito a los mensajes que desea dejar de recibir. La suscripción y la desuscripción pueden realizarse en cualquier momento.

Para que una aplicación pueda enviar información a otras aplicaciones a través del Centro de Mensajes, lo único que tiene que hacer es enviar el mensaje correspondiente, sin necesidad de ejecutar ningún paso previo.

El formato del mensaje que se intercambia debe incluir el remitente del mismo, el destinatario (sólo en caso necesario), el tipo del mensaje y la información específica que se desea enviar en el mensaje. El tipo del mensaje es la principal información que el Centro de Mensajes utilizará para gestionar las suscripciones.

B.4. Comportamientos

En la figura que se muestra a continuación se puede ver el comportamiento que debe tener el sistema al arrancar, momento en el que se producirán la mayor parte de las suscripciones en el Centro de Mensajes.

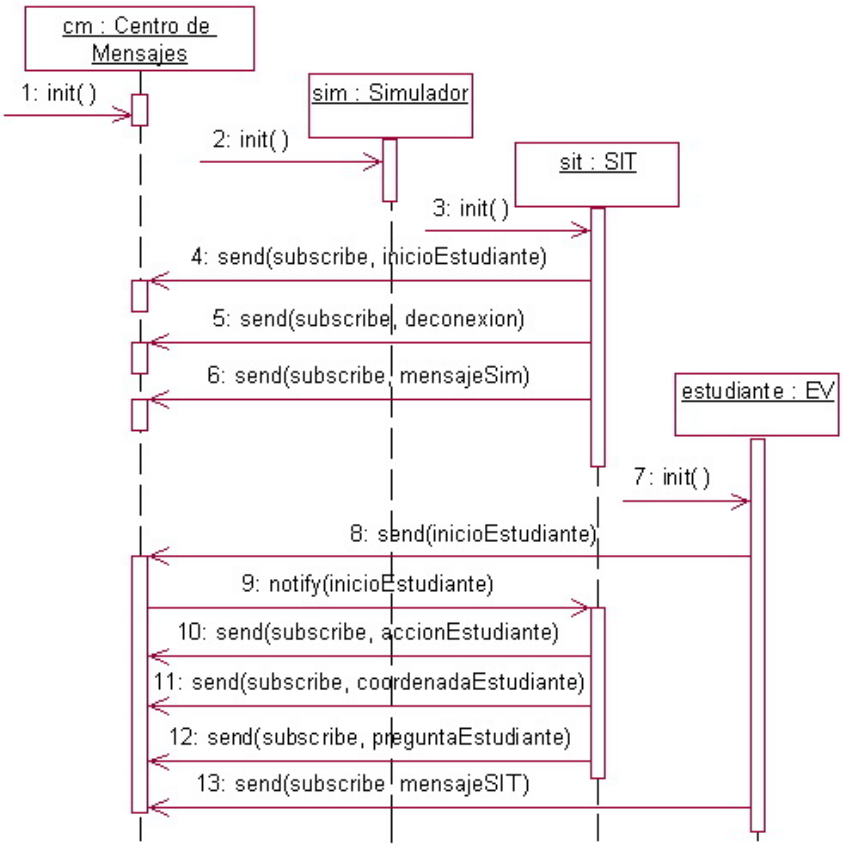

Figura 5.64: Inicio del Sistema

En primer lugar debe iniciarse el Centro de Mensajes, para que así esté disponible cuando el resto de subsistemas se inicien y comiencen a realizar sus suscripciones. El siguiente en arrancar debe ser el Simulador, siempre y cuando nos encontremos en el caso en que es un simulador independiente que no espera una gran interacción con otras aplicaciones. En caso de que el Simulador esté integrado en el EV o en el SIT, su inicio deberá producirse al mismo tiempo. A continuación debe iniciarse el SIT, que se suscribe a los mensajes de inicio y finalización de sesión de un estudiante, y quedará a la espera de que éstos se conecten. Finalmente, cada estudiante que se conecte dará lugar a que el SIT se suscriba a los mensajes que él envíe, y a su vez el EV del estudiante pedirá recibir los mensajes provenientes del SIT. Estas últimas suscripciones pueden realizarse en paralelo, no sucediendo lo mismo con las anteriores.

C. Diagrama de contexto

Ver el diagrama de contexto mostrado en la sección 5.5.1.3.

D. Guía de variabilidad

Como se ha mencionado anteriormente, el propósito al introducir el Centro de Mensajes consiste en desacoplar lo máximo posible el resto de elementos del sistema. Por este motivo, los subsistemas pueden enviar mensajes al Centro de Mensajes sin saber si alguien va o no a recibirlos, lo que, a su vez, posibilita que cualquier subsistema pueda suscribirse y borrarse en cualquier momento de la recepción de un tipo de mensaje, sin que el resto de subsistemas se vean afectados por ello.

Lo único que se presupone, tanto de los subsistemas como del Centro de Mensajes, es que son capaces de enviar y recibir los mensajes con el formato descrito en la sección de interfaces. De esta manera, en cualquier momento debe ser posible incluir nuevas aplicaciones que envíen o reciban mensajes sin que ello afecte al resto de subsistemas.

Esto debe, además, ser de utilidad durante el desarrollo, ya que se puede prescindir de alguna de las aplicaciones para realizar pruebas, sustituyéndola, por ejemplo, por una consola. También puede resultar de utilidad para monitorizar todos los mensajes que pasan por el Centro de Mensajes, lo cual puede servir para depurar el funcionamiento de la aplicación.

Por otro lado, como se ha descrito en las vistas relativas al Simulador, la naturaleza de éste puede ser bastante distinta según los casos. La documentación de esta vista se ha realizado pensando que el simulador se comunicará con el resto de las aplicaciones a través del Centro de Mensajes,

pero no se ha tenido en cuenta la información que pueda circular por otros cauces, ya sea de forma directa con el SIT o con el EV.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

En un principio se pensó en la utilización de un mecanismo directo de suscripción, por el cual cada subsistema recibiría directamente del resto la información que necesitase, sin hacer uso del centro de mensajes. Esta solución tiene la desventaja de que hace necesario saber qué aplicaciones existen para poder solicitarles la información deseada.

En lugar de ello se optó por la utilización del Centro de Mensajes, que hace posible que cada aplicación conozca únicamente la existencia de éste, aunque los mensajes provengan originalmente de una amplia variedad de fuentes. Además, también se consigue que el envío y la recepción de mensajes sea asíncrona, independizando aún más a los productores y consumidores de información.

Los productores (que hacen las veces de editores) no tienen necesidad de anunciar la información que ofrecen, mientras que los consumidores (que hacen las veces de suscriptores) deben informar al Centro de Mensajes del tipo de mensajes que deseen recibir. Es posible hacer esto así por el hecho de encontrarnos en un sistema cerrado, de manera que siempre podemos controlar los tipos de información que se producen y se consumen.

El funcionamiento resulta entonces similar al de la *invocación implícita*, ya que un mensaje enviado por una aplicación actúa como un evento que hace que reaccionen todos los suscriptores que se han apuntado a la recepción de ese evento.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

En todo momento se ha evitado asumir que los elementos de la aplicación vayan a estar desarrollados con el mismo lenguaje de programación o con lenguajes que puedan integrarse directamente, como permiten hacer los métodos nativos de Java.

Además, se supone que las aplicaciones pueden ser pesadas y que pueden existir varios usuarios conectados a la vez a la aplicación de entrenamiento, lo cual hace bastante probable que las distintas aplicaciones deban funcionar dentro de una red.

Debido a los factores mencionados, se ha asumido que las aplicaciones estarán desarrolladas en lenguajes que permitan la comunicación a través de una red, preferiblemente a través de sockets, y que la comunicación con el Centro de Mensajes se realizará a través de este mecanismo. Por este motivo, salvo que los lenguajes de programación permitan hacerlo de otra forma, también se supone que el contenido de los mensajes que se intercambien estará basado en tipos de datos simples, como números y cadenas de caracteres.

F. Otra información

Cuando se ha documentado que un elemento de la arquitectura usa un determinado servicio de otro elemento, debe quedar claro que la llamada a ese servicio no se produce de manera explícita, puesto que al Centro de Mensajes sólo se envían mensajes con información. Lo que se pretende es que el envío de un mensaje provoque la ejecución del servicio que se *usa*.

G. Vistas relacionadas

Ver secciones 5.5.1.1 a 5.5.1.5 y 5.5.2.1.

5.5.3.2. Vista Editor-Suscriptor Paquete 2: Plataforma de Agentes

A. Presentación de la vista

La mera utilización de agentes hace suponer a muchos autores y desarrolladores que el sistema tratado va a ser más flexible que otros sistemas construidos mediante otras tecnologías. Pero para que esto sea cierto no basta con utilizar agentes, sino que es necesario hacer uso de determinadas herramientas que éstos nos proporcionan.

Como se ha mencionado anteriormente, el uso de un mecanismo de edición-suscripción es una de las técnicas que contribuyen a obtener un sistema más modificable. En este sentido es donde un sistema basado en agentes puede ayudarnos a conseguir la modificabilidad deseada, ya que, si proporciona un servicio de páginas amarillas, facilitará la labor de implementación del citado mecanismo de edición-suscripción.

En este paquete se va a mostrar qué servicios proporciona cada agente. Dichos servicios serán anunciados en las páginas amarillas para que puedan ser utilizados por el resto de agentes. Aunque no todos los servicios requieran una suscripción, el hecho de que sean anunciados en las páginas amarillas permitirá que otros agentes los usen sin necesidad de saber a priori qué agente

los proporciona. También se muestran los servicios que ejecuta cada agente como respuesta a la información que recibe a través de una suscripción.

B. Catálogo de elementos

B.1. Elementos y sus propiedades

En la siguiente tabla se muestra, para cada agente, qué servicios ofrece cada uno de sus comportamientos, que son los encargados de ofrecer dichos servicios.

Elemento	**Servicio**	**Propósito**
Agente de Comunicación Global		
Comunicación con el Centro de Mensajes	Notificar conexión	Este servicio ofrece información acerca de nuevas conexiones de estudiantes al sistema, de manera que puedan ser tenidos en cuenta por el resto de los agentes del SIT que estén interesados en ello.
	Notificar desconexión	Al igual que el anterior servicio, se encarga de informar de las desconexiones de los estudiantes para que los demás agentes estén informados y puedan obrar en consecuencia.
	Enviar información	Este servicio ofrece la posibilidad de que un agente envíe información a cualquier elemento que se encuentre fuera del SIT, como un mensaje que afecte a todos los alumnos conectados. Este servicio no necesita suscripción para ser utilizado.
	Recibir información	Este servicio se encarga de recibir información enviada desde el Centro de Mensajes al SIT.
Gestión de Suscripciones	Solicitar información	Este servicio ofrece la posibilidad de que un agente solicite recibir información procedente del exterior del SIT. Se encarga tanto de registrar la suscripción en la tabla de suscripciones del agente como de registrarla en el Centro de Mensajes, en caso de ser necesario. Este servicio no necesita suscripción para ser utilizado.

Agente de Comunicación		
Comunicación con el Centro de Mensajes	Enviar información	Este servicio permite que un agente envíe información al estudiante cuya comunicación gestiona este agente, como pueden ser los mensajes de respuesta a una pregunta. Este servicio no necesita suscripción para ser utilizado.
	Recibir información	Este servicio se encarga de recibir la información relativa a un estudiante enviada desde el Centro de Mensajes al SIT.
Gestión de Suscripciones	Solicitar información	Este servicio ofrece la posibilidad de que un agente solicite recibir información que llega desde el estudiante con el que se comunica este agente. Se encarga tanto de registrar la suscripción en la tabla de suscripciones del agente como de registrarla en el Centro de Mensajes, en caso de ser necesario. Este servicio no necesita suscripción para ser utilizado.
Agente de Simulación		
Comunicación con el Simulador	Enviar información	Este servicio ofrece la posibilidad de que un agente envíe información al simulador. Este servicio no necesita suscripción para ser utilizado.
	Ejecutar acción	Este servicio permite que se envíe al simulador una acción que debe ejecutarse en él. También se encarga de gestionar la respuesta a la acción que devuelve el simulador. Este servicio no necesita suscripción para ser usado.
	Recibir información	Este servicio se encarga de recibir información enviada desde el simulador al SIT.

Gestión de Suscripciones	Solicitar información	Este servicio ofrece la posibilidad de que un agente solicite recibir directamente información que llega desde el simulador. Se encarga de registrar la suscripción en la tabla de suscripciones del agente. Este servicio no necesita suscripción para ser utilizado.
Agente Experto		
Validar Acciones	Validar acción	El propósito de este servicio es validar una acción realizada por un estudiante, de manera que se compruebe si es posible llevarla a cabo en función del estado actual de la actividad que se está entrenando. No necesita suscripción.
	Validar precondición	Este servicio sirve para comprobar si se cumple una única precondición de una acción para verificar si puede llevarse a cabo. No necesita suscripción.
Ejecutar Post-condiciones	Ejecutar acción	Este servicio se utiliza para ejecutar una acción y actualizar el estado de una actividad. No necesita suscripción.
	Ejecutar postcondición	Este servicio se utiliza para ejecutar una única postcondición de una acción y actualizar así el estado de una actividad. No necesita suscripción.
Agente Mundo		
Actualizar Elemento del Mundo	Actualizar posición	Este servicio se utiliza para actualizar la posición de un elemento dentro del mundo, ya sea un personaje o un objeto. No necesita suscripción.
	Actualizar estado	Este servicio se utiliza para actualizar otros atributos de un objeto o personaje distintos a su posición. No necesita suscripción.

Responder Consulta	Consultar	Este servicio se utiliza para realizar cualquier consulta de los atributos de un objeto o personaje, incluyendo su posición. No necesita suscripción.
	Validar precondición	Este servicio permite validar la precondición de una acción realizada por un alumno, siempre que esté relacionada con su ubicación o estado dentro del EV. No necesita suscripción.
Agente de Planificación		
Dar Acción	Dar acción siguiente	Este servicio devuelve la que, de acuerdo con el plan establecido por el planificador, es la siguiente acción correcta que debería ejecutarse en una actividad. No necesita suscripción.
Aplicar Acción	Aplicar acción	Este servicio sirve para aplicar una acción sobre un plan para que se actualice el estado actual del mismo.
Planificar	Planificar	Este servicio se utiliza para solicitar al agente que se elabore un nuevo plan para una actividad dada. No necesita suscripción.
	Replanificar	Este servicio se utiliza para solicitar que se elabore un nuevo plan para la actividad en curso. No necesita suscripción.
Agente de Trayectoria		
Anotar Coordenada	Anotar coordenada	Este servicio se encarga de registrar las nuevas coordenadas de la posición de un estudiante para que pueda realizarse el seguimiento de sus desplazamientos por el EV. No necesita suscripción.
Realizar Seguimiento	Iniciar seguimiento	Este servicio se encarga de solicitar el envío de las coordenadas de un alumno y de calcular la trayectoria óptima que debería seguir para moverse entre dos puntos del EV. No necesita suscripción.

	Realizar seguimiento	Este servicio se encarga de comparar la trayectoria que realiza el alumno con la trayectoria que se supone que debería seguir. No necesita suscripción.
	Finalizar seguimiento	Este servicio se encarga de solicitar la finalización del envío de coordenadas del alumno, así como de enviar el resultado de la evaluación de la trayectoria seguida por el mismo. No necesita suscripción.
Agente de Estudiante		
Anotar Acción	Anotar acción	Mediante este servicio, el Agente de Estudiante realiza una traza de las acciones que un estudiante lleva a cabo dentro de una actividad. No se registra ninguna evaluación de la acción. No necesita suscripción.
	Anotar evaluación de acción	El propósito de este servicio es registrar una acción llevada a cabo por el estudiante junto con la evaluación de su corrección. No necesita suscripción.
Anotar Pregunta	Anotar pregunta	Este servicio sirve para que el agente de estudiante esté informado de las preguntas que realiza el estudiante, aunque no se almacena información alguna relativa a su evaluación. No necesita suscripción.
	Anotar evaluación de pregunta	El propósito de este servicio es proporcionar al agente de estudiante información acerca de una pregunta realizada por un estudiante, junto con una evaluación de la misma. No necesita suscripción.
Anotar Movimiento	Anotar movimiento	Este servicio proporciona al Agente de Estudiante información relativa a los movimientos del estudiante por el EV. No incluye ningún tipo de evaluación de los mismos. No necesita suscripción.

	Anotar evaluación de movimiento	Este servicio sirve para que el Agente de Estudiante reciba información acerca de los movimientos del estudiante por el EV, junto con una evaluación de los mismos. No necesita suscripción.
Anotar Actividad	Anotar actividad	El propósito de este servicio es que el Agente de Estudiante reciba información acerca de la actividad sobre la que el estudiante está realizando el entrenamiento. No incluye una evaluación de la misma. No necesita suscripción.
	Anotar evaluación de actividad	Este servicio sirve para que el Agente de Estudiante reciba información acerca de la actividad objeto de entrenamiento junto con una evaluación de la misma. No necesita suscripción.
Dar Información del Estudiante	Dar información de estudiante	Este servicio es el medio por el cual otros agentes pueden consultar al Agente de Estudiante información relativa al estudiante del que está a cargo. La información que se le puede requerir viene dada en función del modelo de estudiante concreto que se esté utilizando. No necesita suscripción.
Agente de Tutoría		
Tratar Acción	Tratar acción	Este servicio recibe una acción realizada por un estudiante y se encarga de comprobar si se puede llevar a cabo y si es correcta. En función de ello, la evalúa y le envía el resultado a los agentes encargados del tratamiento de esta información. Para realizar este envío es necesario que los agentes interesados se suscriban a la recepción de esta información.

Tratar Actividad	Tratar actividad	Este servicio recibe información de inicio y finalización de una actividad y se encarga de realizar una evaluación de la misma para cada estudiante y de enviar la información resultante a los agentes interesados en ella. Es necesario que los agentes interesados se suscriban a la recepción de esta información.
Tratar Movimiento	Tratar movimiento	Este servicio recibe los movimientos realizados por un estudiante y su ajuste a la trayectoria ideal, si ésta existe. En función de ello, lo evalúa y le envía el resultado a los agentes encargados del tratamiento de esta información. Para realizar este envío es necesario que los agentes interesados se suscriban a la recepción de esta información.
Tratar Pregunta	Tratar pregunta	Este servicio recibe una pregunta del estudiante y se encarga de elaborar su respuesta y enviarla al estudiante. Además, realiza una evaluación de la pregunta y le envía la información correspondiente a los agentes que se hayan suscrito a la recepción de esta información.

En la siguiente tabla se puede ver, para cada agente, qué otro agente es el encargado de prestar los servicios que demanda. Los agentes que no se muestran no utilizan servicios de ningún otro agente.

Elemento	**Servicio**	**Propósito**
Agente de Simulación		
Agente de Comunicación Global	Enviar información	Cuando se ejecuta una acción en el simulador, puede ser necesario actualizar la información que ven los estudiantes en el EV. Para ello, el agente de Simulación utiliza este servicio, que se encarga de enviar la actualización a todos ellos.

Agente Experto		
Agente de Simulación	Ejecutar acción	Cuando el Agente Experto ejecuta una acción o una postcondición, puede encontrar que alguna de las acciones a realizar no la puede llevar a cabo directamente, sino que implica la actualización de algún aspecto de la simulación que debe llevarse a cabo a través de este servicio del Agente de Simulación.
Agente Mundo	Validar precondición	Cuando se le solicita al Agente Experto que valide las precondiciones de una acción, alguna de ellas puede estar relacionada con el estado o la posición de algún elemento dentro del mundo. Si ese ese el caso, le solicita al Agente Mundo que realice la validación a través de este servicio.
Agente de Comunicación Global	Enviar información	Cuando se ejecuta una acción como consecuencia de una acción del estudiante, puede suceder que tenga efectos visibles en uno o varios EVs de los estudiantes, por lo que se utiliza este servicio para notificar los efectos de la acción a todos los estudiantes.
Agente Mundo		
Agente de Comunicación	Solicitar información	La principal labor del Agente Mundo es mantener actualizada la información que maneja el SIT respecto al estado del mundo. Para ello, solicita a todos los agentes de comunicación que le envíen la información que reciban de sus respectivos EVs, realizando las suscripciones necesarias para recibir toda la información que sea de interés, como posición de estudiantes y objetos.

Agente de Simulación	Solicitar información	Cuando se realiza una acción dentro del simulador, puede ocurrir que se cambie el estado o la ubicación de un objeto del mundo. Para mantener información actualizada del mundo, el Agente Mundo utiliza este servicio para suscribirse a todos los cambios que tengan lugar en la simulación y que puedan afectar a los objetos de los que mantiene información.
Agente de Trayectoria		
Agente de Comunicación	Solicitar información	Para poder realizar el seguimiento de un alumno, el Agente de Trayectoria necesita que se le envíen las coordenadas de sus movimientos por el EV, lo cual hace solicitando dicha información al Agente de Comunicación correspondiente a través de este servicio.
Agente de Estudiante		
Agente de Tutoría	Tratar acción	El Agente de Estudiante se suscribe a este servicio para recibir información acerca de las acciones realizadas por un estudiante y así poder incorporarlas a su modelo sobre dicho estudiante.
	Tratar actividad	El Agente de Estudiante se suscribe a este servicio para recibir información acerca de las actividades en la que se entrena un estudiante y poder incorporarlas a su modelo sobre dicho estudiante.
	Tratar movimiento	El Agente de Estudiante se suscribe a este servicio para recibir información acerca de las trayectorias seguidas por un estudiante y poder incorporarlas a su modelo sobre dicho estudiante.

	Tratar pregunta	El Agente de Estudiante se suscribe a este servicio para recibir información acerca de las preguntas realizadas por un estudiante y poder incorporarlas a su modelo sobre dicho estudiante.
Agente de Tutoría		
Agente de Comunicación Global	Notificar conexión	El Agente de Tutoría utiliza este servicio para controlar los estudiantes conectados a una sesión de entrenamiento, de manera que pueda saber, por ejemplo, si hay suficientes estudiantes para una actividad. También necesita saber si se ha conectado un nuevo estudiante para poder enviarle información acerca de las actividades disponibles, roles dentro de esa actividad y cualquier otra información que considere necesaria.
	Notificar desconexión	Al igual que en el caso anterior, el Agente de Tutoría necesita conocer si un estudiante se ha desconectado para poder gestionar en consecuencia la actividad en curso o en preparación.
Agente de Comunicación	Solicitar información	El Agente de Tutoría utiliza este servicio para recibir información relativa a las acciones que realiza un estudiante en el EV, sus movimientos, preguntas, etc.
	Enviar información	El Agente de Tutoría utiliza este servicio para dar información al estudiante, como respuestas a sus preguntas, pistas u otro tipo de instrucciones.
Agente Experto	Validar acción	Cuando un alumno realiza una acción, el Agente de Tutoría debe comprobar que esa acción puede llevarse a cabo. Para ello, una de las acciones que debe realizar es comprobar que se cumplen las precondiciones de la mencionada acción, lo que hace a través de este servicio.

	Validar precondición	Cuando un alumno realiza una acción, el Agente de Tutoría debe comprobar que esa acción puede llevarse a cabo. Para ello, una de las acciones que debe realizar es comprobar que se cumplen las precondiciones de la mencionada acción, lo que puede hacer a través de este servicio. La validación precondición a precondición puede hacerse, por ejemplo, para dar información más detallada al alumno acerca de por qué no puede realizar una acción.
	Ejecutar acción	La ejecución de una acción por parte de un alumno puede traer como consecuencia la necesidad de actualizar el estado de la actividad, lo cual se solicita a través de este servicio.
	Ejecutar postcondición	La ejecución de una acción por parte de un alumno puede traer como consecuencia la ejecución de otras acciones que actualicen el estado de la actividad, lo cual se solicita a través de este servicio.
Agente de Planificación	Dar acción siguiente	Se usa este servicio para obtener la siguiente acción del plan y poder compararla con la que ha realizado el estudiante y poder así evaluar su corrección.
	Aplicar acción	Se utiliza este servicio para actualizar el plan con la acción realizada por el estudiante.
	Planificar	Se utiliza este servicio para solicitar que se realice un plan para la actividad que se desea entrenar.
	Replanificar	Se usa este servicio para pedir la elaboración de un plan alternativo cuando, debido a las acciones realizadas por el alumno, no es posible continuar llevando a cabo el plan actual.

Agente de Trayectoria	Iniciar seguimiento	Se usa este servicio para indicar cuándo se desea comenzar a realizar un seguimiento de los movimientos del estudiante por el EV.
	Finalizar seguimiento	Se utiliza este servicio para indicar cuándo se desea terminar de hacer el seguimiento de los movimientos del alumno por el EV.
Agente de Estudiante	Dar información de estudiante	Dado que uno de los propósitos de un SIT es personalizar la tutoría utilizando la información que se posee acerca del estudiante, el Agente de Tutoría utiliza este servicio para consultar información sobre el estudiante y tomar decisiones de tutoría en función de esa información.

B.2. Relaciones y sus propiedades

La relación existente entre los elementos de esta vista es de *uso* de las páginas amarillas como medio para anunciar los propios servicios y para buscar los servicios proporcionados por los demás agentes.

B.3. Interfaces

Existen varias interfaces a tener en cuenta en esta vista. En primer lugar se encuentran las páginas amarillas, que utilizan todos los agentes para anunciar sus servicios y buscar otros. La manera de utilizar las páginas amarillas vendrá dada por la plataforma utilizada para implementar el sistema multiagente, pero se trata de un mecanismo estándar para todos ellos.

Además, hay que tener en cuenta que el nombre de los servicios debe ser común para el agente que los publica y el que los busca. Como muchos de estos servicios dependen de implementaciones concretas del SIT y no son arquitectónicamente relevantes, se ha decidido que los nombres de estos servicios se definan en una etapa más avanzada del diseño. También es recomendable que estos nombres no hagan referencia al agente que los proporciona ni al agente que los utiliza, ya que si éstos variasen también habría que modificar el nombre de los servicios, con el coste asociado que

ello pudiera implicar. En cualquier caso, el nombre del agente que facilita los servicios lo proporcionan las páginas amarillas, y el nombre del agente que los utiliza es lo que se quiere evitar conocer.

Por otro lado, hay que considerar la comunicación entre los agentes, que se va a realizar utilizando paso de mensajes que siguen el estándar FIPA-ACL. El contenido de los mensajes dependerá de cada tipo de mensaje concreto, por lo que se pospone su elaboración a una fase más avanzada del diseño, en la que se deberá especificar tanto los distintos mensajes que se pueden enviar como su contenido.

Finalmente, también debe tenerse en cuenta que tanto el Agente de Comunicación Global como los Agentes de Comunicación, encargados de comunicarse con el Centro de Mensajes, deben hacerlo manejando los mensajes definidos en el paquete anterior para comunicar los distintos sistemas entre sí.

B.4. Comportamientos

Cada agente va a buscar en las páginas amarillas los servicios que necesite de otros agentes, y esta búsqueda va a realizarse cuando el agente se inicie.

Algunos agentes, como el de Tutoría y el de Comunicación Global, no dependen ni de la existencia de estudiantes ni del problema concreto a resolver, por lo que deberían iniciarse al arrancar el SIT. Otros agentes, como los de Estudiante o los de Comunicación, dependen de la existencia de estudiantes, por lo que deberían iniciarse cuando se conecte un nuevo estudiante. Finalmente, otros agentes, como el de Planificación o el de Trayectoria, dependen de la actividad concreta que se vaya a resolver, por lo que deberían iniciarse una vez la actividad haya sido elegida y se vaya a comenzar su entrenamiento. Todo esto debe ser tenido en cuenta a la hora de buscar servicios en las páginas amarillas.

De esta manera, al iniciarse un agente lo primero que debe hacer es anunciar sus servicios, para que estén disponibles para todos los demás, y acto seguido comenzar a buscar qué agentes pueden prestarle los servicios que necesita. Según vaya encontrando a estos agentes, deberá ir suscribiéndose a los servicios que necesiten suscripción, y anotando a qué agentes solicitar el resto de servicios. Aunque esta anotación no es estrictamente necesaria, evita tener que buscar a los agentes en las páginas amarillas cada vez que se necesiten sus servicios.

C. Diagrama de contexto

Ver la presentación de la vista de descomposición del SIT (sección 5.5.1.5).

D. Guía de variabilidad

En esta parte del sistema es donde se hace necesaria una mayor flexibilidad, ya que parte de los escenarios de modificabilidad hacen referencia a elementos presentes en esta vista. En concreto, se menciona la posibilidad de deshabilitar funcionalidades tales como el seguimiento de las trayectorias o la propia tutoría.

Esto se va a poder realizar para una actividad completa, pero no durante la ejecución de una actividad, para evitar deshabilitar comportamientos, y por tanto servicios, que sean necesarios para el funcionamiento de otros agentes. Por tanto, una vez se haya anunciado un servicio en las páginas amarillas no se permitirá deshabilitarlo en tanto los estudiantes continúen el entrenamiento de esa actividad.

Puesto que estamos trabajando en un sistema cerrado, no debería darse la situación en que un agente no encuentre un servicio que necesita para funcionar. Aún así, debe considerarse la posibilidad de que esto suceda para algunos servicios. Para ello, habría que dotar a los agentes de comportamientos por defecto en caso de que no se encuentren determinados servicios (por ejemplo, el seguimiento de trayectorias, asumiendo que siempre se sigue la mejor trayectoria), aunque si fallan otros, como puede ser la planificación, no debería poder continuar la ejecución.

De esta manera, además de proteger el sistema frente a errores, se consigue que pueda simplificarse la inhabilitación de algunas funcionalidades. Por ejemplo, eliminar el seguimiento de trayectorias consistiría en no iniciar al Agente de Trayectoria. No es necesario que el Agente de Tutoría decida si necesita sus servicios o no, sino que lo buscará siempre en las páginas amarillas. Cuando no lo encuentre, supondrá que el estudiante realiza bien los desplazamientos por el EV.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

En el apartado B1 de esta vista se han mostrado dos tablas: la primera de ellas contiene los servicios que proporciona cada uno de los agentes del SIT, mientras que la segunda contiene los servicios que cada agente usa de entre los que ofrecen los demás.

Aunque el número de servicios que existen no es muy elevado, se ha decidido que, de todos los descritos, sólo se van a anunciar en las páginas amarillas los servicios que son utilizados directamente por otros agentes, que son los que aparecen en la segunda de las tablas mencionadas.

De todos los servicios enumerados, puede considerarse que pertenecen a tres tipos:

- El primer tipo son los servicios que proporcionan información de manera periódica y sin control por parte de los agentes, como pueden ser los servicios que proporcionan información sobre las acciones que realizan los estudiantes. Estos servicios requieren suscripción por parte de los otros agentes, y deben ser anunciados en las páginas amarillas.
- El segundo tipo son los servicios que se ejecutan bajo demanda y con control directo por parte del agente que los solicita, como la petición de validación de una precondición. Estos también deben ser anunciados en las páginas amarillas.
- El tercer tipo son los servicios que se ejecutan como resultado de recibir un mensaje producto de una suscripción. Estos servicios, que se invocan de forma indirecta por parte del agente que envía la información, no necesitan ser anunciados en las páginas amarillas, ya que nunca se utilizan de manera directa. Son servicios como el tratamiento de las coordenadas de los desplazamientos del estudiante por el EV.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

En todo momento se ha supuesto que se va a trabajar con una plataforma de desarrollo de sistemas multiagente que proporciona un servicio de páginas amarillas en el que cada agente puede anunciar sus servicios y buscar los servicios publicados por otros agentes.

También se ha supuesto que la plataforma permitirá que los agentes se comuniquen mediante paso de mensajes y que estos mensajes seguirán el estándar FIPA-ACL, según se ha mencionado en el apartado de interfaces.

Se ha tomado como prototipo la plataforma JADE (*Java Agent DEvelopment Framework*).

F. Otra información

No aplicable.

G. Vistas relacionadas

Ver vistas de descomposición 5 a 14 (secciones 5.5.1.5 a 5.5.1.14).

5.5.4. Vista *Peer-to-Peer*

La vista *peer-to-peer* resulta de utilidad para mostrar el comportamiento del sistema en ejecución. Aunque existen otro tipo de vistas que también sirven a este fin, los sistemas basados en agentes son esencial y originalmente de tipo *peer-to-peer*, por lo que se ha considerado más adecuado utilizar esta vista para documentar el comportamiento del sistema. Por otra parte, si bien es cierto que existen otro tipo de organizaciones para sistemas basados en agentes, éstas se suelen utilizar cuando el número de agentes es elevado, cuando existen protocolos de interacción complejos o cuando existe una clara diferenciación en las funciones de determinados grupos de agentes. Puesto que el sistema actual no cumple ninguna de estas características, se ha optado por mantener la decisión de utilizar una vista *peer-to-peer* para documentar los principales flujos de actividad del sistema.

Debido a que una de las características deseadas del sistema es la posibilidad de modificar con facilidad determinados elementos del mismo, como las estrategias de tutoría o la manera de realizar el modelado del estudiante, en los paquetes que se describen a continuación se ha introducido cierta dosis de ambigüedad que permita adaptar el flujo de acciones a las características propias de los elementos para los cuales se desea obtener dicha modificabilidad. Por lo tanto, los aspectos que puedan resultar más ambiguos deberán concretarse en una posterior fase de diseño donde se tomen ya en consideración los detalles concretos de, por ejemplo, una estrategia de tutoría específica.

Para poder reflejar el funcionamiento deseado en cada uno de los paquetes, se ha decidido simplificar los diagramas para mantener su complejidad en un nivel aceptable. De esta manera, sólo se muestra el intercambio de mensajes entre agentes, aunque cada una de las acciones las debe llevar a cabo el correspondiente comportamiento descrito en la sección 5.5.1, a través de los servicios descritos en la sección 5.5.3.2. Estos mensajes intercambiados entre los agentes siguen el estándar FIPA-ACL, aunque su formato concreto deberá ser determinado en una fase más avanzada del diseño.

5.5.4.1. Vista *Peer-to-Peer* Paquete 1: Ejecución de una Acción del Estudiante

A. Presentación de la vista

En este paquete se muestra el intercambio de mensajes que tiene lugar entre los agentes del SIT cuando un estudiante realiza una acción en su EV y ésta se envía al SIT a través del Centro de Mensajes para que sea tratada.

Como se ha mencionado anteriormente, la secuencia concreta de acciones depende de la estrategia de tutoría que se esté utilizando, y puede no estar definida hasta que el sistema se encuentre en ejecución, como ocurre si esta estrategia de tutoría es leída desde un fichero de configuración. Puesto que éste es uno de los aspectos en los que se desea que el sistema sea modificable, las correspondientes acciones se muestran para reflejar cómo tendrían lugar en caso de ser necesarias. En los siguientes apartados de este paquete se pueden encontrar más detalles relativos a esta característica.

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Los elementos pertenecientes a esta vista son tanto los agentes y comportamientos que se pueden ver en la vista de descomposición (ver secciones 5.5.1.5 a 5.5.1.14) como los servicios descritos en la vista editor-suscriptor (ver sección 5.5.3.2).

B.2. Relaciones y sus propiedades

La relación existente entre los elementos de esta vista es *invoca*, por la cual un elemento invoca la ejecución de un servicio de otro elemento a través del envío de un mensaje.

B.3. Interfaces

Los servicios requeridos y proporcionados por cada elemento pueden verse en el paquete 2 de la vista editor-suscriptor (ver sección 5.5.3.2).

B.4. Comportamientos

La acción comienza cuando un Agente de Comunicación, que es el encargado de comunicarse con un estudiante concreto, recibe a través del Centro de Mensajes una acción realizada por el citado estudiante (mensaje 1). En ese momento envía la acción a los agentes que estén interesados en las acciones del estudiante, que se habrán suscrito previamente a la recepción de las mismas. En este caso concreto son el Agente de Estudiante (mensaje 2), quien mantiene la traza de acciones realizadas por el alumno, y el Agente de Tutoría (mensaje 3), que es el encargado de comprobar su adecuación a la actividad en curso.

Una vez el Agente de Tutoría ha recibido la acción del estudiante, debe comprobar que se dan las condiciones para que el alumno pueda realizar la acción (por ejemplo, que posea la llave que abre una determinada puerta).

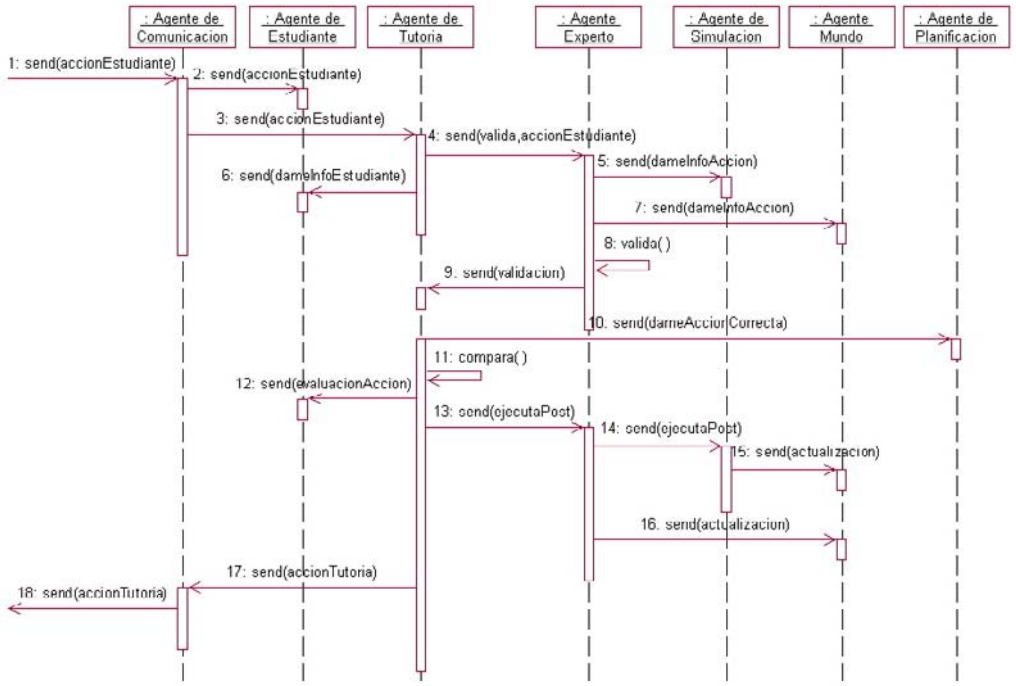

Figura 5.65: Tratamiento de una acción del estudiante

Para ello, le solicita al Agente Experto (mensaje 4) que compruebe si se cumplen todas las precondiciones asociadas a la acción.

Para realizar esta comprobación, el Agente Experto consulta los datos pertinentes a todas las fuentes que pueden tener información relevante para validar las precondiciones. De esta forma, se le puede solicitar información al Agente de Simulación (mensaje 5) acerca del estado de la misma (por ejemplo, si una determinada máquina está en marcha) y al Agente Mundo (mensaje 7) relativa a la situación de algún elemento dentro del EV (por ejemplo, si un objeto está cerca). Con esta información, el Agente Experto valida las precondiciones de la acción (mensaje 8) y le envía el resultado al Agente de Tutoría (mensaje 9).

Mientras el Agente Experto realiza la validación de la acción, el Agente de Tutoría puede ir solicitándole al Agente de Estudiante (mensaje 6) la información relativa a éste que puede hacerle falta para evaluar la acción. Esta información dependerá tanto de las necesidades de la estrategia de tutoría como de las capacidades que posea el Agente de Estudiante de modelar las características del estudiante, sus acciones, etc.

Una vez se ha comprobado que la acción es válida, es decir, que se puede llevar a cabo, también se debe comprobar que es correcta, para lo cual hay que

compararla con la que, según el plan realizado por el Agente de Planificación, sería la siguiente acción a realizar (mensajes 10 y 11). Una vez se conoce la corrección de la acción, y utilizando la información del estudiante de la que se dispone, se puede evaluar la acción y enviarle esta evaluación al Agente de Estudiante (mensaje 12) para que la almacene y la utilice para completar el modelo del estudiante.

Tras comprobar que la ejecución de la acción es posible, es necesario actualizar la actividad con los cambios que produce la ejecución de la acción. Para ello, el Agente de Tutoría le solicita al Agente Experto (mensaje 13) que se encargue de ejecutar las postcondiciones de la acción, lo que implica que se actualice el estado de la simulación (mensaje 14) y el del mundo en función de ésta (mensaje 15) y que además se actualice la información del mundo con datos que no dependen de la simulación (mensaje 16). Aunque no aparece en el diagrama, también podría ser necesario ejecutar alguna acción en el EV de alguno de los estudiantes, lo que se haría a través de un mensaje del Agente Experto al Agente de Comunicación Global.

Finalmente, también puede ser necesario proporcionarle al estudiante algún tipo de realimentación acerca de su acción, lo cual se realiza a través de un mensaje del Agente de Tutoría al Agente de Comunicación del estudiante (mensaje 17) que se le envía a través del Centro de Mensajes (mensaje 18).

C. Diagrama de contexto

Ver la presentación de la vista de descomposición del SIT (sección 5.5.1.5).

D. Guía de variabilidad

Las variaciones sobre lo expuesto pueden llegar a ser bastante notables. Para que la estrategia de tutoría sea lo más modificable posible, una de las opciones más atractivas, aunque también más complejas de implementar, consiste en que se lea de un fichero de configuración, donde se ajustarían distintos parámetros relativos a la misma, como si se permite realizar acciones erróneas, si se le dan pistas al alumno y cuándo, si se le responden preguntas, si se dan distintas respuestas en función de sus conocimientos y aptitudes, y tantos otros aspectos como se quieran considerar. De esta manera, se puede variar la estrategia de tutoría sin necesidad de modificar el diseño o la implementación de la aplicación.

Por otra parte, también puede hacerse que la estrategia de tutoría se apoye en mayor o menor grado en el conocimiento que se tiene sobre el alumno, y a su vez, este conocimiento puede variar enormemente su contenido y su grado de detalle. Por tanto, lo que se ha modelado como un único mensaje para solicitar información sobre el estudiante puede transformarse en un

intercambio de mensajes más numeroso. Lo mismo sucede con el intercambio de mensajes para validar las precondiciones de las acciones realizadas por el estudiante.

Lo que sí queda patente es que la sofisticación de la tutoría - y por tanto las variaciones sobre el modelo presentado - dependerá en gran medida de las capacidades de los agentes que dan apoyo al Agente de Tutoría.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

Siguiendo el curso de acciones de tratamiento de una actividad, a lo largo del diseño de la arquitectura se han tenido en cuenta otras posibilidades, que se describen a continuación.

En primer lugar, hay que tener en cuenta que la estrategia de tutoría puede no permitir la realización de acciones que no sean válidas. En ese caso, todas las acciones que realice un alumno pueden ser correctas o incorrectas, pero siempre serán válidas. En este caso, no sería necesario comprobar la validez de la acción, pues ya se sabría de antemano que la acción sí es válida.

Para realizar la validación de las acciones, inicialmente se pensó que podría ser suficiente con preguntar al planificador, ya que es labor del planificador validar precondiciones para elaborar el plan. Sin embargo, pronto se hizo patente la necesidad de realizar la validación por otros medios, pues el planificador muchas veces carece de la información necesaria para efectuar la validación, y en otras ocasiones no es capaz de operar con la información que se necesita en la validación.

Por poner un ejemplo de cada uno de estos dos factores, se puede considerar un simulador de una máquina de una central nuclear. La realización de una determinada acción puede estar condicionada al estado en que se encuentre la mencionada máquina, y ese estado puede estar influenciado por multitud de factores internos al simulador. En este caso, el estado de la máquina sólo podremos consultárselo al Agente de Simulación, ya que el Agente de Planificación no tendrá conocimiento de lo que sucede en la simulación. Por otra parte, para realizar una acción puede ser necesario que un alumno esté cerca de un objeto. El concepto de *cerca*, así como el cálculo de una distancia a partir de unas coordenadas, es una labor que no está al alcance de un planificador, que en muchos casos no están dotados con la capacidad de realizar operaciones aritméticas. Así, en este caso, se consideró que era más adecuado que la validación de estas acciones la realizase el Agente Mundo, quien dispone ya de esta información y al que se le puede dotar de la capacidad para trabajar con ella.

Una vez validada la acción, se le solicita al Agente de Planificación que nos diga si la acción es correcta o no. Si la acción no era válida tampoco será correcta, pero si era válida puede darse el caso de que no sea correcta o que no sea relevante. El hecho de que se permita la ejecución de acciones incorrectas o irrelevantes depende de que la estrategia de tutoría sea más o menos restrictiva, pero lo que no debería permitir en ningún caso es la realización de acciones no válidas, pues el estado del mundo podría quedar inconsistente. En caso de que se permita la realización de acciones incorrectas, será de utilidad que el planificador sea capaz de devolvernos a la situación anterior por si se desea reconducir las acciones del alumno. También resulta de utilidad que sea capaz de realizar replanificaciones, para tratar de resolver la actividad en curso por un camino que, sin ser el mejor desde el punto de partida, es el que ha decidido seguir el estudiante, ya sea de forma consciente o no. Todas estas características permitirán que la tutoría se adapte en mayor medida a las acciones del alumno.

Aunque en el diagrama anterior se han mostrado todas las acciones de manera secuencial, algunas de ellas pueden realizarse en paralelo, ya que no interfieren unas con otras y, si la plataforma utilizada lo permite, la ejecución de los agentes se realiza en paralelo. Así, la información que se solicita en los mensajes 5, 6 y 7 se puede pedir a la vez, sin necesidad de esperar a recibir una para solicitar la siguiente. Sin embargo, si habrá que disponer de la información de los mensajes 5 y 7 para poder realizar la acción 8. De igual manera, la acción que desencadena el mensaje 6 puede realizarse en paralelo con las acciones 5 a 11, pero será necesario disponer de la información para poder realizar la evaluación de la acción y enviar el mensaje 12. Se sugiere que la acción 6 se realice en ese momento porque la obtención de la información del estudiante puede requerir un procesamiento que tome algo de tiempo, lo que, si se realiza en paralelo con las otras acciones, repercutirá en una mayor velocidad de ejecución.

Finalmente, debe tenerse en cuenta el hecho de que, dado que la validación de precondiciones y la ejecución de postcondiciones se hacen de manera separada a la planificación, será necesario definir, por un lado, los dominios y operadores que maneje el planificador, y por otro, los operadores que vaya a manejar el Agente Experto, junto con las precondiciones y postcondiciones correctamente especificadas para que este agente sepa a quién debe pedirle la información para hacer la validación.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

En todo momento se ha supuesto que los agentes tienen a su disposición los servicios que necesitan para llevar a cabo sus responsabilidades. Se ha hecho esta suposición porque, como ya se ha mencionado anteriormente, se trabaja en un entorno cerrado en el que todos los agentes se encuentran bajo nuestro control. De esta forma, siempre podemos asegurarnos, antes de poner en marcha el sistema, de que contamos con todos los agentes necesarios y que éstos prestan todos los servicios que se requieren de ellos.

Aún así, es necesario tener en cuenta que, para que se puedan añadir y quitar agentes en función de las necesidades de la actividad que se entrena, se debe diseñar un mecanismo para que, o bien no se busquen los servicios que no se van a necesitar, o bien se busquen todos los servicios conocidos pero se pueda iniciar la actividad si no se encuentran los servicios que luego no se van a utilizar.

F. Otra información

El hecho de que se necesite que el Agente Mundo realice ciertas operaciones ha influido notablemente en la decisión de que la información con la que trabaja se almacene en una ontología en lugar de en una base de datos. La ontología permite la definición de relaciones entre elementos y la utilización de razonadores que trabajen sobre ella, lo que supone una herramienta muy potente y elimina la necesidad de desarrollar esas funcionalidades.

Lo mismo sucede en el caso del Agente de Estudiante, que en esta propuesta almacena la información del modelo del estudiante en una ontología sobre la que poder realizar razonamientos y deducciones.

Por otra parte, dado que varios de los agentes manejan información relativa a las acciones de los estudiantes, se puede inferir que el sistema es sensible al formato que adopte esta representación. Por este motivo, de cara a poder sustituir el planificador utilizado, se debe intentar que la representación de las acciones sea independiente de la representación de los operadores que maneja el planificador, y deberá realizarse la traducción pertinente con algún elemento situado entre el Agente de Planificación y el planificador. Resulta probable que el Agente de Planificación se vea afectado por un cambio de planificador, pero el hecho de utilizar un envoltorio (lo que en la vista de descomposición se ha denominado *Wrapper*) posibilita que el impacto de este cambio pueda verse atenuado incluso en el propio agente. De esta manera, con cambios menores podría sería posible sustituir el planificador solamente diseñando un nuevo envoltorio que realice la comunicación con el planificador nuevo.

G. Vistas relacionadas

Descripción de agentes y comportamientos en la vista de descomposición (secciones 5.5.1.5 a 5.5.1.14) y descripción de los servicios en la vista editor-suscriptor (sección 5.5.3.2).

5.5.4.2. Vista *Peer-to-Peer* Paquete 2: Seguimiento de Movimientos del Estudiante

A. Presentación de la vista

En este paquete se muestra el intercambio de mensajes entre los agentes cuando tiene lugar un desplazamiento del estudiante por el EV. De manera similar a lo que sucedía en el paquete anterior, el intercambio de mensajes está sujeto a lo que estipule la estrategia de tutoría que se esté utilizando, por lo que, igual que en el caso anterior, lo aquí expuesto está sujeto a variaciones en función de la estrategia de tutoría.

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Los elementos pertenecientes a esta vista son tanto los agentes y comportamientos que se pueden ver en la vista de descomposición (ver secciones 5.5.1.5 a 5.5.1.14) como los servicios descritos en la vista editor-suscriptor (ver sección 5.5.3.2).

B.2. Relaciones y sus propiedades

La relación existente entre los elementos de esta vista es *invoca*, por la cual un elemento invoca la ejecución de un servicio de otro elemento a través del envío de un mensaje.

B.3. Interfaces

Los servicios requeridos y proporcionados por cada elemento pueden verse en el paquete 2 de la vista editor-suscriptor (ver sección 5.5.3.2).

B.4. Comportamientos

La ejecución comienza cuando el Agente de Tutoría indica al de Trayectoria que comience la realización del seguimiento de los movimientos del estudiante (mensaje 1) y éste calcula las trayectorias más adecuadas.

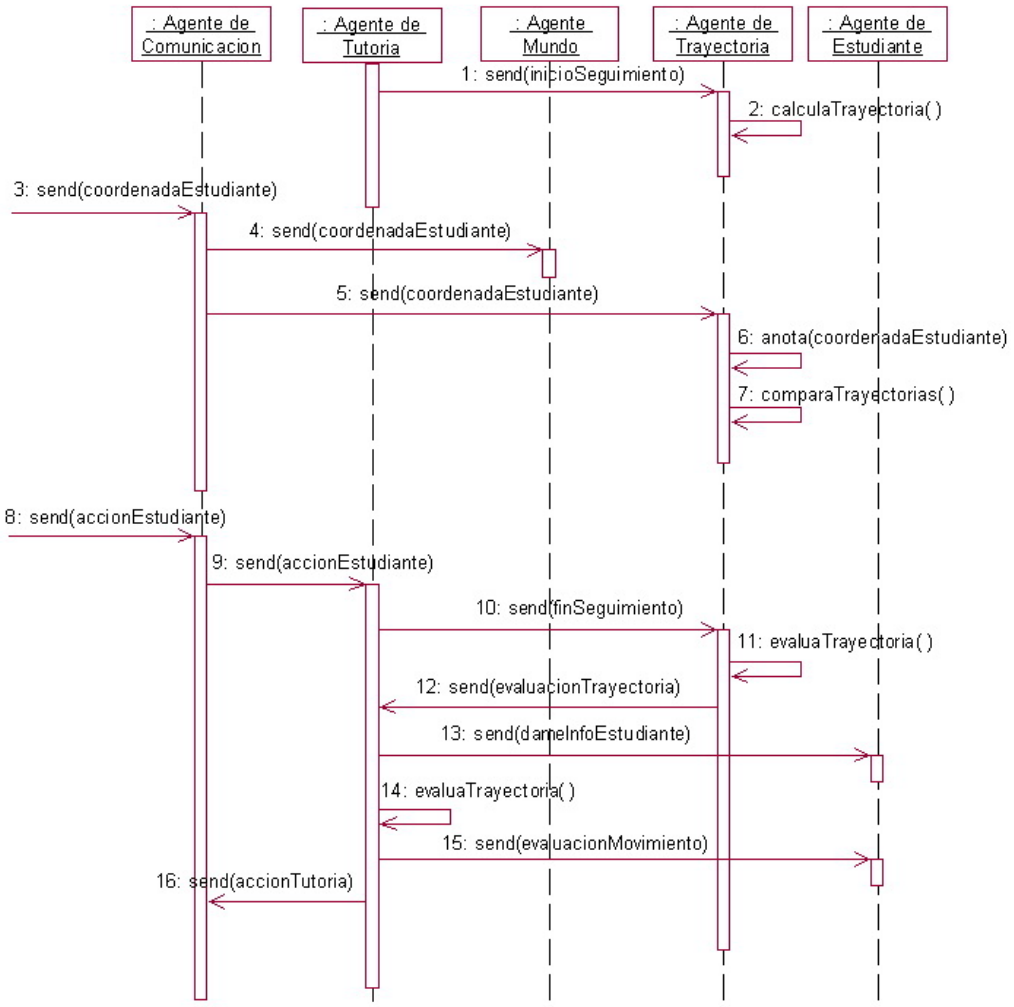

Figura 5.66: Tratamiento de un movimiento del estudiante

Posteriormente, el Agente de Comunicación, que es el encargado de comunicarse con un estudiante concreto, comienza a recibir mensajes desde el Centro de Mensajes (mensaje 3) con los que se le indican las nuevas coordenadas de situación del estudiante dentro de su EV. Este agente reenvía las coordenadas a los agentes que se hayan suscrito a su recepción, en este caso al Agente Mundo (mensaje 4), que mantiene actualizada la situación del estudiante en su representación del EV, y el Agente de Trayectoria (mensaje 5), que realiza el seguimiento de los movimientos. Para ello, registra la coordenada (mensaje 6) y va construyendo una trayectoria que compara con la trayectoria calculada inicialmente (mensaje 7).

Este proceso se repite hasta que el estudiante realiza una acción (mensaje 8). Cuando el Agente de Tutoría recibe notificación de este suceso (mensaje 9), le indica al de Trayectoria que finalice el seguimiento del tramo de trayectoria actual (mensaje 10). Es entonces cuando el Agente de Trayectoria calcula cuánto se ajusta la trayectoria del estudiante a la ideal calculada por él (mensaje 11) y le envía el resultado al Agente de Tutoría (mensaje 12).

Al recibir esta evaluación es cuando, junto con la información que recibe sobre el estudiante (mensaje 13), realiza su propia evaluación de la trayectoria seguida por el estudiante (mensaje 14) y le envía dicha evaluación al Agente de Estudiante (mensaje 15) para que la utilice para completar el modelo que realiza sobre el estudiante.

En caso necesario, se envía un mensaje al EV del estudiante (mensaje 16) con las acciones de tutoría necesarias para indicarle al estudiante la idoneidad de sus movimientos.

C. Diagrama de contexto

Ver la presentación de la vista de descomposición del SIT (sección 5.5.1.5).

D. Guía de variabilidad

Una de las variaciones que se contempla sobre el esquema propuesto es que no se realice ningún seguimiento de los movimientos del estudiante, bien porque la estrategia de tutoría no lo considere como un factor a tener en cuenta, bien porque la actividad objeto de entrenamiento no implique desplazamientos por el EV.

Otra de las posibles variaciones consiste en que el Agente de Trayectoria, a medida que va comparando la trayectoria del estudiante con la calculada, le envíe pistas si detecta que se desvía en exceso del camino que debería seguir.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

Aunque se ha considerado la posibilidad de enviar desde el EV mensajes que indiquen el comienzo y la finalización de un desplazamiento, lo que facilitaría la labor del Agente de Tutoría y del de Trayectoria, se ha descartado principalmente por dos motivos.

El primero es que hay dispositivos de interacción que funcionan directamente transmitiendo coordenadas, como los sensores de posición, lo cual dificulta identificar cuándo se inicia y cuándo finaliza un movimiento.

El segundo motivo es que, aunque podría realizarse en el EV un análisis de las coordenadas que indique cuando comienza y finaliza el desplazamiento,

esto conlleva trasladar al EV una lógica que no le corresponde. Puesto que uno de los objetivos que se persiguen es poder intercambiar los EVs utilizados, hay que pensar que este tipo de acción no la realizan habitualmente los Entornos Virtuales, por lo que habría que realizar modificaciones en los mismos siempre que se desee sustituir uno por otro.

Dependiendo del número de estudiantes y de los algoritmos de cálculo y seguimiento de trayectorias que se utilicen, puede suceder que el Agente de Trayectoria sea incapaz de procesar los desplazamientos de todos los estudiantes con la rapidez necesaria. Se postpone, por tanto, la decisión de utilizar tantos Agentes de Trayectoria como estudiantes para poder realizar el procesamiento con mayor velocidad, en caso de que sea necesario.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

Se parte de la premisa de que una trayectoria ideal no viene dada únicamente por criterios geométricos, sino que puede haber otros factores que influyan en la adecuación de la misma. Así, por ejemplo, en una central nuclear la trayectoria más corta puede pasar junto a una fuente de radiación, mientras que otra más larga puede mantenernos alejados de dicha fuente, convirtiéndola así en una trayectoria más adecuada. Por tanto, se debe contar con algoritmos de cálculo de trayectorias que permitan incorporar criterios heurísticos adicionales a los meramente geométricos.

F. Otra información

No disponible.

G. Vistas relacionadas

Descripción de agentes y comportamientos en la vista de descomposición (secciones 5.5.1.5 a 5.5.1.14) y descripción de los servicios en la vista editor-suscriptor (sección 5.5.3.2).

5.5.4.3. Vista *Peer-to-Peer* Paquete 3: Respuesta a una Pregunta del Estudiante

A. Presentación de la vista

En este paquete se presenta el funcionamiento del SIT cuando un estudiante realiza una pregunta que debe ser respondida por éste. La variedad de preguntas que puede realizar un estudiante es bastante amplia, y las formas de realizar una misma pregunta son también numerosas, por lo que en la presente vista no se considera la problemática de analizar la pregunta para que la entienda el SIT, sino que se supone que ya está en un formato manejable por éste.

Por el mismo motivo, muchos de los agentes pueden verse involucrados en la elaboración de las respuestas, por lo que de momento se ha restringido el sistema a la elaboración de respuestas no excesivamente complejas a preguntas relativamente sencillas, que en general se pueden responder con la información de la que dispone uno sólo de los agentes.

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Los elementos pertenecientes a esta vista son tanto los agentes y comportamientos que se pueden ver en la vista de descomposición (ver secciones 5.5.1.5 a 5.5.1.14) como los servicios descritos en la vista editor-suscriptor (ver sección 5.5.3.2).

B.2. Relaciones y sus propiedades

La relación existente entre los elementos de esta vista es *invoca*, por la cual un elemento invoca la ejecución de un servicio de otro elemento a través del envío de un mensaje.

B.3. Interfaces

Los servicios requeridos y proporcionados por cada elemento pueden verse en el paquete 2 de la vista editor-suscriptor (ver sección 5.5.3.2).

B.4. Comportamientos

La acción comienza cuando el Agente de Comunicación, encargado de la comunicación con un estudiante determinado, recibe desde el Centro de Mensajes un mensaje con la pregunta realizada por el estudiante (mensaje 1). Este agente le envía la pregunta a los agentes que se hayan suscrito a

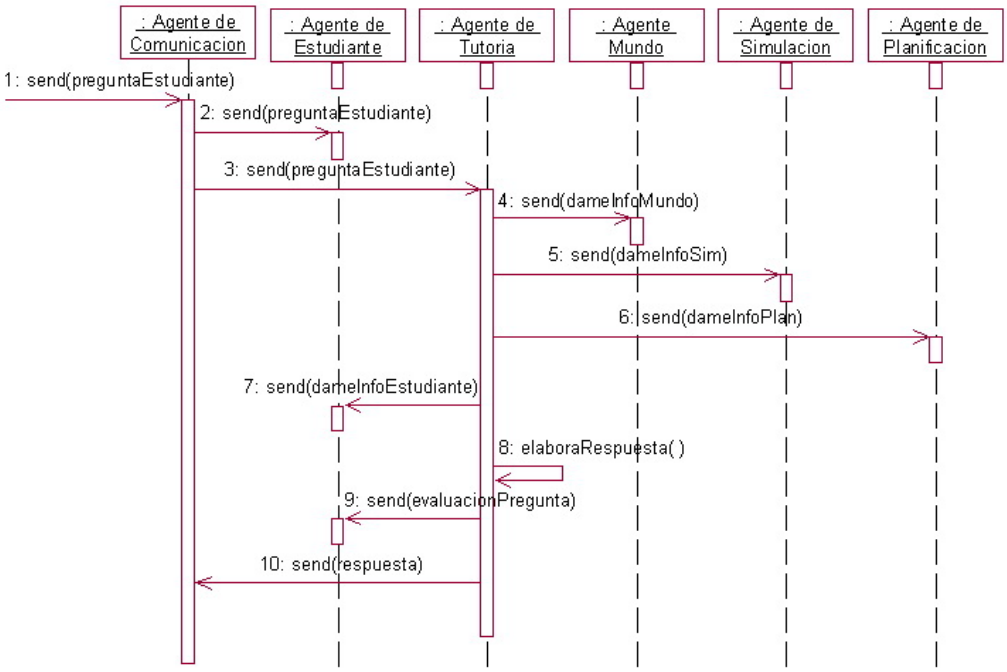

Figura 5.67: Tratamiento de una pregunta del estudiante

su recepción, que en este caso son el Agente de Estudiante (mensaje 2) para elaborar el modelo de estudiante, y el Agente de Tutoría (mensaje 3), que es el encargado de responder. Tras analizar el tipo de pregunta realizada, el Agente de Tutoría solicita la información necesaria para elaborar la respuesta al agente apropiado (alternativamente, mensajes 4 a 6). Tras recibir información sobre la respuesta, solicita también al Agente de Estudiante información relativa a éste (mensaje 7) para, por un lado, decidir el nivel de detalle de la respuesta (mensaje 8), y por otro, para evaluar la adecuación de la misma (mensaje 9). Finalmente, se le envía la respuesta al Agente de Comunicación (mensaje 10) para que éste se la envíe al estudiante que la realizó.

C. Diagrama de contexto

Ver la presentación de la vista de descomposición del SIT (sección 5.5.1.5).

D. Guía de variabilidad

Para diseñar el mecanismo de respuesta a una pregunta hay que tener en cuenta el grado de complejidad que se quiere incluir tanto en el análisis de las preguntas como en el tipo y la elaboración de las respuestas.

La variación más obvia respecto a lo planteado es la inclusión de nuevos tipos de preguntas simples para las que puede ser necesario solicitar información a otros agentes. Para ello, sería necesario incluir nuevos agentes en el diagrama mostrado, pero el funcionamiento básico sería el mismo. En la medida en que se quiera complicar el tipo de respuesta a la que se responde, es probable que sea necesario incorporar un elemento que trabaje con la manipulación de lenguaje natural y que sea capaz de elaborar respuestas complejas a partir de diversas fuentes.

Otra posible variación consiste en que la estrategia de tutoría no contemple la posibilidad de ofrecer respuestas con distinto nivel de detalle, o que decida que en ciertas situaciones o con ciertos estudiantes no se ofrezcan respuestas a las preguntas.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

La probabilidad de encontrar un EV que permita el envío de preguntas, salvo que se haya diseñado expresamente para ello, es bastante baja. Esto, que a priori puede parecer un inconveniente, puede acabar convirtiéndose en una ventaja, ya que posibilita que pueda disponerse de una aplicación en el cliente, construida más a medida del SIT, que se ejecute junto con el EV pero que funcione de manera separada y, sobre todo, que procese las preguntas y las envíe al SIT en un formato fácilmente manejable por éste.

La preguntas que se ha pensado resolver, en principio, son relativas a los objetos del entorno (qué son o para qué sirven), cuya respuesta se puede almacenar como una descripción relativa a los objetos en el mismo soporte, fichero o base de datos, en el que se guarde otra información relativa a ellos, y que puede ser manejada y proporcionada por el Agente Mundo.

También se pueden responder preguntas relativas a las acciones a realizar (por ejemplo, qué hay que hacer o qué se debería haber hecho), cuya respuesta la puede proporcionar el Agente de Planificación.

Otros posibles tipos de preguntas a considerar son por qué no se puede hacer algo, cuya respuesta la proporcionaría el Agente Experto a partir de las precondiciones de la acción que se quiere realizar, o por dónde llegar a algún punto del EV, que sería respondida a partir de información proporcionada por el Agente de Trayectoria, posiblemente completada con información proporcionada por el Agente Mundo, ya que el primero sólo maneja información relativa a coordenadas, y esto no es especialmente informativo a la hora de construir una respuesta para un estudiante.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

Se ha supuesto que el estudiante realiza sus preguntas utilizando un reconocedor de voz o una aplicación de texto similar a un chat. No se han contemplado otras posibilidades ni, por tanto, sus posibles implicaciones en la arquitectura del sistema.

F. Otra información

No disponible.

G. Vistas relacionadas

Descripción de agentes y comportamientos en la vista de descomposición (secciones 5.5.1.5 a 5.5.1.14) y descripción de los servicios en la vista editor-suscriptor (sección 5.5.3.2).

5.5.5. Vista de Despliegue

Dentro del conjunto de vistas de asignación, la vista de despliegue permite especificar en qué elementos hardware se van a ejecutar los elementos software que se han descrito en las vistas anteriores.

Como se verá a continuación, algunos de los elementos hardware pueden variar significativamente en función del dominio de entrenamiento, pero en líneas generales no se requiere un hardware con unas exigencias mayores que las prestaciones que puede ofrecer un PC de gama alta.

5.5.5.1. Vista de Despliegue Paquete 1: Despliegue de la Aplicación

A. Presentación de la vista

Como se puede ver en la imagen que acompaña este apartado, la idea general es que cada uno de los subsistemas que conforman la aplicación de entrenamiento, Sistema Inteligente de Tutoría, Centro de Mensajes, Simulador y Entorno Virtual, se ejecuten en máquinas distintas, y que cada estudiante que se conecte a la aplicación a través de un EV lo haga desde una máquina distinta a la de otros estudiantes.

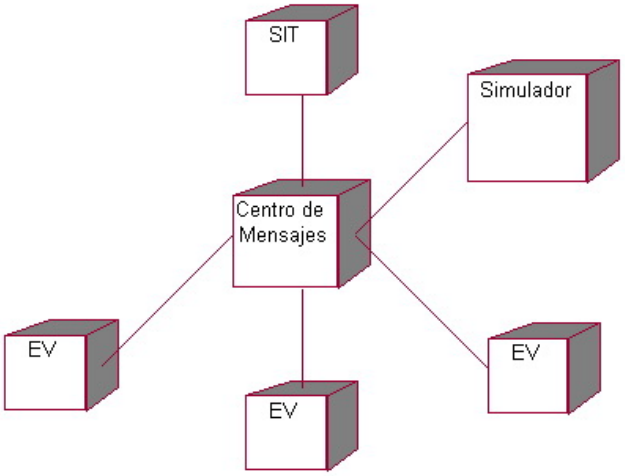

Figura 5.68: Diagrama de despliegue de la aplicación

B. Catálogo de elementos

B.1. Elementos y sus propiedades

Entorno Virtual	Las máquinas donde se ejecuten los EVs, al ser aplicaciones tridimensionales cuyos requisitos a nivel gráfico suelen ser relativamente altos, necesitan ser PCs de gama alta provistos de tarjetas gráficas con aceleración 3D, de manera que sean capaces de realizar una ejecución suave, de al menos 30 fps (*frames por segundo*), para dar una sensación de movimiento fluido durante la navegación a través del EV.
Sistema Inteligente de Tutoría	La máquina donde se ejecute el SIT debe ser un PC de gama alta con capacidad de procesamiento múltiple, que pueda dar cabida a la ejecución de los agentes en paralelo y que soporte en momentos determinados la ejecución de procesos con un consumo alto de procesador y memoria, como puede ser el cálculo de trayectorias o, en mayor medida, el proceso de planificación.

Centro de Mensajes	En principio, el Centro de Mensajes es una aplicación que consume pocos recursos y que no realiza un procesamiento exigente, por lo que será suficiente con que se ejecute en un PC de gama baja con una conexión a la red de velocidad media (100Mbps) que le permita redirigir todos los mensajes que reciba sin retardos apreciables.
Simulador	Las características de la máquina donde se ejecute el simulador deben ser definidas según los requisitos de cada simulador concreto.

B.2. Relaciones y sus propiedades

La relación que dirige esta vista es *asignado a*, que relaciona los elementos software con los elementos hardware en los que se ejecutan. No existe ninguna variación respecto a lo definido por esta relación.

B.3. Interfaces

Las interfaces entre los distintos nodos, al producirse la comunicación en una red a través del Centro de Mensajes, están constituidas por unas tarjetas de red de velocidad media o alta ($\geq$100Mbps) que realizan la comunicación a través del protocolo TCP/IP.

B.4. Comportamientos

No aplicable.

C. Diagrama de contexto

No aplicable.

D. Guía de variabilidad

Existen dos puntos importantes de variabilidad respecto a lo mostrado anteriormente. El primero de ellos lo constituye el simulador, que en vez de ejecutarse en un máquina separada, que puede ser el caso más habitual, puede presentar otras situaciones. La primera de ellas es que se ejecute en una misma máquina junto con el EV, ya que en muchos casos ambas aplicaciones se encuentran implementadas de manera conjunta. También puede darse el

caso de un simulador sencillo que se implemente como parte del agente de simulación, en cuyo caso se ejecutaría en la misma máquina que el SIT.

Si se ejecuta en una máquina separada, dependiendo de los requisitos de la simulación se pueden dar la situación de que se ejecute en un PC de gama media o en una máquina de altas prestaciones diseñada exclusivamente para ejecutar esa simulación, como podrían ser cabinas simuladoras de vuelo.

Por otra parte, en función de las características del dominio de entrenamiento, puede aparecer la necesidad de ejecutar el SIT en distintas máquinas para dar soporte a unas necesidades de procesamiento exigentes o para poder albergar un número alto de estudiantes que trabajan de manera conjunta. En este caso, plataformas de agentes como JADE proporcionan de serie mecanismos para distribuir la plataforma y sus agentes en varias máquinas sin modificar la forma en que los agentes están programados.

Lo que no se contempla en ningún caso, por el momento, es la migración dinámica de partes de la aplicación a otras máquinas.

E. Información sobre la arquitectura

E.1. Decisiones de diseño

Dentro del tratamiento de una acción del estudiante, las operaciones que pueden llevar más tiempo de ejecución y más carga de trabajo son las replanificaciones de la actividad, si el alumno realiza muchas acciones incorrectas, la validación de precondiciones en el mundo, si las precondiciones son grandes o complejas, y el seguimiento de las trayectorias, especialmente si el alumno se desvía mucho de las trayectorias ideales.

Puesto que la comunicación de los agentes encargados de estas labores con otros está bastante restringida, son candidatos bastante claros a ejecutarse en una máquina distinta a la del resto de los agentes. Lo mismo sucede con los Agentes de Estudiante, que además pueden proliferar mucho, dependiendo del número de estudiantes involucrados en una actividad. En este último caso la distribución en otras máquinas podría ser algo más compleja, ya que darían lugar a una mayor comunicación entre distintas máquinas.

La distribución de agentes en distintas máquinas también puede resultar problemática por imponer la restricción de que la plataforma utilizada para implementar los agentes debe permitir realizar esta distribución.

E.2. Análisis de resultados

Ver sección 5.6

E.3. Suposiciones

Para realizar el despliegue de la aplicación se ha supuesto la utilización de una plataforma con características similares a JADE, que permite la ejecución de una plataforma de agentes en varias máquinas de manera que el diseño de los agentes no se vea afectado, sino que es la propia plataforma la que se encarga de gestionar el envío y recepción de mensajes de manera idéntica, se encuentren los agentes en la misma máquina o en máquinas diferentes.

F. Otra información

No disponible.

G. Vistas relacionadas

Vistas *peer-to-peer* (ver sección 5.5.4) y vista de descomposición del sistema (ver sección 5.5.1.1)

5.6. Análisis de la Arquitectura

El propósito de esta sección es analizar la arquitectura descrita en las secciones anteriores para comprobar que cumple con las condiciones establecidas al comienzo del capítulo con los escenarios de calidad. El objetivo es, pues, ver si la arquitectura que se ha diseñado cumple los requisitos de modificabilidad y eficiencia que se desea que tenga el sistema.

Para ello, se analizará la arquitectura escenario a escenario, razonando, para cada uno de ellos, de qué manera cumple el diseño de la arquitectura con las necesidades descritas en ellos.

5.6.1. Escenarios de Modificabilidad

La mayoría de los escenarios de modificabilidad que aparecen a continuación están relacionados con la sustitución de un elemento del sistema por otro que cumple funciones similares pero de distinta manera.

A nivel teórico, siempre es posible realizar esta sustitución, independientemente de cómo esté implementado el nuevo elemento, a través del uso de lo que en (Hayden et al., 1999) se denomina *patrón Wrapper*, lo que informalmente se denomina *agentificar* un elemento para que a los ojos del resto de los agentes aparezca como un agente más.

Las mayores dificultades para realizar las sustituciones van a estar originadas principalmente por dos motivos. El primero de ellos es la incompatibilidad del nuevo elemento con los existentes, por el hecho de que

no proporcione la funcionalidad que el resto requiere. Y el segundo es la posibilidad de que esté implementado de tal manera que no se pueda adaptar al modelo utilizado basado en la prestación de servicios.

5.6.1.1. Escenario de Modificabilidad 1 Sustitución del Entorno Virtual

La arquitectura del sistema se ha diseñado forma tal que todos los subsistemas se comunican a través del Centro de Mensajes. Además, cada subsistema se suscribe a la recepción de los tipos de mensajes que desea recibir, pero no en función de quién realiza el envío de los citados mensajes.

De esta manera, no existe una comunicación directa entre el SIT y el EV, y la que tiene lugar, a través del Centro de Mensajes, se hace de manera anónima, utilizando un mecanismo de paso de mensajes.

Por lo tanto, siempre y cuando el EV que se incorpore sea capaz de manejar la comunicación por paso de mensajes a través del Centro de Mensajes, y siempre que esos mensajes tengan el mismo formato y sigan los mismos protocolos que utilizaba el EV que se sustituye, el intercambio de un EV por otro no debería afectar al resto de subsistemas.

Puesto que lo más probable es que el nuevo EV no se haya diseñado para funcionar junto con el SIT, lo más habitual será que aparezca la necesidad de modificar el EV para poder incorporarlo al sistema.

En concreto, lo normal es que surja la necesidad de realizar dos tipos de modificaciones. En primer lugar, hará falta crear un módulo de comunicación con el centro de mensajes que se encargue de realizar la comunicación con el resto de subsistemas que conforman la aplicación de entrenamiento. Aunque la comunicación se realice sólo a través del centro de mensajes, habrá que dotarlo con la capacidad para manejar todos los tipos de mensajes que manejaba el EV anterior.

En segundo lugar, es poco probable que el EV esté preparado para enviar a otras aplicaciones las acciones que tienen lugar dentro del mismo, razón por la cual se deberá modificar para que las acciones que realice el usuario se envíen al nuevo módulo de comunicación, además de tratarlas en el propio EV. También se deberán enviar las posiciones y las orientaciones de todos los elementos móviles que experimenten algún cambio en alguna de ellas.

De hecho, y dependiendo de la estrategia de tutoría utilizada, existe la posibilidad de que sea necesario que las acciones del estudiante cuenten con el permiso del SIT para que se puedan ejecutar, por lo que llegado el caso haría falta también integrar este mecanismo de validación previa en la interacción del estudiante con el EV. De no hacerlo así no sería posible hacer uso de este tipo de tutoría.

Esta necesidad puede surgir tanto si se sustituye el EV como si se cambia la estrategia de tutoría, por lo que constituye un punto sensible en cuanto a la modificabilidad del sistema. Si el EV se diseña con cuidado, el cambio no pasará de afectar a la secuencia de acciones que tienen lugar cuando el estudiante interactúa con el sistema. En caso contrario, probablemente habrá que modificar sustancialmente su diseño.

5.6.1.2. Escenario de Modificabilidad 2 Sustitución de la Estrategia de Tutoría

La estrategia de tutoría, a través del Agente de Tutoría, constituye la pieza fundamental de un sistema cuyo principal objetivo es, precisamente, la tutoría.

Tal y como se ha concebido la arquitectura, ningún agente depende de la presencia del Agente de Tutoría, lo cual, en apariencia, hace pensar que la sustitución de este elemento puede resultar sencilla. Sin embargo, aunque algunos de los mecanismos que se han introducido al diseñar la arquitectura buscan facilitar la modificabilidad de la misma, no se debe caer en la tentación de dar por hecho que cualquier cambio será fácil y rápido.

El SIT no deja de der un conjunto de agentes orquestados alrededor del Agente de Tutoría para satisfacer sus necesidades a la hora de ayudar al estudiante durante el entrenamiento. Así, la sustitución de la estrategia de tutoría puede suponer la necesidad de añadir nuevos agentes con capacidades que presten servicios que no se han considerado hasta el momento.

Por ejemplo, la introducción de una tutoría basada en el diálogo, como la de Autotutor (Graesser et al., 2005b), es probable que implique un elevado número de modificaciones en la arquitectura del sistema, ya que actualmente es un SIT orientado al entrenamiento y no está preparado para permitir el diálogo con el estudiante, más allá de la realización de preguntas acerca de los objetos del entorno y de las acciones a realizar. Por contra, la utilización de un elemento de tutoría como el de Steve (Rickel y Johnson, 1999) resultará más fácilmente integrable, aunque también requerirá la realización de ajustes en la arquitectura.

Como se verá en el siguiente apartado, cualquier cambio de estrategia de tutoría es susceptible de provocar modificaciones, cuando no la sustitución, del modelo del estudiante, ya que resulta fundamental para la personalización de la tutoría y, por tanto, existe una dependencia notable entre ambos elementos. Aunque no se produzcan diferencias en la manera en que ambos elementos se comunican, sí pueden producirse variaciones en la información que solicita el Agente de Tutoría y que el modelo del estudiante debe mantener.

En cuanto al resto de los agentes, puede suceder que no se requieran sus servicios, como es el caso del Agente de Trayectoria si la estrategia de tutoría no valora la adecuación de los desplazamientos del estudiante. También puede resultar sensible la relación con el Agente de Planificación. Si no se permite que el estudiante se desvíe del plan establecido, puede darse el caso de que el Agente de Tutoría suponga que se le va a entregar el plan completo de una vez, y así no tener que estar comunicándose continuamente con el Agente de Planificación. En ese caso, la modificación más sencilla será añadir un servicio a ese agente para que proporcione el plan completo, lo que también requerirá añadir un elemento que traduzca el plan a un formato manejable por el Agente de Tutoría.

Como se puede apreciar, la sustitución de la estrategia de tutoría no está exenta de producir ciertas modificaciones en otros elementos al ser ésta uno de los elementos fundamentales del sistema. Al sustituir la esencia del sistema, lo esperable es que otras partes del mismo se vean afectadas. Sin embargo, el impacto de estos cambios, que son susceptibles de producirse en cualquier situación, puede verse notablemente reducido a través de la utilización de servicios, siempre que el nuevo elemento introducido admita el funcionamiento utilizando este enfoque.

5.6.1.3. Escenario de Modificabilidad 3 Sustitución del Modelo del Estudiante

El modelo del estudiante es uno de los componentes del sistema que es susceptible de sufrir una mayor evolución. Poco a poco se van incorporando elementos que hacen un modelado más complejo de los estudiantes, ampliando la simple traza de sus acciones e incorporando elementos cada vez más relacionados con el mundo de la psicología, lo que incluye modelos de personalidad, de atención o de aprendizaje.

La evolución de estos modelos va pareja a la evolución de la estrategia de tutoría, que se irá adaptando al estudiante de una forma más compleja a medida que el modelo del estudiante vaya realizando un perfil más completo del mismo. Por lo tanto, como se mencionó en el apartado anterior, un cambio en la estrategia de tutoría es susceptible de producir cambios en la información que se necesita que proporcione el modelo del estudiante.

Las modificaciones en este modelo, y en concreto la sustitución del mismo, pueden llevar aparejados cambios en la estrategia de tutoría, debido a que el nuevo modelo del estudiante no incorpore elementos que el anterior sí incluía, o incorpore elementos diferentes o que se tratan de otra manera. No obstante, una de las principales razones para sustituir un modelo de estudiante por otro es precisamente que se deseen realizar modificaciones en la estrategia

de tutoría para incorporar las funcionalidades proporcionadas por el nuevo modelo del estudiante, razón por la cual muchas de las modificaciones a realizar no serán consecuencia del cambio, sino una de sus causas.

En los casos mencionados, las modificaciones son inherentes a la propia estrategia de tutoría o a la funcionalidad proporcionada por el modelo del estudiante. Siempre que la estrategia de tutoría y el modelo de estudiante que se utilizan sean compatibles, la arquitectura permite que cada uno de ellos publique y utilice los servicios que necesite. En caso de que no sean compatibles, el problema que se produce es más a nivel teórico que arquitectónico, por lo que la solución que pueda proporcionarse en lo referente a la arquitectura estará supeditada a la existencia de una solución conceptual aceptable.

Lo más importante en este punto es dejar patente la estrecha relación existente entre la estrategia de tutoría y el modelo del estudiante. Con unos objetivos de modificabilidad distintos, o sin ellos, la arquitectura resultante sería diferente y podría no presentar esta relación tan estrecha entre ambos elementos. Sin embargo, a la luz de las condiciones con las que se trabaja, lo más conveniente es resaltar este hecho, ya que se produce y es una potencial fuente de dificultades. Por ello, se debe dejar la arquitectura preparada para introducir los cambios que puedan surgir, siempre y cuando no sean los cimientos conceptuales los que causen que ambos elementos no puedan trabajar de manera conjunta.

5.6.1.4. Escenario de Modificabilidad 4 Sustitución del Planificador

Una de las principales razones por las que se decidió rediseñar la arquitectura descrita en el capítulo 4 se debió a la dificultad para sustituir el planificador utilizado por otro con unas características diferentes, más sofisticadas, desarrollado por otro grupo de investigación.

En ese momento quedó claro que el planificador es uno de los elementos susceptibles de ser sustituidos, y que además esa sustitución puede afectar, como lo hizo, a muchos otros elementos del sistema.

En la arquitectura descrita en el presente capítulo, el planificador es un elemento controlado por el Agente de Planificación, y ningún otro agente tiene relación con él salvo el Agente de Tutoría, que le envía al de Planificación las acciones realizadas por el estudiante.

La labor del Agente de Planificación consiste en atender las peticiones del Agente de Tutoría y en realizar labores de traducción entre éste y el planificador, de manera que los operadores manejados por los agentes y los EVs no dependan, en la medida de lo posible, del lenguaje utilizado por el

planificador para definir el dominio del problema y las acciones que se pueden realizar en el mismo.

Éste constituye uno de los puntos sensibles en lo tocante al planificador. La posible necesidad de adaptar los operadores manejados por los agentes al planificador utilizado puede afectar a varios de los agentes del sistema.

Por tanto, una de las principales labores de diseño del sistema consistirá en cuidar el formato de los operadores que representan las acciones de los estudiantes, de manera que la traducción que debe realizar el Agente de Planificación cuente con todos los elementos necesarios para convertirlos en operadores del planificador. De forma paralela se debe evitar propagar datos innecesarios o inconvenientes al resto de los agentes o, incluso, a los EVs, que son los que, en definitiva, generan los operadores relativos a las acciones de los estudiantes.

5.6.1.5. Escenario de Modificabilidad 5 Sustitución del Planificador de Rutas

El planificador de rutas es un elemento que incluyen algunos de los sistemas que requieren que el alumno o el tutor virtual se desplacen por el mundo virtual.

En los casos más sencillos, como sucede con Steve (Rickel y Johnson, 1999), el cálculo de rutas se realiza a través de un grafo que indica qué puntos significativos del mundo son accesibles desde otros. En casos algo más complejos, el uso de los grafos evoluciona hacia el uso de algoritmos como el de Dijkstra (Russell y Norvig, 1995).

En otros casos, como en (Sud et al., 2007; Sud et al., 2008), en lugar de trabajar con lugares significativos del mundo para definir nodos de un grafo, se trabaja con diagramas de Voronoi (Aurenhammer, 1991), lo que hace posible que se puedan utilizar algoritmos más precisos e, incluso, trabajar con entornos cambiantes en los que no siempre se puede seguir la misma ruta para recorrer el camino entre dos puntos.

En cualquiera de los dos casos, si el planificador de rutas no recibe directamente del Agente de Comunicación la información que necesita, existe la posibilidad de que el Agente Mundo, que es el encargado de mantener el modelo del EV que manejan los agentes, proporcione la coordenada correspondiente a un lugar significativo o viceversa.

Esto implica que, de no ofrecer un servicio similar, el cambio del planificador de rutas puede suponer la necesidad de actualizar el Agente Mundo para que pueda realizar la mencionada traducción entre lugares y coordenadas.

5.6.1.6. Escenario de Modificabilidad 6 Sustitución del Simulador

La sustitución del simulador es una tarea que habrá que realizar de manera regular cada vez que se cambie el dominio en el que se realiza el entrenamiento. Un simulador, en general, es específico de un dominio de entrenamiento determinado, incluso de una parte concreta de un dominio específico, razón por la que puede constituir uno de los elementos que presenten una mayor variabilidad dentro del sistema.

La naturaleza de los simuladores puede ser enormemente variada, y en función de cómo esté implementado y de lo que suceda dentro de él habrá que integrarlo en el sistema de muy distintas maneras. Por este motivo, dentro del alcance del presente trabajo resulta difícil plantear una solución lo suficientemente abierta como para poder aventurarse a decir que se podrá integrar con el resto de subsistemas que conforman la arquitectura sin hacer grandes modificaciones.

Como principales opciones se han considerado dos situaciones distintas. En primer lugar, que el simulador sea de una complejidad relativamente pequeña, en cuyo caso puede formar parte del Agente de Simulación, comunicándose directamente con él. En este caso, el Agente de Simulación se comportaría como un *Wrapper* (Hayden et al., 1999), transformando el simulador en un agente a ojos de los demás agentes. Una segunda opción consiste en que el simulador sea una aplicación autónoma de complejidad mayor, en cuyo caso puede comunicarse con el SIT a través del centro de mensajes.

Entre las situaciones que se han contemplado aparece la posibilidad de que el simulador sea de pequeño tamaño pero realice frecuentes actualizaciones que deban verse reflejadas en el EV, en cuyo caso no interesa que se comunique directamente a través del Agente de Simulación, pues puede llegar a inundar el SIT con sus mensajes. Otra posible situación consiste en que el simulador esté integrado junto con un EV, en cuyo caso se debe tener también en cuenta si el EV satisface el resto de necesidades del sistema y, si es posible, hacer que los mensajes salgan del sistema para que puedan ser utilizados por el resto de elementos del sistema de entrenamiento.

Como puede apreciarse, la cuestión es lo suficientemente abierta como para encontrar una solución que satisfaga todas las posibles situaciones, o incluso como para poder decir que será fácil realizar las modificaciones que sean necesarias en cada uno de los casos. Por ello, la integración de un simulador queda como cuestión parcialmente abierta para ser estudiada como una posible creación de una línea de producto (Clements y Northrop, 2001).

5.6.1.7. Escenario de Modificabilidad 7 Deshabilitación de la Tutoría

La deshabilitación de la tutoría es una funcionalidad de la aplicación que además está relacionada con la capacidad de modificación de la misma en tiempo de ejecución.

La forma más inmediata de hacerlo consiste en no iniciar el Agente de Tutoría cuando comience a ejercitarse la actividad. De esta manera, al no estar presente este agente, las labores de tutoría quedan suspendidas, ya que no realizará ninguna de ellas y no solicitará a ninguno de los agentes restantes que realicen las tareas asociadas a ellas. Por este motivo, de deshabilitarse así la tutoría, será posible deshabilitar también otros agentes, como el de planificación.

Sin embargo, al desactivar los agentes completos también se desactivan otras acciones que podemos querer que se realicen, como la respuesta a preguntas del estudiante. Por ese motivo, se considera más conveniente la desactivación de comportamientos de manera selectiva.

Una tercera opción consiste en hacer que el agente de tutoría no se suscriba a la recepción de acciones del estudiante. De esta manera, el resto de funciones seguirán activas, pero no se realizará el seguimiento de las acciones del estudiante.

Cualquiera de las tres opciones es sencilla de realizar con la arquitectura propuesta, aunque la primera de ellas es la que entraña más riesgos, ya que puede provocar problemas inesperados al desactivar funciones necesarias de forma inadvertida.

5.6.1.8. Escenario de Modificabilidad 8 Deshabilitación del Cálculo de Trayectorias

Al igual que sucedía en el caso anterior, existe más de una forma en la que se puede desactivar el cálculo de trayectorias dentro del entrenamiento de una actividad.

Puesto que el cálculo de trayectorias lo activa y desactiva el Agente de Tutoría cuando solicita este servicio del Agente de Trayectoria, el hecho de que la actividad no lleve aparejado el desplazamiento por el entorno es una de las formas de hacer que el Agente de Tutoría no solicite este servicio.

Aunque, de manera análoga al caso anterior, se podría pensar en desactivar el Agente de Trayectoria completo, en este caso existe la posibilidad de que haya otros agentes que utilicen los servicios que presta, por lo que la opción más factible en este caso es que simplemente no se solicite la prestación del servicio.

5.6.2. Escenarios de Eficiencia

La velocidad en la ejecución de un Entorno Virtual de Entrenamiento es un aspecto clave para conseguir el éxito en la experiencia del estudiante, ya que una respuesta pobre ante sus acciones puede desembocar en frustración o aburrimiento y, por ende, en el abandono del entrenamiento.

Entre las causas de una ejecución pobre se encuentran, por un lado, el hecho de que el EV en sí funcione despacio o a saltos, por lo que es precisa una programación cuidadosa del mismo para que la ejecución se realice de manera suave. Es por ello que uno de los requisitos de modificabilidad tiene que ver con la posibilidad de incorporar EVs desarrollados por terceras partes que estén especializadas en la implementación de este tipo de sistema.

La otra causa se encuentra en respuestas con excesivo retardo ante las acciones del estudiante, lo cual puede llegar a provocar que el usuario realice varias acciones antes de recibir la respuesta a la primera de ellas. Esto, a su vez, puede originar confusiones, ya que el estudiante puede no identificar a cuál de las acciones corresponde la respuesta que acaba de recibir.

A estas dos potenciales fuentes de dificultades es a las que se refieren los dos escenarios de eficiencia que se discuten a continuación.

5.6.2.1. Escenario de Eficiencia 1 Respuesta ante Acciones

Parte del éxito en la consecución de una buena velocidad de ejecución se debe al elemento tecnológico, a través del uso de máquinas de una potencia elevada y redes de comunicación de alta velocidad.

Por otra parte, el hecho de que los agentes trabajen de manera concurrente posibilita que la velocidad de ejecución sea más fluida, especialmente si la máquina donde se ejecutan permite el procesamiento en paralelo o si la plataforma utilizada facilita la distribución del conjunto de agentes en varias máquinas.

Además, el orden de ejecución de las acciones también determina que, cuando se reciba una acción del estudiante, esté todo preparado para evaluarla, y cuando se haya hecho esto, se terminará de trabajar con ella y se preparará la llegada de la siguiente.

De esta forma, antes de que llegue una acción, el agente de tutoría debe haber solicitado al agente de planificación la siguiente acción correcta. Cuando llegue la acción se deberá validar y enviar la respuesta al estudiante, tras lo cual se evaluará la corrección de la misma para dársela al Agente de Estudiante y se solicitará la siguiente acción al Agente de Planificación para iniciar nuevamente el ciclo.

Llegado el caso en que se perciba que la velocidad de respuesta es algo lenta, se ha pensado en incorporar una pequeña mejora, que consiste en evaluar de antemano la acción que debería realizar el estudiante, de forma que si es esa la acción que se realiza, la respuesta será inmediata.

Si el estudiante se equivoca mucho, en muchos casos se realizarán el doble de evaluaciones, por lo que el SIT estará más cargado. En ese caso, se puede complementar esta técnica con consultas al modelo del estudiante, de manera que si el estudiante no se suele equivocar se realice la validación por adelantado y, en caso contrario, se espere a que llegue la acción.

Para hacer esto último, bastaría con reutilizar uno de los servicios que ya presta el Agente de Estudiante, aunque con unos fines distintos a los previstos originalmente. Esto da una muestra de una posible forma de reutilizar funcionalidades proporcionadas por los agentes.

Otra opción consistiría en la desactivación de la validación de las acciones, lo cual condicionaría la estrategia de tutoría a utilizar. Además, esta alternativa impone restricciones en las opciones de interacción disponibles en el EV, que deben ser siempre válidas, aunque no sean correctas.

5.6.2.2. Escenario de Eficiencia 2 Respuesta en Navegación

Cuando el estudiante se desplaza por el EV, sus coordenadas se envían continuamente al SIT para que pueda supervisarse su trayectoria.

Aunque el envío de coordenadas es un paquete de tamaño reducido, la ejecución continuada puede provocar que se envíe al SIT demasiada información, en ocasiones innecesaria, y que se sobrecargue ligeramente al cliente, lo cual puede constituir un factor de retardo en la ejecución.

El EV debe permitir decidir cada cuánto tiempo se quieren enviar las coordenadas al SIT, de manera que pueda mantenerse un equilibrio entre el número de coordenadas que se envían para reproducir la trayectoria fielmente y la velocidad de ejecución del mismo.

Teniendo en cuenta que lo habitual es que los estudiantes se desplacen por el EV andando en lugar de corriendo, lo normal es que para reproducir fielmente la trayectoria seguida no hagan falta más de 10 coordenadas por segundo, lo cual da un grano bastante fino para seguir los movimientos.

Llegado el caso, puede implementarse un sencillo mecanismo que regule cada cuánto tiempo se envían coordenadas en función de la velocidad de ejecución del EV.

También puede suceder que el EV envíe más coordenadas de las que el Agente de Trayectoria puede o debe procesar, por lo que éste debe disponer de un mecanismo que le permita descartar coordenadas.

5.7. Reflexiones

El diseño de la arquitectura mostrada en el presente capítulo ha supuesto un trabajo dirigido a conseguir las propiedades de modificabilidad y eficiencia que se buscan para la arquitectura del sistema.

Para cada decisión de diseño que se ha tomado se ha prestado especial atención a tener en cuenta si servía a los fines descritos en los escenarios de calidad definidos al principio del capítulo. Para ello, se ha hecho uso de técnicas que otros autores han identificado como útiles o necesarias para que una aplicación se pueda modificar con facilidad en determinadas situaciones.

La principal de las técnicas utilizadas, aunque quizá la menos concreta por cuanto existen diversas formas de usarla, es la introducción de relaciones ambiguas (Álvaro García y Medinilla, 2007) entre los elementos que forman parte de la arquitectura, tanto entre los distintos subsistemas que conforman la aplicación como entre los agentes que integran el SIT. Esta introducción de ambigüedad en el diseño consiste en hacer que los distintos elementos sean indiferentes a determinadas características de los elementos con los que se deben comunicar.

Una de las formas de llevar esto a la práctica ha sido, en el caso del SIT, mediante la combinación de un mecanismo de edición-suscripción más las páginas amarillas. El mecanismo de edición-suscripción posibilita que los productores de información o eventos no necesiten conocer directamente a los consumidores de los mismos. Por otro lado, el uso de las páginas amarillas hace posible que los consumidores tampoco tengan que conocer la identidad de los productores. La combinación de ambas técnicas debilita mucho la relación entre productores y consumidores, lo que redunda en una mayor facilidad para sustituir tanto unos como otros, siempre que la semántica y la manera de invocar los servicios que los relacionan no varíe.

Otra de las técnicas utilizadas para facilitar la modificabilidad de algunas partes del sistema ha consistido en la *agentificación* de determinados componentes de la arquitectura propuesta, como el simulador o el planificador, utilizando lo que se denomina un patrón *Wrapper* (Hayden et al., 1999). Al usar el mecanismo explicado anteriormente, a ojos de los demás agentes podrán sustituirse como si fuesen un agente más, por lo que el trabajo de la sustitución involucrará principalmente al agente que realiza las funciones de *wrapper*.

El uso de ficheros de configuración es otra de las herramientas que posibilita la mayor facilidad de modificación del sistema, hasta el punto de que, en unos pocos minutos, se pueden cambiar, por ejemplo, parámetros relativos a la estrategia de tutoría utilizada, haciéndola más permisiva o proporcionando más elementos de ayuda.

Finalmente, el hecho de que la comunicación se realice a través de paso de mensajes asíncrono también independiza a unos agentes de otros, ya que no es necesario esperar la recepción del mensaje ni la respuesta del agente al que se le envía, sino que los agentes pueden enviar un mensaje y continuar con sus tareas sin preocuparse del agente al que le han enviado el mensaje.

Esto, además, nos permite pensar que el uso de agentes puede ser un factor que facilite la modificabilidad del sistema por una razón, que es que lleva aparejados una serie de mecanismos de utilidad, como algunos de los que se acaban de mencionar: paso de mensajes asíncrono y páginas amarillas. Esto no quiere decir que sólo por usar agentes el sistema vaya a ser más modificable. Habrá que hacer uso de estos mecanismos para que de verdad el sistema sea más modificable, pero el hecho de que los agentes proporcionen estos mecanismos favorece su uso, lo cual puede que no se hiciese si hubiese que implementarlos desde cero para añadírselos al sistema.

Respecto a los métodos utilizados para diseñar la arquitectura, el resultado ha sido un poco desigual, y quizá han resultado de menos utilidad de lo que se esperaba.

En lo que se refiere a QAW (*Quality Attribute Workshops*), desde luego ha resultado útil para poner de manera explícita lo que se esperaba obtener de la arquitectura. Sin embargo, su utilidad ha resultado relativa por varios motivos. En primer lugar, la experiencia previa hacía que el equipo involucrado en el desarrollo de este sistema tuviese bastante claro hacia dónde quería avanzar, por lo que no se ha obtenido mucha información que no se conociese ya. Tampoco existían conflictos de intereses que necesitasen de algún tipo de priorización de los atributos de calidad del sistema, por lo que tampoco se ha sacado mayor partido a esa característica de QAW. Para lo que sí ha servido, cuando menos, es para dejar por escrito de una manera medianamente formal lo que se requería de la arquitectura en lo tocante a los atributos de calidad.

Una característica de QAW es que es un método cuyos resultados dependen de los participantes en las reuniones. Esto puede parecer un inconveniente a priori, pues con otros participantes, o con más participantes, se podría obtener más información. Sin embargo, no es éste el objetivo del método, ya que lo que se persigue no es hacer un listado exhaustivo de los atributos de calidad que podría satisfacer el sistema. Por contra, lo que se pretende es identificar qué escenarios de calidad son relevantes para el proyecto, y esos son los que sacan a la luz los participantes en las sesiones de QAW. De esta manera, este método proporciona lo que se quiere obtener.

En lo concerniente a ADD (*Attribute Driven Design*), por contra, la sensación tras su uso es que no ha resultado de especial utilidad por el hecho de ser un método basado en la descomposición.

A ojos del autor de este trabajo, para que una descomposición sea efectiva debe dar lugar a un número manejable de elementos. De otro modo, resulta complicado manejarla, asignar la responsabilidad adecuada a todos los elementos resultantes, definir sus interfaces y las relaciones entre ellos. Además, se hace necesario disponer de mucha información para no tener que rehacer la descomposición demasiadas veces a medida que se realiza el diseño de la arquitectura.

Adicionalmente, tal vez por los "vicios" adquiridos a lo largo de los años, realizar un diseño a través de una descomposición no resulta natural, o al menos no es la forma en la que el autor y muchos otros diseñadores trabajan. La costumbre de diseñar de una manera iterativa e incremental hace que parezca más adecuado proceder de la misma manera en lo tocante al diseño de la arquitectura. De esta manera, en lugar de descomponiendo, la arquitectura se puede obtener por el proceso inverso, es decir, componiendo. Para ello, se puede comenzar diseñando una arquitectura muy pequeña, de dos o tres módulos, a la que poco a poco se le vayan añadiendo más módulos hasta que se hayan tenido en cuenta los mismos factores que en el caso de utilizar ADD.

Esta visión se ve apoyada por otro hecho. Si suponemos que disponemos de un sistema en el que ya se ha diseñado la arquitectura utilizando ADD, llegará un momento en el que a ese sistema se le quieran realizar ampliaciones. Los nuevos módulos que se pretendan añadir no se pueden introducir en el diseño de la arquitectura siguiendo un proceso de descomposición, puesto que es muy posible que no haya nada que descomponer. Por el contrario, habrá que decidir cómo componerlos con el resto de elementos que ya forman parte de la arquitectura.

La cuestión que se plantea entonces es que si esto se puede hacer cuando se quiere ampliar el sistema, ¿a partir de qué momento es posible hacerlo? Si la respuesta fuese que a partir de que tengamos un sistema implementado, la siguiente cuestión sería ¿qué considero como mi sistema y qué puedo considerar como una extensión? Y por tanto, ¿por qué no hacerlo desde el principio?

La conclusión respecto a este punto es que ADD nos lleva ligeramente atrás en el tiempo, a una época en la que era más habitual realizar descomposiciones. Sin embargo, con los sistemas que se desarrollan actualmente, es necesario incorporar otra visión al diseño de los sistemas, y en concreto a su arquitectura.

Aún queda otro punto a tener en cuenta respecto a los atributos de calidad para dirigir el diseño. Como se ha mencionado al principio del capítulo, no se ha tenido en cuenta para el diseño de la arquitectura ningún atributo de calidad relacionado con la usabilidad.

Esto es debido a que, para un Sistema Inteligente de Tutoría, algunos de los que se consideran atributos de usabilidad constituyen funcionalidades propias de la aplicación. Son, por ejemplo, la adaptación al usuario o la posibilidad de deshacer acciones. Estas funcionalidades pueden estar o no presentes en el SIT, e incluso pueden activarse y desactivarse a voluntad, bajo el control de la estrategia de tutoría. Por eso, por el hecho de ser características que no se desea que estén siempre presentes, se ha preferido considerarlas sólo desde el punto de vista de la funcionalidad, y no como atributos de calidad del sistema.

Sin embargo, queda abierta la posibilidad de utilizar atributos de usabilidad que dirijan también el diseño de la arquitectura, para comprobar si esto tiene un efecto beneficioso sobre el resultado.

Otra reflexión que nos ha asaltado durante el diseño y documentación de la arquitectura está relacionada con la necesidad o la utilidad de introducir agentes en el diseño de la arquitectura. Por un lado, en (Bass et al., 2003) se dice que el diseño de la arquitectura es independiente de las tecnologías utilizadas para implementar el sistema. Por otro lado, en algunas de las revisiones realizadas sobre las publicaciones relacionadas con este trabajo se realizaba el mismo comentario.

Después de analizarlo desde varios puntos de vista, surgen distintos motivos que apoyan la inclusión de los agentes ya en esta parte del diseño.

El primero y menos razonado lo constituye el hecho de que estamos trabajando con sistemas complejos, por lo que las interacciones que pueden surgir entre los distintos factores que influyen en la arquitectura hacen que difícilmente se pueda asegurar la independencia de ninguno de ellos respecto de los demás.

El segundo, quizá sin demasiado peso pero que no deja de ser cuando menos curioso, lo constituye el hecho de que los mismos que abogan por la independencia de la arquitectura respecto de otros factores, proponen como posibilidad para documentar la arquitectura una *Vista de Generalización* (Clements et al., 2002a), adecuada para sistemas orientados a objetos (*sic*).

El tercero lo constituye el hecho de que, sin haber utilizado agentes, probablemente no se hubiese planteado una solución basada en servicios, con un mecanismo de edición-suscripción y una comunicación mediante paso de mensajes asíncrono.

Especialmente este último punto apoya la idea de que la solución planteada ha sido posible por el hecho de haber introducido los agentes en el diseño de la arquitectura, y por lo tanto sí se considera necesario haber tenido en cuenta este factor para realizar el diseño.

Finalmente, queda por hacer una última reflexión respecto al diseño obtenido. Como resultado de la búsqueda de modificabilidad en el sistema se

ha llegado a una solución en la que los agentes del SIT interactúan a través de la solicitud de realización de unos determinados servicios. El resultado es similar a una arquitectura orientada a servicios obtenida mediante la aplicación de técnicas que intentan conseguir modificabilidad, y no como solución elegida a priori.

A la vista del resultado, cabe la posibilidad de explorar el área de las Arquitecturas Orientadas a Servicios (SOA, del inglés *Service Oriented Architecture*) para comprobar en qué medida influyen en la modificabilidad del sistema y para estudiar si los métodos de diseño desarrollados en ese área pueden ser útiles para diseñar sistemas basados en agentes en general y Sistemas Inteligentes de Tutoría en particular.

Capítulo 6

Recomendaciones Metodológicas y Criterios de Diseño

"Me labré así una gran reputación; pero el único motivo era que mi juego de herramientas era distinto del de los demás"
— Richard Feynman

6.1. Introducción

A lo largo de la elaboración del presente trabajo de tesis doctoral se ha podido comprobar que uno de los defectos de los que suelen adolecer las metodologías de desarrollo de software, especialmente las orientadas al desarrollo de software con agentes, es que prestan poca atención al diseño de la arquitectura del sistema (Bass et al., 2003).

Si bien es cierto que los pasos marcados por la metodología nos conducen de una especificación inicial a un producto que cumple de manera más o menos satisfactoria la mencionada especificación (Pressman, 2004; Sommerville, 2007), también lo es que el resultado final depende en gran medida de las habilidades de la persona o personas que estén utilizando la metodología para desarrollar el sistema software.

Como se ha podido ver en capítulos anteriores, uno de los aspectos que suelen quedarse sin considerar son los atributos de calidad con los que se desea dotar al sistema (Bass et al., 2003), al dar por sentado que no son necesarios, que aparecerán solos, o que el desarrollador será capaz de obtenerlos porque la metodología allana suficientemente el camino para ello.

Por este motivo, además del diseño de una arquitectura software para Entornos Virtuales de Entrenamiento con Tutoría Inteligente, se ha pensado que el presente trabajo debe aportar una herramienta que complemente a

la arquitectura y, sobre todo, que ayude en el desarrollo de este tipo de sistemas allí donde las metodologías dejan al desarrollador a expensas de sus habilidades.

Esa aportación adicional que se proporciona con este trabajo son una serie de recomendaciones metodológicas, obtenidas a partir del diseño de la arquitectura expuesta en el capítulo 5, y que constituyen las bases que han permitido llegar hasta la solución descrita. Estas recomendaciones metodológicas deben entenderse como criterios de decisión que ayuden al diseño del tipo de sistemas objeto de estudio allí donde las metodologías no cubren estos aspectos.

Si se ha conseguido dotar al sistema de la modificabilidad que se deseaba, y así se ha pretendido hacer y demostrar, entonces es de esperar que en un periodo de tiempo relativamente corto el diseño de la arquitectura sea distinto del expuesto en este trabajo. Este hecho no podrá considerarse un fracaso. Muy al contrario, no será sino la demostración del éxito logrado en los objetivos de modificabilidad propuestos, los cuales se han conseguido, en gran parte, gracias a las recomendaciones que se recogen en el resto del presente capítulo.

6.2. Recomendaciones Metodológicas

Una de las herramientas que más se ha echado en falta en el proceso de elaboración de la arquitectura objeto del presente trabajo es una guía que ayude en la elaboración de la misma.

Aunque el método ADD (Bass et al., 2003) constituye un buen punto de partida, en poco tiempo quedó patente que aportaba una visión sobre el diseño de arquitecturas que no resolvía las necesidades de la presente investigación. De hecho, en (Haythorn, 1994) ya se apunta a los inconvenientes que entraña esta forma de diseñar, que si bien puede ser adecuada como técnica de programación, no tiene por qué dar el mismo resultado al definir una arquitectura.

Por ello, basándose en el citado método, se ensayó una manera distinta de abordar el problema que, en vista de los resultados obtenidos, constituye una aproximación más adecuada para una tarea como la abordada en este trabajo.

De esta manera, y debido a su utilidad para la definición de la arquitectura descrita en el Capítulo 5, a continuación se presentan unas recomendaciones metodológicas que servirán de ayuda para extender la arquitectura que constituye la solución al problema planteado en este trabajo.

En primer lugar se describe la manera de decidir qué elementos de la arquitectura utilizar para construir un nuevo sistema. Seguidamente se describe la forma de introducir nuevos elementos en la arquitectura general del sistema, y a continuación se muestra la manera de llevar a cabo la misma tarea dentro de la arquitectura del Sistema Inteligente de Tutoría.

6.2.1. Selección de Componentes de la Arquitectura

A la hora de desarrollar un Entorno Virtual de Entrenamiento a partir de la arquitectura presentada en el Capítulo 5, resulta complejo asimilar toda la información presente en la documentación generada. Por ese motivo, si no se necesita utilizar todos los componentes descritos, cabe la posibilidad de que se cometan errores a la hora de decidir qué componentes resultan necesarios y cuáles no.

Con objeto de facilitar la labor de quien desee llevar a cabo esta tarea, en esta sección se propone la utilización de una lista de comprobación que ayude a decidir qué elementos de la arquitectura utilizar y cuáles se pueden descartar.

Mediante la selección de unas determinadas opciones y la respuesta a una serie de preguntas se le indicará a la persona encargada de definir la arquitectura cuáles de los elementos descritos le resultarán de utilidad en el nuevo sistema a construir.

Para poder identificar qué elementos será necesario incluir en la arquitectura se ha recurrido a la documentación generada para describir la arquitectura propuesta en el capítulo 5, en concreto a la relativa a dos de las vistas: la vista de uso (pag. 233) y la vista de edición-suscripción (pag. 247).

Según indican los autores de (Clements et al., 2002a), la vista de uso documenta la relación de uso existente entre los diferentes elementos de la arquitectura. Entre otras utilidades, la vista de uso puede utilizarse para dirigir el proceso de desarrollo, ya que se pueden identificar los elementos que deben estar presentes para que otro elemento dado pueda funcionar.

Con este objetivo, se ha elaborado un diagrama que muestra una vista conjunta de las relaciones de uso que se dan entre los agentes que forman el SIT (Figura 6.1), lo que a su vez servirá para identificar, dadas unas necesidades, qué elementos deben incluirse para satisfacerlas.

Una vez identificados los agentes que deben estar presentes, se han utilizado las tablas de la sección 5.5.3 para seleccionar los servicios que deben ofrecer cada uno de ellos.

En las mencionadas tablas se mostraban los servicios de los que hace uso cada uno de los elementos que forman la arquitectura propuesta, junto con los elementos que los prestan. De esta forma, una vez identificados los

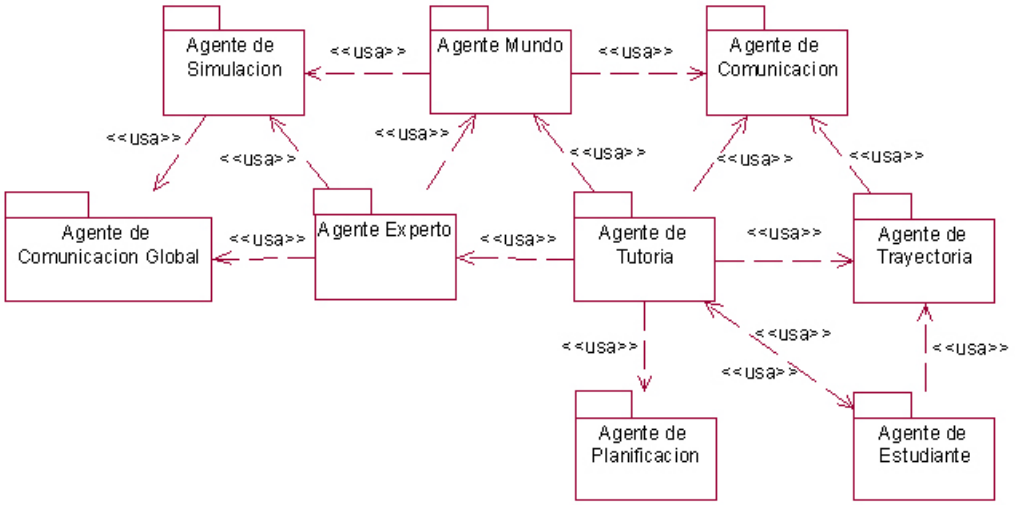

Figura 6.1: Vista de Uso de los agentes del SIT

elementos que deben estar presentes a través de las vistas de uso, se puede hacer uso de estas tablas para decidir qué servicios debe implementar cada uno de ellos.

Teniendo en cuenta la utilidad de ambas vistas, se propone la utilización de las listas de comprobación que se muestran a continuación para seleccionar los elementos que deben incluirse en la arquitectura de la aplicación.

Con la elaboración de estas listas de comprobación no se pretende ser exhaustivo y cubrir todos los casos que se puedan presentar, lo cual resultaría inmanejable. Por el contrario, sólo se pretenden cubrir las situaciones habituales.

Tanto en uno de estos casos como para cubrir situaciones más específicas, se puede hacer uso de un modelo como el mostrado en la Figura 6.2. En ella se muestran los agentes junto con sus comportamientos y los servicios que ofrece cada uno de ellos, en una combinación de algunas de las vistas utilizadas para documentar la arquitectura: descomposición, uso y edición-suscripción.

Sobre esta vista del SIT se pueden ir marcando los servicios que se necesiten. Una vez marcado un servicio, se debe marcar también el comportamiento que lo proporciona y el agente al que pertenece, así como el agente que lo solicita.

Los servicios que no solicita directamente ningún agente se ejecutan como respuesta a un mensaje recibido a través de una suscripción. Así, cuando un agente se suscriba a un servicio habrá que marcar los servicios de ese agente que se ejecutan cuando llega el mensaje correspondiente.

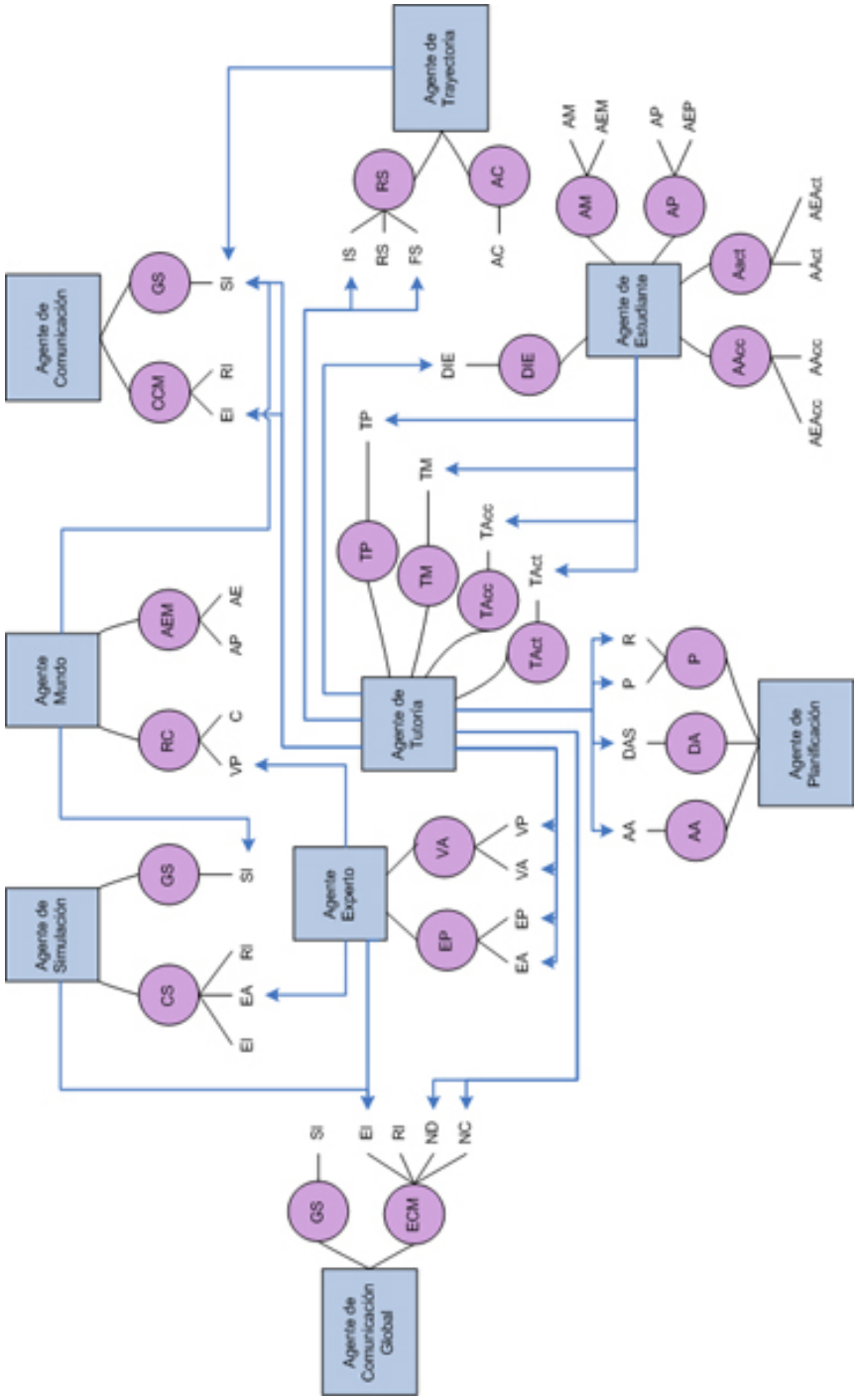

Figura 6.2: Vista de Descomposición y Uso del SIT

De esta manera, se puede obtener una primera visión de los elementos de la arquitectura que serán necesarios para la implementación de un sistema concreto.

En cualquier caso, se recomienda la consulta de la documentación de la arquitectura para llevar a cabo una selección final de los elementos de la arquitectura a utilizar.

A continuación se muestran las listas de comprobación que se han elaborado para ayudar a realizar la selección de elementos de la arquitectura.

Selección de Subsistemas

1. Si se necesita disponer de tutoría inteligente, utilizar el Sistema Inteligente de Tutoría.
2. Si se va a interactuar a través de un EV, añadir este elemento a la arquitectura.
3. Si existe alguna aplicación que simule el comportamiento de elementos del EV, añadir el Simulador.
4. Si se han seleccionado al menos dos de las anteriores, utilizar el Centro de Mensajes. Opcionalmente, utilizarlo también si sólo se seleccionó una.

Selección de Agentes, comportamientos y servicios

1. Incluir los Agentes de Comunicación y Comunicación Global, junto con sus comportamientos y servicios. Si no se van a enviar respuestas a los estudiantes no incluir los servicios de `Enviar Información`.
2. Si se va a hacer uso de un simulador, seleccionar el Agente de Simulación, junto con sus comportamientos y servicios.
3. Si se va a realizar una traza de las acciones de los estudiantes, añadir al Agente de Tutoría y al Agente de Estudiante. Para éste, añadir los comportamientos de `Anotar Acción` y `Anotar Actividad`, así como los servicios del mismo nombre.
4. Si se va a realizar tutoría sobre las acciones de los estudiantes, añadir los comportamientos de `Tratar Acción` y `Tratar Actividad` del Agente de Tutoría, así como los servicios del mismo nombre. Para el Agente de Estudiante, añadir también los servicios de `Anotar Evaluación de Acción` y `Anotar Evaluación de Actividad`.

5. Si se permite que el alumno pueda resolver una actividad por distintos caminos, incluir el Agente de Planificación junto con sus servicios. Añadir también los Agentes Mundo y Experto, junto con sus comportamientos y servicios para validar precondiciones y ejecutar postcondiciones y acciones.

6. Si se va a permitir la realización de preguntas, incluir el comportamiento de `Tratar Pregunta` del Agente de Tutoría, el de `Anotar Pregunta` del Agente de Estudiante y el servicio de `Consultar` del Agente Mundo. En caso de haber incluido un Simulador, si dispone de la capacidad de proporcionar información para responder preguntas, añadir el Agente de Simulación, junto con el servicio `Solicitar Información`.

7. Si se va a realizar el seguimiento de las trayectorias del estudiante, añadir el Agente de Trayectoria, junto con sus comportamientos y servicios. Añadir también el comportamiento de `Tratar Movimiento` del Agente de Tutoría y el de `Anotar Movimiento` del Agente de Estudiante.

8. Si se va a personalizar la tutoría según las características del estudiante, añadir el comportamiento de `Dar Información del Estudiante` del Agente de Estudiante.

6.2.2. Realización de Modificaciones en el Sistema

La introducción en la arquitectura de un nuevo subsistema, como podría ser un Entorno Virtual diferente u otra aplicación que deba comunicarse con los Entornos Virtuales, el Simulador o el Sistema Inteligente de Tutoría, requiere la realización de una serie de tareas que se detallan a continuación.

En primer lugar, se debe verificar que el nuevo subsistema permite realizar una comunicación basada en paso de mensajes a través del Centro de Mensajes, mostrado en la Figura 5.15 y que se reproduce a continuación para mayor comodidad. En caso de no ser así, se deberá introducir un elemento de nueva creación que sea capaz de realizar la comunicación entre el nuevo subsistema y el Centro de Mensajes.

Después deberán revisarse los escenarios de calidad existentes para comprobar si la introducción del nuevo subsistema interfiere con alguno de ellos. Además, se deberán elaborar nuevos escenarios de calidad en caso de que la introducción del nuevo subsistema lo requiera.

A continuación se deberán listar los tipos de mensajes que los subsistemas existentes envían al Centro de Mensajes, y extraer de entre ellos los que el nuevo subsistema debe recibir, de manera que pueda realizarse la suscripción

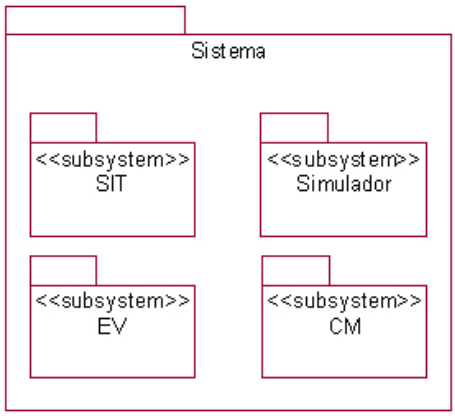

Figura 6.3: Descomposición del sistema

a los mismos en el Centro de Mensajes. También se deberá verificar que el nuevo subsistema es capaz de tratar esos mensajes. En caso de que no lo sea, se deberán introducir las modificaciones pertinentes para que lo haga.

Posteriormente, se deben analizar las necesidades de los subsistemas existentes que puede satisfacer el nuevo subsistema, e identificar en función de las mismas los mensajes que éste debe enviar al Centro de Mensajes.

Tras ello, será necesario actualizar los subsistemas existentes, para que se suscriban a los nuevos tipos de mensajes, si los hay, y añadir las modificaciones pertinentes en los mismos para que estos subsistemas puedan realizar el tratamiento de los nuevos mensajes.

Finalmente, deberán comprobarse nuevamente los escenarios de calidad existentes y verificar que la introducción del nuevo subsistema no ha provocado el incumplimiento de ninguno de ellos.

El procedimiento descrito se ha elaborado bajo la suposición de que la introducción del nuevo subsistema no implica descartar los atributos y escenarios de calidad manejados hasta el momento. En caso de que las necesidades hayan variado notablemente, será necesario someter a la arquitectura a un estudio más minucioso que excede los objetivos de estas recomendaciones metodológicas.

6.2.3. Realización de Modificaciones en el Sistema Inteligente de Tutoría

El proceso que debe realizarse para introducir modificaciones en el Sistema Inteligente de Tutoría es similar al que se sigue para introducir

cambios en el sistema. En este caso, sin embargo, deben tenerse en cuenta los servicios que prestan los agentes en lugar de los mensajes que envía cada uno de los subsistemas.

En primer lugar se debe especificar la funcionalidad del elemento que se desea introducir en el SIT, para así poder comprobar si parte o la totalidad de esa funcionalidad la presta alguno de los agentes presentes en el SIT.

En caso de que se desee sustituir un agente existente en el SIT, se deberá comprobar que el nuevo agente proporciona, al menos, toda la funcionalidad proporcionada por el agente que se va a eliminar, o que la funcionalidad que no se proporciona no será requerida por los agentes restantes.

A continuación se deberán revisar los escenarios de calidad existentes, para comprobar si son suficientes para la funcionalidad que se desea introducir. En caso contrario, deberán elaborarse los escenarios de calidad correspondientes a la funcionalidad que se va a introducir en el SIT.

Posteriormente debe decidirse si la funcionalidad se introduce en forma de un nuevo agente dentro del SIT, como un nuevo comportamiento de un agente existente o como parte de un comportamiento de alguno de los agentes del SIT. Además, si la nueva funcionalidad no se encuentra en el SIT como un agente o como parte integrante de un agente, se deberá proceder a *agentificarla* a través de la utilización de un *wrapper* (Hayden et al., 1999).

Tras ello, se deberán identificar los servicios que el nuevo agente o comportamiento necesita de los que hay presentes en el SIT. Si alguno de los servicios no lo prestase ninguno de los agentes, se deberá decidir si se incluirá en alguno de los agentes existentes, como parte de un nuevo agente o incluso si se puede prescindir de él.

De manera análoga, se deben definir los servicios que los agentes existentes necesitan del elemento nuevo, y se crearán los comportamientos necesarios para prestar los servicios demandados. También, de manera similar al paso anterior, si alguno de los servicios necesarios no se prestan, se deberá decidir si incluirlos en el nuevo agente, como parte de un agente existente o si se puede prescindir de ellos.

Finalmente, debe comprobarse que tras la introducción de la nueva funcionalidad en el SIT se siguen satisfaciendo los escenarios de calidad que afecten tanto al SIT como al sistema en su totalidad.

Al igual que sucedía en la sección anterior, este proceso es válido en casos en los que las modificaciones a realizar se ajusten a los criterios de diseño utilizados para diseñar esta arquitectura, los cuales se muestran en la siguiente sección, y siempre que la nueva funcionalidad no rompa con la funcionalidad ofrecida por el SIT.

6.3. Criterios para Realizar el Diseño de la Arquitectura

6.3.1. Diseño Top-Down

El diseño *top-down* es una herramienta de la que hacen uso numerosos métodos en ingeniería de software, ya que permite, aplicando una estrategia de tipo divide y vencerás, ir refinando el diseño de un sistema a través de su descomposición en elementos progresivamente más pequeños y sencillos. Es, además, un enfoque que cuadra muy bien con el diseño de sistemas con estructura jerárquica, ya que permite ir asignando responsabilidades a los sub-elementos a medida que se van descomponiendo los de nivel superior.

Sin embargo, durante la elaboración de las arquitecturas descritas en el presente trabajo se ha puesto de manifiesto que un enfoque de este estilo no nos ha conducido hacia los objetivos deseados, ya que en sistemas del tipo que nos ocupa presenta tres grandes inconvenientes.

El primero de ellos es que resulta necesario conocer *a priori* gran parte de los requisitos de la aplicación que se quiere construir, lo que no suele resultar habitual en sistemas cuya aplicación principal todavía se encuentra en el campo de la investigación. A pesar de que en métodos como ADD se afirma explícitamente lo contrario (Bass et al., 2003), es decir, que puede utilizarse manejando un mínimo de información, no deja de ser cierto que se necesita que la información de la que se dispone sea la información clave que le va a dar forma al sistema (lo que llaman *drivers* de la arquitectura).

El segundo inconveniente es que resulta relativamente sencillo aplicar este enfoque cuando se construye un sistema completo desde el principio, pero no lo es tanto cuando se pretende modificar un sistema existente, en el que hay que tener en cuenta no sólo las nuevas modificaciones, sino cómo fue diseñado el sistema inicialmente. Además, el diseño *top-down* indica que hay que descomponer un elemento en otros más simples, pero no dice cómo introducir nuevos elementos junto con los existentes, lo que se debería hacer, además, respetando los criterios que se utilizaron inicialmente. Si no se puede hacer, entonces se requiere la realización de un nuevo diseño desde el principio (Wirth, 1971).

El tercer inconveniente, y quizá el que puede presentar mayores dificultades, es que no se suele indicar cuáles son los criterios que se deben utilizar para realizar la descomposición. Estos criterios están muy relacionados con los atributos de calidad con los que se desee dotar al sistema, y por tanto juegan un papel fundamental en la forma que adoptará la arquitectura del sistema. En el caso de la modificabilidad, los autores de

(Bass et al., 2003) mencionan la utilidad de usar como criterio el *principio de ocultación de información* (Parnas, 1972) que, si bien puede emplearse como criterio de descomposición, también puede usarse, simplemente, como principio de diseño.

Existen otros problemas que se añaden a la utilización del enfoque *top-down* como estrategia de diseño. El primero es la tendencia a identificar los módulos de la descomposición con diferentes etapas de la ejecución del sistema, como ya señalaron en distintos momentos Parnas y Haythorn (Parnas, 1972; Haythorn, 1994). Si lo que deseamos es obtener modificabilidad, este criterio de diseño es bastante pobre, y el sistema obtenido con su uso resulta difícil de mantener.

El segundo problema es que puede resultar sencillo realizar una descomposición cuando el número de elementos resultantes se encuentra en un nivel manejable, pero no lo es tanto cuando el resultado de la descomposición es un número de elementos demasiado elevado. En este caso, resulta más factible trabajar con un número poco elevado de módulos e ir añadiendo los demás paulatinamente y de manera controlada.

Finalmente, el hecho de utilizar un enfoque *top-down* hace bastante probable que el diseño obtenido sea de tipo jerárquico, lo cual, como se verá con más detalle en el siguiente apartado, es una característica que no aporta grandes ventajas y sí algunos inconvenientes.

Por lo tanto, frente al diseño *top-down*, se propone la utilización de un enfoque diferente, que no se base en la descomposición, sino en la composición.

Hoy en día resulta completamente imposible diseñar un sistema que sea infinitamente modificable (Haythorn, 1994). Esto quiere decir que, si diseñamos una aplicación para que sea modificable, lo será en aquellos aspectos que se hayan diseñado para acomodar cambios con facilidad, pero habrá otros cambios que serán difícilmente realizables porque rompen del todo la filosofía del diseño o, simplemente, porque nunca se pensó el sistema para que esos cambios se pudiesen introducir, fuese o no de manera intencionada.

Como se dice tanto en (Haythorn, 1994) como en (Bass et al., 2003), la manera de saber si un sistema es modificable consiste en hacer una lista de posibles modificaciones para contrastar el diseño con los cambios propuestos. De esta manera, sabremos si el sistema admite o no los cambios listados y cuál es la implicación de dichos cambios.

Lo que se propone, por tanto, es comenzar desarrollando una parte pequeña y central del sistema, y contemplar como posibles cambios las características que sabemos que hay que añadir a la arquitectura. En la medida en que esas características vayan añadiéndose, se irá comprobando

que las demás modificaciones se pueden seguir realizando y se irán resolviendo las dificultades que puedan ir apareciendo. Esta manera de proceder es similar a lo que se propone en el desarrollo iterativo e incremental (Pressman, 2004; Sommerville, 2007).

Con esto, podremos estar razonablemente seguros de que se podrán incluir las características requeridas, pero no se asegura que puedan realizarse otras modificaciones.

Para ello, es necesario que la arquitectura se haya diseñado utilizando algún mecanismo lo suficientemente flexible como para incorporar cambios no contemplados a priori, aunque siempre teniendo en cuenta que será difícil que se pueda incorporar fácilmente cualquier tipo de cambio. En nuestro caso, como se verá más adelante, los mecanismos propuestos para ello son la utilización de servicios y el esquema editor-suscriptor.

6.3.2. Estructura Jerárquica

La arquitectura descrita en el capítulo 4 es el producto de utilizar un diseño con un enfoque *top-down* que ha dado como resultado una arquitectura con una estructura jerárquica, lo que constituía uno de los objetivos perseguidos. Al utilizar esta estructura jerárquica se pretendía dotar a la arquitectura de un alto grado de modificabilidad, aunque, como se explicó al final del capítulo 4, no se consiguió.

Nuestra experiencia ha sido que la estructura jerárquica, por sí sola, no proporciona mayor modificabilidad que otro tipo de estructuras, a pesar de que, en apariencia, la comunicación a través de los agentes que están en lo alto de la jerarquía debería proporcionar la independencia entre los agentes que se sitúan por debajo de ellos.

En un SIT al uso existe una dependencia bastante notable entre los distintos módulos, especialmente en lo que se refiere al Módulo de Tutoría. A medida que se descompone una estructura jerárquica se deben trasladar las responsabilidades del módulo superior a los módulos inferiores. Como consecuencia las dependencias entre módulos también descienden por la jerarquía, con el resultado de que los sub-módulos de una rama dependen de los de otra (ver Figura 6.4).

Por otro lado, la necesidad de utilizar varios intermediarios para comunicar a dos agentes también resulta en muchas ocasiones en la necesidad de modificar los intermediarios cuando se producen cambios en los agentes que inician y terminan la comunicación. Además, este tipo de comunicación resulta relativamente sencillo cuando se produce únicamente entre dos agentes, pero se complica notablemente cuando se ven implicados más de dos de ellos.

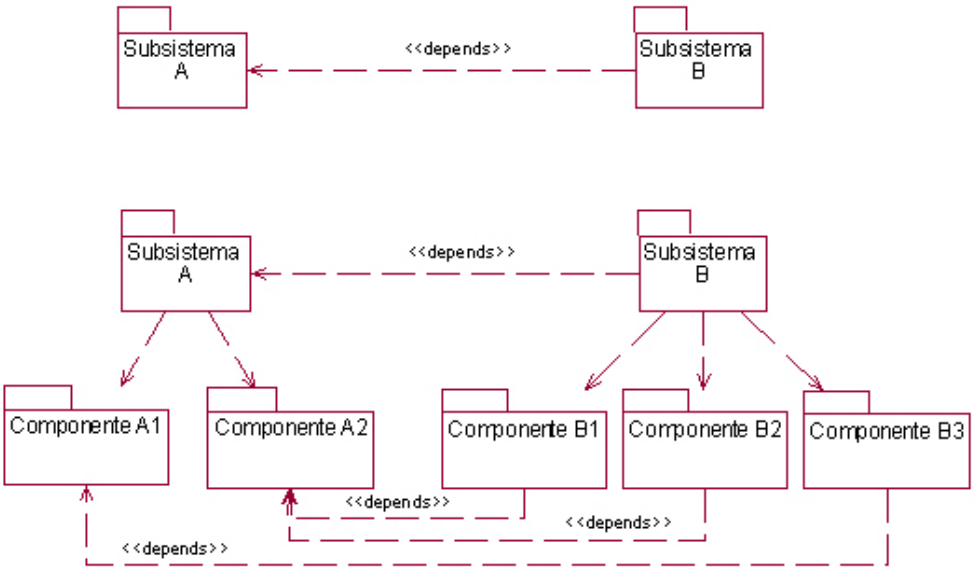

Figura 6.4: Propagación de dependencias en la descomposición

Como detalle adicional, la comunicación entre dos agentes que se encuentran bajo el mismo coordinador debería realizarse de manera distinta a como se realiza entre agentes que cuelgan de distintas ramas, para evitar que se envíen al coordinador mensajes que no van a salir de ese grupo de agentes (ver Figura 6.5).

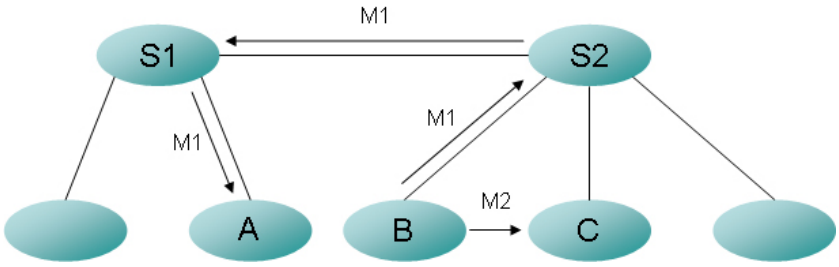

Figura 6.5: Envío de mensajes entre agentes

No hacerlo así supone cargar innecesariamente la comunicación, ya que se envían el doble de mensajes de los necesarios. Sin embargo, hacerlo así supone, por una parte, utilizar dos mecanismos distintos para una misma acción (la comunicación entre dos agentes), lo que complica el manejo de la arquitectura. Por otra parte, también supone la necesidad de saber si un agente tiene el mismo supervisor que otro para decidir cómo se realiza la

comunicación entre ellos, lo que, generalmente, disminuye la modificabilidad del sistema en dos sentidos: se necesita conocer directamente a un agente, lo que hace probable que un cambio en ese agente provoque cambios en los que lo conocen; y se necesita hacer cambios importantes en la forma de comunicación entre dos agentes si, por alguna causa, se hace necesario cambiar el agente del que cuelga otro agente dado, ya que haría falta cambiar la forma de comunicación de los agentes con los que habla, que deben enterarse de su nueva ubicación.

Esto último nos conduce además al hecho de que establecer una jerarquía se hace de acuerdo a uno o más criterios para decidir de qué agente debe colgar otro. Si los criterios con los que se realizó la clasificación cambian, lo más probable es que haya que realizar cambios grandes en toda la arquitectura. Además, en algunas ocasiones, puede suceder que los criterios utilizados para establecer la jerarquía no acomoden bien la necesidad de introducir un nuevo agente, lo que a su vez puede dar lugar a la necesidad de modificar los criterios con que se estableció la mencionada jerarquía.

Por todas las razones expuestas, se concluye la no idoneidad de utilizar una estructura jerárquica para construir un sistema como el que nos ocupa en el presente trabajo, al menos si lo que nos interesa es construir un sistema fácilmente modificable.

6.3.3. Organización Peer-To-Peer

Ahora que en el campo de los agentes parecen haber cobrado importancia los sistemas auto-organizados (Parunak y Brueckner, 2001) y los comportamientos emergentes (Li et al., 2006), una organización de tipo *peer-to-peer* se ajusta bastante bien para construir sistemas con características como las del que nos ocupa en este trabajo.

Sin embargo, no han sido estos dos aspectos los que nos han llevado a considerar la utilización de una organización *peer-to-peer*. Muy al contrario, un comportamiento emergente, incontrolado, no es una característica especialmente deseable en un sistema orientado al entrenamiento, salvo, quizá, precisamente para generar comportamientos incontrolados en situaciones en las que se requiera. En cualquier caso, es probable que esa situación no se deje en manos de un posible comportamiento emergente, sino que se intentará generar de una manera supervisada. En cuanto a la posibilidad de hacer uso de un sistema auto-organizado, no resulta especialmente atractivo en un entorno en el que el tipo y número de agentes es bastante reducido, por lo que la mencionada organización va a estar bastante dirigida.

Así, lo que nos ha llevado hacia una organización de tipo *peer-to-peer* no es sino la facilidad que proporciona para introducir nuevos tipos de agentes

en la medida en que se vayan añadiendo nuevas funcionalidades que requieran su inclusión. Esta facilidad viene dada, sobre todo, por la ausencia de unas reglas que puedan llegar a dificultar la introducción de nuevos agentes.

Esto no quiere decir que la arquitectura vaya a ser más fácilmente modificable sólo por el hecho de tener una estructura *peer-to-peer*; para ello habrá que complementarla con otros elementos. Lo que sí es cierto es que no va a suponer una dificultad adicional a la hora de modificar el sistema, lo que sí parece suceder, por contra, con sistemas de tipo jerárquico.

Por tanto, la propuesta realizada en el presente trabajo para conseguir un sistema que resulte fácilmente modificable se basa en la utilización de una organización de tipo *peer-to-peer*, complementada por otros mecanismos que se describen en el presente capítulo.

6.3.4. Edición-Suscripción

El esquema de edición-suscripción (*publish-subscribe*), como solución de diseño orientado a objetos, aparece publicada en (Gamma et al., 1993; Gamma et al., 1995), y posteriormente en (Buschmann et al., 1996). En ambos casos se identifica también como patrón observador. Como elemento de diseño con agentes, aparece referenciado de forma similar a la utilizada en este trabajo en (Silva et al., 2005).

Este mecanismo es descrito como una manera de desacoplar productores y consumidores de eventos o de información, de manera que los productores no necesitan conocer directamente a los consumidores de información, quienes se suscriben a la recepción del material que les interesa, y el editor mantiene una lista de suscriptores a los que notificar cualquier cambio que se produzca.

En la descripción que se hace en ambos trabajos, el patrón aparece muy relacionado con el modo de funcionamiento de las interfaces gráficas de usuario, donde un mismo evento puede ser notificado a varios elementos, y donde un cambio puede mostrarse por varias salidas. A este comportamiento es al que en (Clements et al., 2002a) se denomina *invocación implícita*. Esta invocación implícita consiste en que los consumidores de información se suscriben a la recepción de determinados eventos a través de la implementación de unos métodos concretos, mientras que los productores anuncian los eventos escribiéndolos en el lugar que la infraestructura de comunicación determine. Una vez escritos, la infraestructura de comunicación los reparte entre los suscriptores en un orden preestablecido, que puede ser el orden de suscripción, mediante la invocación de los métodos anteriormente mencionados.

No resulta difícil darse cuenta de que este esquema desacopla al productor de información o eventos de los suscriptores, pero no sucede lo mismo al

contrario, ya que los suscriptores sí deben conocer al editor a cuyos eventos se quieren suscribir. Esto resulta aún más evidente si se considera desde el punto de vista del patrón observador, en el que los observadores conocen la identidad del objeto observado.

Al contrario de lo que sucede con los objetos, la utilización de agentes nos facilita la introducción de una pequeña variante, que es la que se propone utilizar, para conseguir también desacoplar a los consumidores de información del productor.

Esto se consigue a través de la introducción de un mecanismo de indirección, las páginas amarillas, que permiten a los productores anunciar los tipos de eventos o información que pueden producir, en lugar de anunciar sólo los eventos.

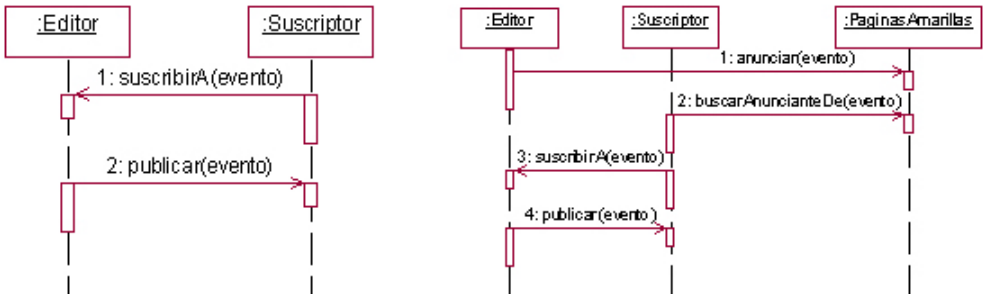

Figura 6.6: Variación del patrón Editor-Suscriptor. A la izquierda, el esquema original, y a la derecha, el modificado con el uso de las páginas amarillas

De este modo, los consumidores tienen un conocimiento de la información que les interesa, algo que ya sucedía antes, si bien con las páginas amarillas se consigue que no necesiten un conocimiento directo y explícito del productor de esa información. Ahora pueden preguntar quién produce lo que les hace falta, evitando conocer directamente a los productores. Lo único que llega a saber un agente es que se comunica con *el agente que proporciona X*, que puede ser cualquiera de los agentes que se encuentran activos en ese momento.

Empleando este mecanismo se consigue, sobre el papel, que los editores y los suscriptores actúen de manera completamente independiente de para quién publiquen o de quién consuman. Si, como diseñadores, somos capaces de seguir este esquema sin hacer suposiciones acerca de lo que debe hacer uno u otro tipo de agente, se habrá debilitado enormemente la relación existente entre productores y consumidores. Esto, a su vez, introduce en el diseño un grado de ambigüedad bastante elevado, ya que se ha pasado de tener una relación uno a uno del habitual esquema productor-consumidor, a la

relación uno a muchos del esquema de invocación implícita, para terminar, finalmente, en una relación de muchos a muchos del esquema editor-suscriptor con páginas amarillas.

De esta manera, siempre que se mantengan los servicios proporcionados por un agente, se podrá sustituir éste por uno o varios agentes diferentes que presten los mismos servicios sin que los consumidores se vean afectados, lo cual amplía el grado de modificabilidad proporcionado por el patrón editor-suscriptor tradicional.

Existen tres inconvenientes que se le achacan habitualmente al uso de una solución de este estilo (Gamma et al., 1995). El primero de ellos radica en que los productores de información pueden estar un tiempo determinado intentando publicar un evento, tras el cual desisten para poder continuar la ejecución. Si no han conseguido publicar el evento, entonces éste se pierde, con la consiguiente pérdida de información y un posible funcionamiento anómalo del sistema. Este problema no se produce en un sistema basado en agentes por el hecho de utilizar para la comunicación un esquema basado en el paso de mensajes asíncrono, lo cual hace que se puedan enviar los mensajes aunque el receptor no se encuentre disponible en el momento del envío.

El segundo inconveniente reside en el hecho de que, al desacoplar productores y consumidores, a veces utilizando algún tipo de estructura intermedia, no se pueden hacer cierto tipo de asunciones u operaciones, como comprobar si un mensaje ha llegado a un destinatario concreto. De esta manera, no se puede tomar ninguna medida ante, por ejemplo, un mal funcionamiento del sistema si el receptor fallase. Este problema también puede solucionarse gracias a la comunicación a través de paso de mensajes, especialmente si los mensajes se estructuran conforme al estándar FIPA-ACL. De acuerdo con este estándar, uno de los campos del mensaje incluye

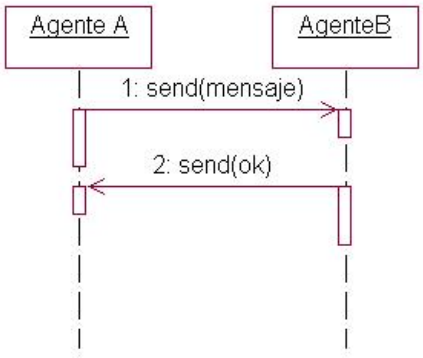

Figura 6.7: Confirmación de recepción de mensajes

la identidad del emisor. En caso de querer implementar algún mecanismo de confirmación de recepción de los mensajes, basta con diseñar un protocolo de comunicación que requiera dicha confirmación. Para ello, en el caso más sencillo, el receptor sólo debería responder al emisor del mensaje con una confirmación de recepción del mismo (ver Figura 6.7).

El tercer inconveniente que se le atribuye a esta solución es que no escala correctamente cuando el número de suscriptores es muy elevado, produciéndose un descenso notable del rendimiento de la aplicación. Sin embargo, en el estado actual de este tipo de sistemas, se estima que el número de usuarios simultáneos no será tan elevado como para llegar al punto en que se produzca este problema. Si aun así se produjese esta situación, existen soluciones alternativas para resolverla, como la descrita en (Neo et al., 2005).

6.3.5. Relaciones de Uso

A lo largo del presente trabajo se ha mencionado repetidamente que uno de los principales objetivos que se persiguen es el diseño de una arquitectura fácilmente modificable para construir Entornos Virtuales de Entrenamiento.

Un indicador utilizado para medir la modificabilidad de un diseño es el número de relaciones existentes entre los distintos módulos del mismo (Bass et al., 2003). Sin embargo, en este trabajo no se ha hecho uso de él por varias razones.

En primer lugar, aunque un diseño en el que existan muchas relaciones entre módulos puede ser indicativo de un acoplamiento alto y, por tanto, de una modificabilidad compleja, lo contrario no tiene por qué ser cierto, es decir, pocas relaciones entre módulos no son necesariamente una indicación de un diseño modificable. Una relación entre dos módulos puede comprometer enormemente un diseño si esa relación no se realiza de la manera adecuada. Esto sucede si, por ejemplo, a través de una única llamada a una función se obtienen datos que deberían ser conocidos únicamente por quien los ha proporcionado. Además, la existencia de pocas relaciones entre módulos también puede ser un síntoma de un criterio pobre a la hora de decidir cómo realizar una división del sistema en módulos. Así, un sistema monolítico que constase de un único módulo sería el ideal de sistema modificable, cuando en general esto nunca es así.

En segundo lugar, una relación de un módulo con otro puede esconder una relación, directa o indirecta, con un tercero, y esa relación también afecta a la modificabilidad del sistema (Clements et al., 2002a). Es lo que sucedía, por ejemplo, en la arquitectura descrita en el capítulo 4, en la que la comunicación a través de los agentes coordinadores, que en apariencia independizaba a los

agentes entre sí, en el fondo ocultaba una dependencia bastante directa de los agentes entre los que verdaderamente tenía lugar la comunicación.

Por esa razón, se estima que, de cara a llevar a cabo un estudio de la modificabilidad del sistema, es interesante disponer de la documentación de una vista de uso que muestre las relaciones de uso entre los distintos módulos del sistema.

Tal y como describen los autores de (Clements et al., 2002a), las relaciones de uso se dan no sólo a través de llamadas directas de un módulo a otro, sino también, por ejemplo, a través de información que se comparte por medio de un elemento intermedio, del estado en que se deja a un módulo de uso común o mediante una llamada a un tercer módulo que se realiza para satisfacer una petición.

Esta relación dará una visión algo más certera de las dependencias entre módulos que la mera existencia de llamadas directas entre ellos.

Por lo tanto, en el proceso de diseño de la arquitectura, interesa realizar un estudio minucioso de las relaciones de uso entre módulos, que permitan identificar las dependencias reales existentes. Llegado el momento, se podrá trabajar sobre esas relaciones de uso para, en la medida de lo posible, debilitarlas o eliminarlas para disminuir el impacto de un cambio en un módulo sobre los módulos que lo usan.

6.3.6. Sistemas Abiertos y Cerrados

Una de las posibles clasificaciones que se establecen para diferenciar los distintos tipos de sistemas basados en agentes es la que los separa en los llamados sistemas abiertos y cerrados (Moya y Tolk, 2007).

Se llama sistemas cerrados a aquellos que sólo albergan a los agentes que han sido desarrollados por el creador del sistema para trabajar dentro de él. Por lo tanto, es posible tener bajo control qué agentes se ejecutan dentro del sistema en todo momento y qué tipo de acciones realizan.

En oposición a estos se encuentran los llamados sistemas abiertos, que ofrecen la posibilidad de que se ejecuten agentes desarrollados por terceros procedentes de fuentes diversas. Estos sistemas plantean retos complejos en cuanto a seguridad, interoperatividad o negociación entre agentes.

Sin embargo, pese a las dificultades, los sistemas abiertos presentan una característica bastante interesante a la hora de diseñar el sistema, que es el hecho de que, al poder admitir agentes que no han sido diseñados por el propietario de la plataforma, obligan de alguna manera a no hacer suposiciones acerca de muchas de sus características.

Por supuesto, se debe suponer que todos los agentes conocen los mecanismos de comunicación con los demás agentes, pero más allá de eso, no

se puede siquiera suponer que vayan a respetar unas determinadas normas. Esto ha obligado a desarrollar un campo de investigación relacionado con la seguridad en sistemas abiertos (He y Sycara, 1998; Jiang et al., 2005; Poslad y Calisti, 2000).

Dejando aparte los problemas de seguridad, al diseñar uno de estos sistemas nos vemos obligados a trabajar con relaciones mucho más ambiguas entre los agentes, ya que no podemos estar seguros de qué agentes vamos a encontrar ni si alguno de ellos va a poder proporcionarnos lo que necesitemos. Esto, a su vez, nos obliga a diseñar el sistema de forma tal que haya que buscar qué agentes existen, qué servicios nos proporcionan y qué se hace en caso de no encontrar quien proporcione lo que se necesita, lo cual hace que las dependencias entre agentes sean muy débiles.

Teniendo esto en mente, a la hora de diseñar un sistema del tipo del que nos ocupa debemos tener en cuenta que una aplicación de entrenamiento, especialmente en dominios como el industrial, es bastante sensible a incursiones externas, por lo que es poco probable que exista la necesidad (o el interés) de acoger agentes venidos del exterior. Esto quiere decir que en la mayor parte de los casos estas aplicaciones serán sistemas cerrados en los que no se dará la posibilidad de que entren agentes externos al sistema.

Aún así, podemos diseñar nuestra aplicación como si de un sistema abierto se tratase, aprovechando las técnicas utilizadas para obtener relaciones ambiguas entre agentes y dependencias débiles, al tiempo que tenemos controlados en todo momento a los agentes que se ejecutan en el sistema. De esta manera, siempre sabremos que van a estar presentes los agentes adecuados prestando todos los servicios que se requieran.

Cabe preguntarse entonces por qué tendríamos que tomarnos la molestia de diseñar sistemas abiertos si siempre va a estar todo bajo control. El principal motivo es que nos facilitará el trabajo a la hora de realizar modificaciones, ya que podremos sustituir un agente por otro sin más que mantener los servicios que prestaba el agente que se ha eliminado. Si además, por alguna razón, el nuevo agente no presta alguno de los servicios que prestaba el original, siempre que no sea un servicio fundamental para otro agente, éste podrá realizar acciones alternativas que se hayan previsto en ausencia de ese servicio. En el fondo, sustituir un agente por otro puede considerarse como dejar entrar a un agente desde el exterior, pero siempre bajo el control del desarrollador. Esto elimina la necesidad de tener que implementar los mecanismos de seguridad en el sistema.

A la vista de lo expuesto, el diseño de la arquitectura del sistema como si de un sistema abierto se tratase proporcionará características que ayudarán a hacerlo más modificable. Sin embargo, habrá que tener en cuenta que no será necesario añadir determinadas funcionalidades que no aportan nada pero

que, por contra, penalizarán el sistema añadiéndole complejidad innecesaria y, con toda probabilidad, reducirán la velocidad de ejecución.

6.4. Reflexiones

El análisis de los sistemas que aparecen en el capítulo de estado de la cuestión, así como el contacto que se ha tenido con otros sistemas basados en agentes, ha puesto de manifiesto que muchos de estos sistemas se desarrollan de manera artesanal y sin aplicar principios de diseño adecuados, tanto en lo relativo a la arquitectura del sistema como al posterior diseño de bajo nivel.

Adicionalmente, la experiencia adquirida a través de la redacción de artículos y la participación en congresos da la oportunidad de conocer la opinión tanto de los revisores de los artículos como de los participantes en los congresos. La sensación que dejan algunas de estas experiencias es que el diseño en general, y el arquitectónico en particular, no recibe la suficiente atención por parte de la comunidad científica.

Desde el área de las arquitecturas software, esto se achaca al hecho de que rara vez se desarrollan sistemas que lleguen a un estado de producción, sino que simplemente se construye lo necesario para validar los resultados de una investigación, por lo que los sistemas desarrollados rara vez pasan de ser prototipos. Aunque esta afirmación no refleja la opinión del autor, al menos sí se está de acuerdo en que el diseño arquitectónico aún no recibe la atención necesaria.

En muchos casos, además, se da por hecho que ciertas prácticas son de uso común y que por lo tanto no merece la pena dedicarles ninguna mención, como puede ser el hecho de utilizar técnicas que mejoren la modificabilidad de una aplicación. Por contra, lo que esto suele enmascarar es la ausencia de estas técnicas, por lo que se da por sentado que las aplicaciones poseen características de las que en realidad carecen.

Aunque los criterios de diseño proporcionados son en cierta manera algo genéricos, constituyen la base que ha permitido dotar a la arquitectura presentada de los atributos de calidad deseados.

Además, y sobre todo, se ha justificado de manera razonada por qué la utilización o no de unos y otros beneficia el diseño de la arquitectura propuesta, lo cual los convierte en herramientas útiles para desarrollar estos sistemas con la garantía, al menos, de que los atributos de calidad han sido considerados, y no simplemente dados por supuesto.

Parte III

Resultados y Aportaciones

Capítulo 7

Evaluación de la Arquitectura

"Con una evaluación cuidadosa, uno puede vencer; sin ella, no puede"
— Sun Tzu

7.1. Introducción

Una vez diseñada la arquitectura, la evaluación de la misma permitirá verificar que cumple con los objetivos planteados al comienzo de este trabajo. Como ya se mencionó en ese momento, existen dos aproximaciones para llevar a cabo esta evaluación.

La primera consiste en realizar medidas cuantitativas relacionadas con los distintos atributos de calidad de la arquitectura. La ventaja de utilizar esta técnica la constituye el hecho de que el resultado estará formado por una serie de medidas objetivas acerca de la calidad de la solución propuesta. La desventaja, como sucede con muchas medidas numéricas, es que la utilización de medidas numéricas para evaluar la arquitectura elimina mucha información que puede resultar relevante para conocer las características de la misma.

La segunda manera de realizar la evaluación consiste en llevar a cabo un análisis cualitativo de la arquitectura, de manera que el resultado no es una medida de la calidad de la solución, sino un informe que recoge las características de la arquitectura, los posibles riesgos y puntos de conflicto y una estimación acerca del cumplimiento de los objetivos de la misma.

Actualmente, están más desarrollados y son más utilizados los métodos de evaluación del segundo tipo (Clements et al., 2002b; Bass et al., 2003). Esto, unido a que proporcionan una información más rica sobre la arquitectura, ha

motivado que nos decantemos por este tipo de evaluación para la arquitectura propuesta en el capítulo anterior.

A continuación se ofrece una breve visión de los métodos que se han barajado para realizar la evaluación y se presentan los resultados obtenidos. Por último, se incluye una serie de reflexiones sobre el proceso de evaluación.

7.2. Selección del Método de Evaluación

Los métodos que se consideraron inicialmente para evaluar la arquitectura fueron ARID (*Active Reviews for Intermediate Designs*) (Clements, 2000), SAAM (*Software Architecture Analysis Method*) (Kazman et al., 1994) y ATAM (*Architecture Tradeoff Analysis Method*) (Kazman et al., 2000), todos ellos desarrollados por el grupo de arquitecturas software del SEI (*Software Engineering Institute - Carnegie Mellon University*).

Tras un primer contacto con los tres, se descartó el primero de ellos, ARID, porque su utilidad se encuentra en la evaluación de arquitecturas sin completar o trozos de arquitecturas, lo cual no se ajusta a nuestras necesidades de evaluación.

De entre los otros dos métodos, se ha preferido ATAM principalmente por dos motivos. El primero de ellos es que SAAM se centra sobre todo en la evaluación de la modificabilidad de la arquitectura. Aunque la modificabilidad ha constituido el principal atributo de calidad que ha dirigido el diseño de la arquitectura, no ha sido el único, pues también se ha tenido en cuenta el rendimiento. ATAM da la posibilidad de que surjan cuestiones relativas a atributos de calidad que no se habían considerado inicialmente.

En segundo lugar, ATAM es un método de evaluación más moderno que SAAM, al que engloba, por lo que todo lo que pueda evaluarse con SAAM debe poder evaluarse también con ATAM.

Sin embargo, no todo son ventajas en el uso de ATAM. El principal inconveniente es que implica un proceso de evaluación más complejo y difícil de realizar que SAAM, aunque dadas las características de las personas que han participado en la evaluación, se ha podido simplificar bastante.

Por otro lado, los resultados de una evaluación con ATAM dependen en gran medida de quién participe en la evaluación. Dadas las características de esta investigación, que forma parte de un proyecto que no se desarrolla para ningún cliente, y en el que el equipo de desarrollo es de pequeño tamaño y mantiene una comunicación bastante fluida, muchas de las personas que se han visto involucradas en la evaluación han formado parte del equipo de investigación, por lo que los intereses que han salido a la luz son muy similares a los que han dirigido la construcción del sistema. Esto no significa que la

evaluación carezca validez, sino que los intereses que movían el diseño de la arquitectura no han variado durante el tiempo que ha durado el mismo.

Sin embargo, también han participado en la evaluación de la arquitectura algunos evaluadores externos y miembros de otro equipo de investigación interesado en utilizar la aplicación, los cuales han aportado una visión ligeramente distinta y, por tanto, han puesto de manifiesto algunas cuestiones que no se habían considerado.

7.3. Evaluación de la Arquitectura

ATAM es un método para evaluar arquitecturas software que consta de 9 pasos agrupados en 4 fases. A continuación se describe brevemente en qué consiste cada uno de ellos, si bien puede obtenerse una explicación mucho más detallada en (Clements et al., 2002b).

Fase 0: Presentación. Explicación del proceso de evaluación ante todos los participantes

1. Presentar ATAM a los participantes en la evaluación.
2. Presentar la motivación para desarrollar el sistema.
3. Presentar la arquitectura.

Fase 1: Investigación y Análisis. Identificación de los atributos de calidad relevantes y soluciones arquitectónicas con el equipo de desarrollo

4. Identificación de los principales mecanismos arquitectónicos utilizados en el diseño.
5. Generación del árbol de pesos de los atributos de calidad.
6. Análisis de las soluciones arquitectónicas en función de los atributos de calidad con mayor peso.

Fase 2: Pruebas. Comparación de los resultados con las necesidades de todos los interesados

7. Identificación y priorización de nuevos escenarios.
8. Análisis de las soluciones arquitectónicas en función de los nuevos escenarios con mayor peso.

Fase 3: Informes. Presentación de los resultados de ATAM

9. Generación de un informe con los resultados de la evaluación.

En las siguientes secciones de este capítulo se presentan los resultados de la evaluación mediante el informe generado en el paso 9. Éste contiene el árbol de pesos de los atributos de calidad (paso 5), los escenarios de calidad correspondientes a las hojas del árbol de pesos y los identificados en el paso 7, y el análisis realizado para cada uno de los escenarios (pasos 6 y 8).

7.3.1. Árbol de Pesos

La elaboración del árbol de pesos es uno de los pasos fundamentales para la realización de la evaluación, ya que permite, por una parte, identificar los aspectos de la arquitectura que deben ser evaluados, y por otra, determinar su relevancia y su nivel de dificultad, de manera que se pueda prestar mayor atención a aquellas características más importantes y complejas.

También da lugar a la elaboración de una parte de los escenarios que se utilizarán más adelante.

El árbol de pesos obtenido para el tipo de sistemas que nos ocupa es el que se puede ver en la Figura 7.1. Cada hoja del árbol representa una característica de calidad que se considera relevante, asignándole a cada una de ellas dos puntuaciones. La primera de corresponde a la importancia que se le concede a esa característica, y la segunda a la dificultad que entraña llevarla a cabo. Cada puntuación puede tomar tres valores: (A)lta , (M)edia y (B)aja.

En una sesión típica de ATAM sólo se lleva a cabo la evaluación de aquellas hojas del árbol que no reciben un valor bajo en ninguna de las dos valoraciones. Esto es debido a que el tiempo para realizar la evaluación suele ser limitado. Por ello, si una característica presenta dificultad baja, no tiene sentido dedicarle un tiempo que se le puede dedicar a otras. Si, por el contrario, el interés en la característica es bajo, tampoco merece la pena perder el tiempo con ella, especialmente si presenta una dificultad alta. En nuestro caso, sin embargo, se llevará a cabo la evaluación de todas las características.

7.3.2. Escenarios para la Evaluación

Los escenarios describen posibles interacciones con el sistema, ya sean mediante su uso o mediante la modificación de alguna de sus características. Su importancia radica en que describen de manera precisa una cualidad a la que el sistema debe dar respuesta, lo cual permite que la evaluación se realice sobre aspectos concretos del sistema. Gracias a ello, aunque la evaluación se realice de forma cualitativa, se pueden obtener resultados concretos relativos a los distintos atributos de calidad que resulten de interés.

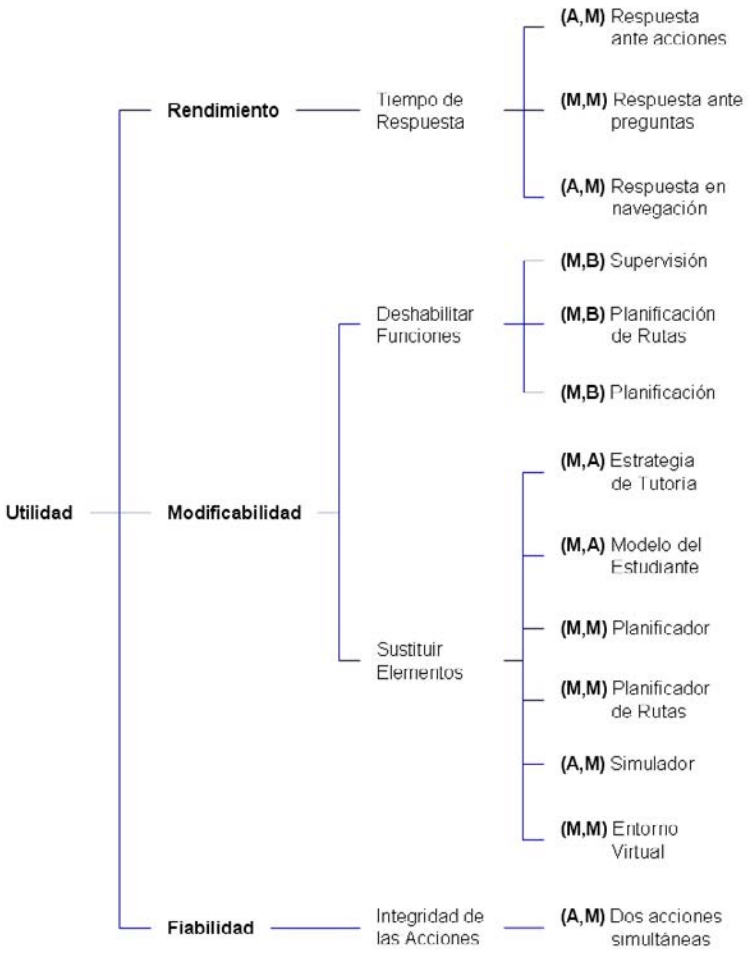

Figura 7.1: Árbol de características de ATAM

Un escenario típico consiste en una descripción corta de una interacción con el sistema, en la que se incluyen: el *estímulo* que inicia el escenario; el *entorno* que rodea a la interacción; y la *respuesta* del sistema al estímulo. Como se puede ver, la estructura es similar a la de los escenarios de calidad utilizados para diseñar la arquitectura, aunque con algo menos de detalle.

Los escenarios se clasifican en tres tipos, según su intención:

Escenarios de Uso. Estos escenarios describen la manera en que el sistema en ejecución debe responder a las acciones de un usuario. A este tipo de escenarios pertenecen los que expresan propiedades dinámicas de los atributos de calidad.

Escenarios de Evolución. Estos escenarios representan cambios anticipados en el sistema, ya sean relativos a su funcionalidad o modificaciones de los atributos de calidad (por ejemplo, necesidad de mayor rendimiento o más seguridad).

Escenarios de Exploración. Estos escenarios tienen como objetivo probar los límites del sistema. En el momento de su redacción pueden representar situaciones poco probables, pero que pueden llegar a hacerse realidad y ayudan a identificar limitaciones del sistema.

A continuación se describen los escenarios de cada tipo que se han elaborado, correspondientes tanto al árbol de pesos mostrado anteriormente como a los obtenidos en otras reuniones.

A cada escenario se le ha asignado una identificación que consta de un número de orden precedido de la inicial del atributo de calidad al que afecta: (M)odificabilidad, (E)ficiencia o (F)iabilidad.

7.3.2.1. Escenarios de Uso

E01 El usuario realiza una acción dentro del EV y el SIT devuelve una respuesta en un tiempo no superior a un segundo.

E02 El usuario realiza una pregunta durante el entrenamiento y el SIT envía una respuesta a la misma en un tiempo no superior a tres segundos.

E03 El usuario realiza movimientos en el EV y la imagen se actualiza con una tasa de refresco no inferior a 15 fps (fotogramas por segundo).

F01 Dos usuarios realizan dos acciones simultáneas y el ITS las procesa sin que se produzcan errores.

M01 El usuario desea realizar una actividad sin la supervisión del tutor, y ésta se desactiva al momento sin necesidad de reiniciar el sistema.

M02 No se necesita supervisar los desplazamientos del usuario y la planificación de rutas se desactiva al momento sin necesidad de reiniciar el sistema.

M03 No se necesita comprobar la corrección de las acciones del estudiante, por lo que la planificación se desactiva al instante sin necesidad de reiniciar el sistema.

7.3.2.2. Escenarios de Evolución

M04 Se desea sustituir la estrategia de tutoría por otra de características similares con un esfuerzo inferior a seis meses·persona.

M05 Se desea sustituir el módulo que realiza el modelado del estudiante por otro que preste una funcionalidad similar con un esfuerzo inferior a seis meses·persona.

M06 Se desea sustituir el planificador por otro de características similares con un esfuerzo inferior a seis meses·persona.

M07 Se desea sustituir el planificador de rutas por otro de similares prestaciones con un esfuerzo inferior a seis meses·persona.

M08 Se desea sustituir el simulador por otro diferente con un esfuerzo inferior a seis meses·persona.

M09 Se desea sustituir el Entorno Virtual por otro distinto con un esfuerzo inferior a seis meses·persona.

M10 Se desean introducir dispositivos *hápticos* para la interacción del usuario con el EV con un esfuerzo inferior a seis meses·persona.

M11 Se desea dotar al SIT de un mecanismo de percepción de los sucesos del EV que emule a la percepción humana con un esfuerzo inferior a seis meses·persona.

M12 Se desea introducir en el EV una representación física del tutor (Tutor Virtual) que acompañe al alumno durante el entrenamiento con un esfuerzo inferior a seis meses·persona.

M13 Se desea dotar al SIT de una arquitectura cognitiva que le permita manejar rasgos de personalidad y emociones que emulen a los humanos con un esfuerzo inferior a un año·persona.

M14 Se desea introducir un mecanismo de tutoría que funcione con Razonamiento Basado en Casos (CBR) con un esfuerzo inferior a un año·persona.

M15 Se desea sustituir las explicaciones *enlatadas* del tutor por un mecanismo de generación de lenguaje natural con un esfuerzo inferior a seis meses·persona.

M16 Se desea incluir un mecanismo de entrenamiento de grupos que incorpore miembros manejados por el ordenador con un esfuerzo inferior a dos años·persona.

7.3.2.3. Escenarios de Exploración

M17 Se desea transformar el sistema en un museo virtual en el que se puedan visitar las instalaciones usadas para el entrenamiento y que disponga de un guía virtual.

M18 Se desea transformar el sistema en un entorno que sirva para contar historias interactivas.

M19 Se quiere que el sistema pueda funcionar en dispositivos móviles.

M20 Se desea que el sistema dé soporte al entrenamiento de grupos de más de 100 usuarios.

M21 Se desea que el sistema soporte varias sesiones de entrenamiento en diferentes actividades de forma simultánea.

M22 Se desea que el SIT permita la entrada de agentes externos que realicen funciones análogas a los existentes.

M23 Se quiere que el sistema pueda funcionar mediante el uso de interfaces olfativas.

7.3.3. Análisis de la Arquitectura

Para realizar el análisis de la arquitectura, cada escenario se muestra en una tabla que sigue el formato sugerido en (Clements et al., 2002b). En ella se muestran los atributos de calidad afectados por el escenario, se identifican, donde resulta posible, el estímulo que inicia el escenario y la respuesta esperada y, finalmente, las decisiones de diseño que posibilitan el cumplimiento del escenario, junto con sus implicaciones:

- Puntos de riesgo: posibles problemas al tomar esa decisión de diseño.
- Puntos sin riesgo: propiedades que garantizan los atributos de calidad pero que podrían cambiar si se modifica la decisión de diseño.
- Puntos sensibles: propiedades críticas para conseguir que se cumplan los atributos de calidad.

- Puntos de compromiso: Puntos en los que la mejora de un atributo de calidad empeora otros.

A continuación de las tablas que describen los escenarios se detallan las implicaciones de cada uno de ellos.

Escenario: E01	El usuario realiza una acción en el EV y el SIT devuelve una respuesta en un tiempo no superior a un segundo			
Atributo	Rendimiento			
Estímulo	El usuario realiza una acción			
Respuesta	Se muestra el efecto de la acción en menos de un segundo			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Preparación anticipada de la siguiente acción	R01,R02		S01,S02	

Escenario: E02	El usuario realiza una pregunta durante el entrenamiento y el SIT envía una respuesta a la misma en un tiempo no superior a tres segundos			
Atributo	Rendimiento			
Estímulo	El usuario realiza una pregunta			
Respuesta	La respuesta a la pregunta se recibe en menos de tres segundos			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Respuestas prediseñadas	R03	N01	S03	C01

Escenario: E03	El usuario realiza movimientos en el EV y la imagen se actualiza con una tasa de refresco no inferior a quince frames por segundo			
Atributo	Rendimiento			
Estímulo	El usuario se mueve por el EV			
Respuesta	La tasa de refresco de la escena es de al menos quince fps			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Ejecutar acciones en función de la tasa de refresco			S04	

Escenario: F01	Dos usuarios realizan dos acciones simultáneas y el ITS las procesa sin que se produzcan errores			
Atributo	Fiabilidad			
Estímulo	Se producen dos acciones simultáneas			
Respuesta	Se procesan sin errores ni inconsistencias			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Procesamiento secuencial de las acciones	R04		S05	C02

Escenario: M01	El usuario desea realizar una actividad sin la supervisión del tutor, y ésta se desactiva al momento sin necesidad de reiniciar el sistema			
Atributo	Modificabilidad			
Estímulo	Se desactiva la supervisión del SIT			
Respuesta	Tiene efecto al momento, sin reiniciar el sistema			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Prestar servicios desde comportamientos		N02		

Escenario: M02	No se necesita supervisar los desplazamientos del usuario y la planificación de rutas se desactiva al momento sin necesidad de reiniciar el sistema			
Atributo	Modificabilidad			
Estímulo	Se desactiva la planificación de rutas			
Respuesta	Se hace al momento y sin reiniciar el sistema			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Prestar servicios desde comportamientos		N02		

Escenario: M03	No se necesita comprobar la corrección de las acciones del estudiante, por lo que la planificación se desactiva al instante sin necesidad de reiniciar el sistema			
Atributo	Modificabilidad			
Estímulo	Se desactiva la planificación			
Respuesta	Se realiza al momento y sin reiniciar el sistema			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Prestar servicios desde comportamientos		N02		

Escenario: M04	Se desea sustituir la estrategia de tutoría por otra de características similares con un esfuerzo inferior a seis meses·persona			
Atributo	Modificabilidad			
Estímulo	Sustitución de la estrategia de tutoría			
Respuesta	Se realiza con un esfuerzo inferior a tres meses·hombre			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Estrategia de tutoría dentro de un agente	R05	N03		
Separación del modelo del estudiante	R06		S06	
Solicitud de servicios	R07	N04		C03

Escenario: M05	Se desea sustituir el módulo que realiza el modelado del estudiante por otro que preste una funcionalidad similar con un esfuerzo inferior a seis meses·persona			
Atributo	Modificabilidad			
Estímulo	Sustitución del mecanismo de modelado del estudiante			
Respuesta	Se realiza con un esfuerzo inferior a tres meses·hombre			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Modelo del estudiante dentro de un agente	R05	N03		
Separación de la estrategia de tutoría	R06		S07	
Solicitud de servicios	R07	N04		C03

Escenario: M06	Se desea sustituir el planificador por otro de características similares con un esfuerzo inferior a seis meses·persona			
Atributo	Modificabilidad			
Estímulo	Sustitución del planificador			
Respuesta	Se realiza con un esfuerzo inferior a seis meses·hombre			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Planificador dentro de un agente	R05,R08	N03	S08	C04
Solicitud de servicios	R07	N04		C03

Escenario: M07	Se desea sustituir el planificador de rutas por otro de similares prestaciones con un esfuerzo inferior a seis meses·persona			
Atributo	Modificabilidad			
Estímulo	Sustitución del planificador de rutas			
Respuesta	Se realiza con un esfuerzo inferior a seis meses·hombre			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Planificador de rutas dentro de un agente	R05,R09	N03	S09	C04
Solicitud de servicios	R07	N04		C03

Escenario: M08	Se desea sustituir el simulador por otro diferente con un esfuerzo inferior a seis meses·persona			
Atributo	Modificabilidad			
Estímulo	Sustitución del simulador			
Respuesta	Se realiza con un esfuerzo inferior a seis meses·hombre			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Simulador detrás de un agente	R05,R10	N03		C04
Solicitud de servicios	R07	N04		C03

Escenario: M09	Se desea sustituir el Entorno Virtual por otro distinto con un esfuerzo inferior a seis meses·persona			
Atributo	Modificabilidad			
Estímulo	Sustitución del EV			
Respuesta	Se realiza con un esfuerzo inferior a seis meses·hombre			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Comunicación a través del Centro de Mensajes		N05	S10	

Escenario: M10	Se desean introducir dispositivos hápticos para la interacción del usuario con el EV con un esfuerzo inferior a seis meses·persona			
Atributo	Modificabilidad			
Estímulo	Añadir dispositivos hápticos al EV			
Respuesta	Se realiza con un esfuerzo inferior a un año·hombre			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Planificación de Eventos	R11		S11	

Escenario: M11	Se desea dotar al SIT de un mecanismo de percepción de los sucesos del EV que emule a la percepción humana con un esfuerzo inferior a seis meses·persona			
Atributo	Modificabilidad			
Estímulo	Añadir un mecanismo de percepción			
Respuesta	Se realiza con un esfuerzo inferior a tres seis·hombre			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Introducir un agente de percepción		N06	S12	
Crear un modelo específico del mundo	R12			C05

Escenario: M12	Se desea introducir en el EV una representación física del tutor que acompañe al alumno durante el entrenamiento con un esfuerzo inferior a seis meses·persona			
Atributo	Modificabilidad			
Estímulo	Añadir un tutor virtual			
Respuesta	Se realiza con un esfuerzo inferior a seis meses·hombre			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Introducir un agente para el tutor virtual		N06	S13	

Escenario: M13	Se desea dotar al SIT de una arquitectura cognitiva para manejar rasgos de personalidad y emociones que emulen a los humanos con un esfuerzo inferior a un año·persona			
Atributo	Modificabilidad			
Estímulo	Añadir arquitectura cognitiva para agentes			
Respuesta	Se realiza con un esfuerzo inferior a un año·hombre			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Tutor con emociones y personalidad	R13	N06	S14	
Compañero con emociones y personalidad	R13	N06	S14	

Escenario: M14	Se desea introducir un mecanismo de tutoría que funcione con Razonamiento Basado en Casos (CBR) con un esfuerzo inferior a un año·persona			
Atributo	Modificabilidad			
Estímulo	Añadir tutoría basada en CBR			
Respuesta	Se realiza con un esfuerzo inferior a un año·hombre			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Eliminar la estrategia de tutoría existente	R14		S15	
Eliminar el agente de planificación		NR07		
Introducir una nueva estrategia CBR	R15			

Escenario: M15	Se desea sustituir las explicaciones enlatadas del tutor por un mecanismo de generación de lenguaje natural con un esfuerzo inferior a seis meses·persona			
Atributo	Modificabilidad			
Estímulo	Añadir generación de lenguaje natural			
Respuesta	Se realiza con un esfuerzo inferior a seis meses·hombre			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Nuevo comportamiento para generar respuestas		N08		
Nuevos mecanismos de razonamiento sobre la ontología del mundo	R16		S16	

Escenario: M16	Se desea incluir un mecanismo de entrenamiento de grupos que incorpore miembros manejados por el ordenador con un esfuerzo inferior a dos años·persona			
Atributo	Modificabilidad			
Estímulo	Añadir compañeros manejados por agentes			
Respuesta	Se realiza con un esfuerzo inferior a dos años·hombre			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Introducir agentes para manejar miembros del grupo	R17	N06		
Introducir agente de modelado de grupo		N06	S17	

Escenario: M17	Se desea transformar el sistema en un museo virtual en el que se puedan visitar las instalaciones que se usan para el entrenamiento y que disponga de un guía virtual			
Atributo	Modificabilidad			
Estímulo	Crear un museo virtual			
Respuesta	El sistema permite usarlo como museo virtual			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Modificación de la estrategia de tutoría			S18	

Escenario: M18	Se desea transformar el sistema en un entorno que sirva para contar historias interactivas			
Atributo	Modificabilidad			
Estímulo	Usar el EV para *storytelling*			
Respuesta	El sistema permite su uso para contar historias			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Modificación de la estrategia de tutoría			S19	

Escenario: M19	Se quiere que el sistema pueda funcionar en dispositivos móviles			
Atributo	Modificabilidad			
Estímulo	Portar el sistema a dispositivos móviles			
Respuesta	El sistema funciona en móviles y PDAs			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Sustitución del EV	R18			
Instalación en dispositivo móvil	R19			

Escenario: M20	Se desea que el sistema dé soporte al entrenamiento de grupos formados por más de 100 usuarios de manera simultánea			
Atributo	Modificabilidad			
Estímulo	Escalar el sistema para grupos muy grandes			
Respuesta	El sistema escala bien			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Sustitución del mecanismo de edición-suscripción				C06
Distribución de los agentes en distintas máquinas			S20	

Escenario: M21	Se desea que el sistema soporte varias sesiones de entrenamiento en diferentes actividades de forma simultánea			
Atributo	Modificabilidad			
Estímulo	Realizar varias sesiones de entrenamiento simultáneas			
Respuesta	El sistema soporta varias sesiones			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Replicar los agentes que son únicos en una sesión			S21	
Modificar el formato de los mensajes			S22	

Escenario: M22	Se desea que el SIT permita la entrada de agentes externos que realicen funciones análogas a los existentes			
Atributo	Modificabilidad			
Estímulo	Transformar el SIT en un sistema abierto			
Respuesta	Se consigue sin afectar a la seguridad y a la fiabilidad del sistema			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Incluir mecanismos de seguridad	R20			C07

Escenario: M23	Se quiere que el sistema pueda funcionar a través de interfaces olfativas			
Atributo	Modificabilidad			
Estímulo	Añadir interfaces olfativas			
Respuesta	El sistema produce olores adecuados al entorno de entrenamiento			
Decisión de Diseño	**Riesgo**	**No riesgo**	**P. Sensible**	**P. Compromiso**
Modificación de la ontología del mundo			S23	

7.3.3.1. Puntos de Riesgo

R01 Si habitualmente existen varias acciones correctas para una situación, se puede dar el caso de que haya que recalcular muy a menudo la siguiente acción antes de dar una respuesta al alumno, lo que puede hacer que el sistema vaya más lento de lo requerido.

R02 La planificación puede requerir mucho tiempo de cálculo, por lo que se deben preferir planificadores que funcionen con lenguajes compilados.

R03 En caso de no poder utilizar respuestas prediseñadas, es posible que sea necesario realizar consultas al agente mundo que precisen razonamiento sobre la ontología, lo que podría llegar a requerir más tiempo de los tres segundos.

R04 Al existir un único tutor las acciones se procesan en secuencia. Existe el peligro de que los retardos en la red de los dos alumnos sean distintos y las acciones lleguen desordenadas. Queda abierta la posibilidad de utilizar marcas de tiempo para las acciones.

R05 Si el elemento no está preparado para funcionar dentro de un agente mediante la prestación de servicios, entonces aumentará el tiempo necesario para integrarlo debido a la adaptación necesaria.

R06 La separación arquitectónica entre la estrategia de tutoría y el modelo del estudiante no implica eliminar la dependencia entre ellos, por lo que la sustitución de uno de ellos puede afectar mucho al otro. Puede suceder que el modelo de estudiante no proporcione la información que necesita la tutoría para tomar decisiones, o que la tutoría no proporcione la información necesaria para elaborar el modelo del estudiante.

R07 La solicitud de servicios con paso de mensajes entraña el riesgo de que, aunque la información sea correcta a nivel sintáctico, puede no serlo a nivel semántico, por lo que debe comprobarse dicha corrección.

R08 El uso de un planificador externo conlleva cierta dependencia de la forma en que hay que modelar el dominio, los operadores y, en general, todos los datos, por lo que los agentes que manejen esos datos se pueden ver afectados por un cambio en el planificador. Para evitarlo debe diseñarse un formato para ellos que no se base en la representación del planificador.

R09 Dependiendo del algoritmo utilizado, la representación de las rutas puede variar, generalmente en forma de una lista de coordenadas o una lista de puntos de interés. Para que el sistema no dependa de este formato, se deberá disponer de un mecanismo que permita traducir coordenadas a puntos de interés y viceversa.

R10 En general, un cambio del simulador se debe a un cambio en el dominio de entrenamiento. Esto siempre supone un riesgo alto, pues la naturaleza de los simuladores puede ser muy distinta. Por tanto, la manera de integrar un nuevo simulador queda planteada como una cuestión abierta.

R11 Aunque la introducción de un planificador asegura que se le asigne al dispositivo háptico todo el tiempo que necesite, hace falta que el procesador en el que se ejecute tenga potencia suficiente para efectuar los cálculos, soportando también en caso necesario la ejecución del EV.

R12 La creación de un modelo específico para el agente de percepción entraña dos riesgos. El primero está relacionado con la posible aparición de inconsistencias entre los distintos modelos que se mantienen por separado pero que tienen efectos simultáneos sobre lo que ve el alumno. El segundo tiene que ver con la posible pérdida de rendimiento que tiene lugar al tener que mantener un modelo adicional.

R13 Una arquitectura cognitiva incorpora mecanismos complejos para dotar a un agente de rasgos de personalidad y emociones, por lo que puede constituir un elemento demasiado pesado que afecte mucho al rendimiento del sistema, especialmente si se utiliza en más de un agente.

R14 El cambio de la estrategia de tutoría es una modificación de una parte fundamental del sistema, por lo que es probable que su eliminación requiera un análisis cuidadoso de todas las interacciones que tiene con otros elementos del sistema.

R15 Existe un riesgo alto de que una estrategia basada en CBR necesite la realización de adaptaciones para poder integrarla en el modelo definido para esta arquitectura. Además, se desconocen muchos aspectos de lo que este tipo de tutoría supone.

R16 Existe la posibilidad de que el razonador que trabaja sobre la ontología no proporcione la información necesaria para que el generador de lenguaje natural pueda construir frases naturales con sentido.

R17 La introducción de compañeros de grupo manejados por el ordenador presenta un problema fundamental relacionado con la planificación de las tareas, ya que no sólo se debe planificar lo que se debe hacer, sino quién debe hacerlo. Esto requiere la utilización de un planificador que ofrezca la posibilidad de manejar esta información. El agente de tutoría también debe tener en cuenta a qué miembros del grupo debe supervisar y a cuáles no.

R18 La alternativa más realista para que el sistema se ejecute en un dispositivo móvil es que sea sólo el EV el que lo haga, conectándose por medio de una red *wifi* al centro de mensajes. Aún así, es necesario hacer uso de un EV simplificado y con características muy reducidas respecto a lo que puede ejecutarse en un ordenador de sobremesa.

R19 Si se desea instalar la aplicación completa en el dispositivo móvil, hay plataformas, como JADE, que se ejecutan en estos dispositivos. Sin embargo, dada la menor capacidad de un dispositivo móvil en comparación con un ordenador de sobremesa, habría que eliminar muchas funcionalidades, especialmente las más costosas, como la planificación y la planificación de rutas.

R20 Abrir la plataforma de agentes supone la posibilidad de dejar entrar agentes maliciosos o simplemente inconvenientes que pueden provocar un mal funcionamiento del sistema. Abrir el sistema implica implementar medidas de seguridad basadas en la certificación del origen de los agentes para que sólo los agentes autorizados puedan entrar en el sistema.

7.3.3.2. Puntos sin Riesgo

N01 En caso de que pueda hacerse uso de respuestas prediseñadas, tres segundos es tiempo suficiente para dar una respuesta.

N02 Al prestar el servicio desde un comportamiento, se puede desactivar dicho comportamiento de manera dinámica sin consecuencias.

N03 El hecho de situar un elemento dentro (o detrás) de un agente permite que se utilicen los mecanismos introducidos para facilitar la modificabilidad, como el paso de mensajes o la edición-suscripción.

N04 La comunicación a través de la solicitud de servicios con paso de mensajes debilita la relación entre quien solicita el servicio y quien

lo presta, pues ésta se establece en tiempo de ejecución, lo que favorece la modificabilidad.

N05 La comunicación a través del Centro de Mensajes mediante un mecanismo de suscripción por tipo de mensaje hace que el resto del sistema no se vea afectado si se sustituye un elemento por otro que proporcione la misma información. De la misma manera, el nuevo elemento puede recibir todos los mensajes que necesite sin preocuparse de quién hace el envío.

N06 La introducción de un nuevo agente puede realizarse utilizando los mismos mecanismos que con el resto de agentes, por lo que técnicamente sólo está supeditado a que existan los servicios que le proporcionen la información que necesita para realizar su trabajo.

N07 La eliminación de un agente no entraña riesgos siempre que nos aseguremos de que sus servicios no los utilizaba ningún otro agente.

N08 Haciendo uso de los mecanismos existentes se puede introducir el nuevo comportamiento, que puede solicitar los servicios que requiera de otros agentes.

7.3.3.3. Puntos Sensibles

S01 Para tardar menos en proporcionarle una respuesta al estudiante, cuando lleva a cabo una acción se le solicita al planificador la siguiente acción a realizar, de manera que se tenga preparada para cuando el estudiante la realice. Si hay varias acciones posibles, es posible que el estudiante realice otra.

S02 Se depende de la potencia de cálculo de la máquina en que se realice la ejecución y de la eficiencia del planificador.

S03 No siempre se puede hacer uso de respuestas prediseñadas.

S04 Este escenario depende de muchos factores que no son arquitectónicos, como la implementación del motor de *render* y la complejidad de los modelos 3D que hay que dibujar. Habrá que tener en cuenta estos factores al elegir el EV y hacer los diseños 3D.

S05 El hecho de que dos acciones simultáneas se procesen bien depende en gran medida de que lleguen en el orden adecuado al SIT en caso de que este orden sea importante.

S06 Para poder cumplir con la estimación son necesarias varias cuestiones, como que la estrategia de tutoría se encuentre implementada y que admita el enfoque orientado a servicios que se ha planteado como solución. También depende de la compatibilidad con el modelo del estudiante existente.

S07 Para poder realizar la sustitución se necesita que el nuevo modelo sea compatible con la estrategia de tutoría, y también que se le proporcione la información necesaria en un formato que al menos pueda transformar en el que utiliza.

S08 El formato de los operadores en el planificador es un punto que puede constituir el origen de problemas para integrarlo en el SIT.

S09 El hecho de que el algoritmo de cálculo de rutas trabaje con grafos o con coordenadas puede causar la necesidad de realizar conversiones entre ambos formatos.

S10 Debe ser posible hacer que desde el EV se envíe al Centro de Mensajes la información que necesita el SIT y, a ser posible, en el mismo formato que se esté utilizando.

S11 La introducción de dispositivos hápticos es sensible en tanto que necesitan realizar cálculos muy precisos con una frecuencia muy alta en comparación con la velocidad de ejecución del EV. Por ello, es necesario que puedan ejecutarse de manera separada, interactuando sólo en los momentos precisos.

S12 La introducción del agente de percepción supone añadir al sistema un agente que realiza unos cómputos costosos, por lo que puede incrementar la carga del sistema y disminuir el rendimiento.

S13 Añadir un tutor virtual con aspecto y comportamiento simples se puede realizar en un tiempo bastante inferior a los seis meses, por lo que el cumplimiento está supeditado al realismo del que se quiera dotar al personaje y a lo que permita hacer el EV.

S14 Se depende de la existencia de otros elementos, como una representación virtual con cierta capacidad expresiva.

S15 La estrategia de tutoría es una pieza fundamental dentro del SIT. Aunque la única dependencia notable se encuentra en el modelo del estudiante, pues le proporciona la información con la que

completarlo, esta sustitución puede suponer un cambio significativo en el funcionamiento del sistema.

S16 Razonar sobre la ontología puede introducir retardos en el sistema.

S17 La introducción del modelado de grupo requiere extender el modelo del estudiante con aspectos que reflejen el comportamiento del grupo. También requiere que la estrategia de tutoría pueda tener en cuenta aspectos relativos al grupo en su conjunto.

S18 Transformar un sistema de entrenamiento en un museo virtual puede requerir pocas modificaciones. Las explicaciones del guía son análogas a las del tutor, el planificador puede utilizarse para decidir en qué orden se debe visitar el museo y el planificador de rutas puede proporcionar el camino por el que se desplaza el guía. Es necesario que el agente experto maneje el conocimiento acerca del museo.

S19 Transformar un sistema de entrenamiento en un sistema de *storytelling* puede requerir pocas modificaciones. Las explicaciones del narrador son análogas a las del tutor, el planificador puede utilizarse para decidir en qué orden se debe contar la historia y el planificador de rutas puede proporcionar el camino por el que se desplazan los personajes.

S20 La distribución en distintas máquinas debe realizarse cuidadosamente para que no se produzcan errores por una mala sincronización entre los agentes. Los mejores candidatos a ser distribuidos son los agentes de estudiante y los de comunicación, que son los que se replican en función del número de estudiantes.

S21 Replicar los agentes por cada sesión de entrenamiento supone que se puede elevar mucho su número, con la consiguiente penalización para el rendimiento de la aplicación. Si, para solucionarlo, se opta por distribuir los agentes, probablemente tenga más sentido ejecutar cada sesión de entrenamiento por separado, en máquinas diferentes.

S22 Si se ejecutan varias sesiones juntas, los mensajes deberían identificarse para que se sepa a qué sesión pertenecen y que, de ese modo, se sepa quién los debe recibir. Existe la posibilidad de que este cambio, aunque de forma ligera, afecte a todos los agentes del sistema.

S23 La introducción de interfaces olfativas requerirá de la introducción de información en la ontología del mundo acerca de los olores que deben producirse cuando se realiza una acción concreta o cuando se pasa por una zona determinada del EV.

7.3.3.4. Puntos de Compromiso

C01 Cuantas más respuestas prediseñadas se utilicen, menos flexible será el sistema de pregunta-respuesta y más costará modificar la manera en que se enuncian.

C02 Al procesar las acciones en secuencia se puede perder eficiencia en la ejecución.

C03 La solicitud de servicios con paso de mensajes entraña la necesidad de interpretar los mensajes, lo que resta eficiencia al procesamiento del servicio.

C04 La representación de datos y operadores de forma independiente a la manejada por el planificador obliga a hacer una traducción de todos ellos, lo que disminuye el rendimiento del sistema.

C05 Mantener un modelo adicional crea un curioso conflicto de rendimiento. No mantenerlo implica la necesidad de realizar más consultas y cálculos, mientras que mantenerlo agiliza esta parte pero carga la memoria con otro modelo del mundo que puede llegar a tener un tamaño bastante grande, además de duplicar el coste de las actualizaciones.

C06 Uno de los inconvenientes que se le atribuyen al mecanismo de edición-suscripción es que se comporta mal cuando se debe escalar la solución, produciendo una disminución del rendimiento. Por lo tanto, para que no se decremente demasiado, habrá que sustituir el mecanismo de suscripción por otro que proporcione mejor rendimiento.

C07 Es habitual que la implementación de políticas de seguridad afecte al rendimiento de un sistema. Así, para que no afecte demasiado, se debe probar un sistema restrictivo a la hora de dejar entrar a nuevos agentes pero que confíe en ellos una vez estén dentro.

7.4. Resultados de la Evaluación

La evaluación de la arquitectura descrita en el Capítulo 5 ha proporcionado unos resultados que, dentro un orden, han sido los que se esperaban antes de realizar la evaluación.

Tal como afirman los autores del método ATAM en (Clements et al., 2002b), los resultados de la realización de la evaluación dependen de los participantes en dicha evaluación.

Aunque a primera vista podría parecer un inconveniente, puesto que los resultados no son objetivos, en realidad sucede lo contrario. Teniendo en cuenta que los participantes en la evaluación son los interesados en el desarrollo del posterior sistema, los factores que hacen acto de presencia son los que conciernen a las personas implicadas, y son por lo tanto los que de una u otra manera afectan a la arquitectura. No habrá, por el contrario, factores ajenos a los interesados que se inmiscuyan en el proceso de evaluación.

De esta manera, muchos de los puntos significativos identificados en la evaluación han sido previamente tratados en el proceso de diseño, por lo que se puede esperar que sus implicaciones y sus efectos sean los ya descritos en la documentación de la arquitectura.

Sin embargo, la participación en la evaluación de miembros ajenos al equipo de desarrollo, así como la generación de escenarios de evolución, han permitido identificar algunos factores que no se habían tenido en cuenta previamente y que pueden conducir a algunas mejoras en el diseño de la arquitectura.

La generación de escenarios de evolución también ha dado pie a la identificación de puntos sensibles en lo relativo a la flexibilidad de la arquitectura, aunque algunas de las modificaciones propuestas llevan al sistema hasta unos límites que hacen poco probable que la arquitectura se adecúe a los fines previstos, a pesar de que se pueda llegar a obtener un sistema en funcionamiento.

7.5. Reflexiones

Los resultados de la evaluación han arrojado un balance bastante favorable en cuanto a los beneficios proporcionados por la realización de un diseño sistemático y pensado, ya que gran parte de los factores que han hecho acto de presencia durante la evaluación ya se habían tenido en cuenta a la hora de realizar el diseño.

Sin embargo, este hecho pone de manifiesto que un método de evaluación como ATAM, que se basa en unos principios muy cercanos a los utilizados en la realización del diseño, puede resultar demasiado aparatoso de llevar a la práctica en circunstancias más cercanas a los desarrollos reales.

Si los miembros implicados en ambos procesos son básicamente los mismos, el diseño de la arquitectura se verá mucho más beneficiado de la realización de un diseño cuidado que de la posterior evaluación, en la que es poco probable que se descubran elementos que no se habían tenido en cuenta. Por lo tanto, resulta más conveniente dedicar más recursos a la realización

de la arquitectura que a la posterior evaluación, que aunque deba realizarse, podrá llevarse a cabo de una forma más ligera.

Por lo tanto, la realización de una evaluación completa con ATAM, de cara a la práctica, parece más adecuada en el caso de que se desee evaluar la arquitectura de un sistema desconocido o, también, si el diseño de la arquitectura se ha realizado sin tener en cuenta los atributos de calidad del sistema.

Capítulo 8

Resultados y Conclusiones

"No hay nada tan estúpido como vencer;
la verdadera gloria estriba en convencer"
— Victor Hugo

8.1. Introducción

Tras la publicación del que es considerado como el primer trabajo relacionado con los Sistemas Inteligentes de Tutoría (Carbonell, 1970), se ha realizado una intensa labor de investigación en este campo. Los Sistemas Inteligentes de Tutoría han avanzado del mismo modo en que lo han hecho otras áreas de la informática, a las que además se ha intentado sacar partido para proporcionar mayores beneficios a los usuarios de estas aplicaciones.

De esta manera, a los avances que se han producido en materia de representación de conocimiento, de modelado de los estudiantes o de desarrollo de distintas estrategias de tutoría, se han unido la aplicación de otras tecnologías relacionadas con la interacción con los usuarios. Así, se ha pasado de sistemas puramente textuales a otros que utilizan gráficos bidimensionales, interfaces multimedia y sistemas de realidad virtual.

Precisamente con la introducción de los sistemas basados en realidad virtual se ha dado cabida a un tipo de aplicaciones que anteriormente no habían encontrado un medio adecuado, como son los sistemas orientados al entrenamiento. La posibilidad de disponer de una representación tridimensional de un entorno real, con objetos manipulables de forma similar a como se haría en el entorno real, ha facilitado que pueda darse el paso desde sistemas centrados en la enseñanza a sistemas que también se ocupan del entrenamiento.

Existen numerosas áreas que se han beneficiado de esta evolución, entre las que se encuentran el entrenamiento médico, militar e industrial. Este es

un campo en el que ha tenido lugar mucha de la actividad del autor de este trabajo, con el desarrollo de sistemas de entrenamiento para personal de centrales nucleares.

Junto con la evolución de las tecnologías que se ven - las distintas formas de interactuar con la aplicación - y las que no se ven - las propias de un SIT - también se ha producido una evolución de las tecnologías que no hacen acto de presencia de manera palpable, pero que son fundamentales para la existencia de cualquier sistema: las técnicas y herramientas utilizadas en el desarrollo.

En este sentido, los Sistemas Inteligentes de Tutoría no han sido ajenos tampoco a los avances en los paradigmas de desarrollo de software, pasando de ser sistemas diseñados e implementados con técnicas y herramientas de la programación estructurada a aplicaciones orientadas a objetos y, más recientemente, a la utilización de servicios web o de agentes, por mencionar algunos de los más relevantes.

Uno de los méritos de los investigadores en Sistemas Inteligentes de Tutoría consiste en haber sido capaces de definir una arquitectura propia que ha sido ampliamente utilizada a lo largo de los años (ver Figura 4.1, pag. 123), independientemente de la manera de implementarla. Esta arquitectura, además, ha dado pie a que distintos equipos de investigación se especialicen en el desarrollo de cada uno de los diferentes módulos de la arquitectura, lo que a su vez ha propiciado el uso de esta arquitectura para las aplicaciones que continúan desarrollándose.

Como ya se mencionaba al principio del trabajo, muchos de los avances en este área provienen del campo de la inteligencia artificial y de la psicología cognitiva. Sin embargo, lo que no ha experimentado un avance tan notable son las cuestiones relacionadas con la ingeniería de software. Pareciera como si la simple definición de una arquitectura sencilla para estos sistemas fuera suficiente.

Sin embargo, hay autores que ya se han percatado de que la aparente separación entre las funciones de las que se ocupa cada uno de los cuatro módulos de la arquitectura de los SIT no es sino eso, aparente, y en muchos casos no llega a ser ni siquiera deseable (Freedman, 2000). Así, una determinada estrategia de tutoría es adecuada para enseñar un determinado tipo de conocimiento, pero no otro. De igual manera, una forma de modelar a un estudiante puede ser adecuada para una estrategia de tutoría, pero no para otra.

Debido a lo costoso que resulta desarrollar estos sistemas, y a la especialización que sufren los grupos de investigación, resulta frecuente encontrar SIT notablemente sofisticados en unos aspectos, a la par que bastante sencillos en otros (Freedman, 2000).

Por esta razón, unida a la experiencia acumulada por el equipo de investigación al que pertenece el autor de este trabajo, se fijó como objetivo de la presente tesis doctoral avanzar hacia un necesario intercambio de componentes desarrollados por distintos grupos, de manera que se pueda suplir con un componente la carencia de otro componente que haya sido desarrollado por un equipo no experto en ello.

Para conseguir este objetivo resulta fundamental que las aplicaciones ofrezcan cierta facilidad para intercambiar componentes, y poco después de desarrollar una primera solución al problema nos topamos con una realidad que distintos autores han ido sacando a la luz con el paso de los años.

Como ya se señalaba en (Parnas, 1972), escrito en la época de la programación estructurada, la modificabilidad de una aplicación no depende de que ésta sea modular, sino de cómo se realice la división en módulos, es decir, que hay que diseñar buscando la modificabilidad para poder obtenerla; no basta sólo con dividir el código en módulos de forma arbitraria.

Con el paso de los años y la llegada de la orientación a objetos volvemos a encontrarnos con un problema similar descrito en (Haythorn, 1994). Los objetos, abanderados de la modularidad, de la encapsulación y de la ocultación de información (que disfrazaba una simple privacidad de los atributos) no son más que el mismo problema pero con distintos ropajes, al menos en lo que a modificabilidad se refiere. Así, el autor del citado trabajo vuelve a señalar la necesidad de diseñar específicamente para conseguir modificabilidad en el software como solución para que el software sea modificable.

Durante los últimos años, los agentes han hecho acto de presencia como una herramienta para facilitar el cada vez más complejo *arte* de crear software. Pero, de nuevo, volvemos a encontrarnos con los mismos problemas de antes. En (Wooldridge y Jennings, 1998) se alerta de la similitud entre los problemas señalados en (Webster, 1995) para el desarrollo orientado a objetos y los que están surgiendo en el desarrollo orientado a agentes.

Parte de estos problemas son debidos a que las nuevas tecnologías se abrazan como una panacea que viene a resolver los problemas anteriores, sin detenerse mucho a pensar si realmente es cierto que estos problemas se resuelven sin más o si, por el contrario, se crean otros nuevos, como de hecho sucede. En ese sentido, no dejan de llamar la atención, de entre una amplia gama de ejemplos, dos bastante ilustrativos. El primero de ellos aparece en (Odell, 2000), donde se afirma

> *One agent could be constructed that does everything, but fat agents represent a bottleneck for speed, reliability, maintainability, and so on (i.e., there are no omnipotent agents). Dividing*

> *functionality among many agents provides modularity, flexibility, modifiability, and extensibility.*

Esta afirmación se realiza en un informe técnico de OMG (*Object Management Group*), organización encargada, por ejemplo, del desarrollo y mantenimiento de UML. Si en un documento de referencia como puede ser éste se encuentra una afirmación de este estilo, en la que se les concede a los agentes unas propiedades sin mayor justificación, no se puede sino esperar que estas propiedades, en general, se den por sentadas sin mayores preocupaciones.

Otro ejemplo aparece en un trabajo de publicación más reciente, en el que los autores, quizá dejándose llevar por afirmaciones como la anterior, exponen (Ramdane-Cherif et al., 2005)

> *However, we exploit the advantage of the multi-agents systems like the flexibility, the performance, the parallelism and the high level of abstraction for the construction of our platform.*

Lo más extraño de esta afirmación es que se realice en un trabajo que describe una aplicación desarrollada para ayudar a evaluar arquitecturas software basándose en los mismos métodos que se han utilizado en el presente trabajo. Estos métodos, sin embargo, dejan claro desde un principio que una tecnología no es propietaria de esas características por sí misma, sino que es necesario utilizarla de la forma adecuada para que esas propiedades se cumplan.

Ambos ejemplos constituyen una muestra de la falta de madurez existente en el desarrollo de software orientado a agentes (y en el desarrollo de software en general), y más concretamente en el aspecto que resulta de mayor interés en este trabajo, que son las arquitecturas software.

A la luz del breve resumen que se ha realizado en esta introducción, se hacía necesario un enfoque más meditado que permitiese desarrollar Sistemas Inteligentes de Tutoría cuyos componentes fuesen fácilmente sustituibles. Para ello, se ha propuesto una solución que se centra en uno de los aspectos que más pueden afectar a la modificabilidad del software, como es su arquitectura, se ha justificado la utilización de agentes para diseñar esta arquitectura y se ha procedido a realizar el diseño prestando especial atención a la introducción de mecanismos que faciliten la modificabilidad de la misma.

8.2. Método de Trabajo

La propuesta que se realiza en este trabajo consiste en definir una arquitectura para Sistemas Inteligentes de Tutoría que sirva como base para

desarrollarlos, dejando libertad para que quien utilice la arquitectura pueda concretar, en una fase posterior del diseño, la manera en que se llevan a cabo las funciones de cada uno de los módulos que componen el sistema.

Además, se ha tratado la problemática de que los distintos módulos se puedan agregar y eliminar en función de los requisitos que plantee el entrenamiento, lo cual hace que el sistema se pueda adaptar a las necesidades que surjan en cada momento.

Esta perspectiva proporciona un marco en el que encuadrar el uso de, por ejemplo, distintos métodos de planificación o de cálculo de trayectorias en un Entorno Virtual, además de poder activarlos o desactivarlos cuando sea requerido.

Para ello, ha sido necesario abordar el diseño de la arquitectura software del sistema de forma que se considere explícitamente qué modificaciones debe soportar la arquitectura, de forma que se puedan incluir en el diseño mecanismos que faciliten esta labor.

El estudio de los trabajos existentes desveló, por un lado, las características que es frecuente encontrar en este tipo de sistemas, lo que ha servido para poder tenerlas en cuenta a la hora de decidir qué funcionalidades debía soportar la arquitectura y qué bloques de funcionalidad deseábamos que se pudiesen sustituir. Por otro lado, también ha resultado útil para identificar la forma en que se suele abordar la construcción de estos sistemas y para revelar las carencias que suelen existir en su desarrollo.

En este punto se tomó la decisión de diseñar una arquitectura lo suficientemente abierta como para que diese cabida a distintas soluciones presentes en la literatura revisada, además de completarla con una serie de recomendaciones metodológicas que ayudasen a mantener las propiedades de las que se dotaría a la arquitectura.

Finalmente, apoyándose tanto en la arquitectura como en las recomendaciones metodológicas, se realizó la implementación de un *framework* que sirviese como base para la integración de los elementos presentes en la arquitectura, que pudiera ser utilizado para desarrollar Entornos Virtuales de Entrenamiento con Tutoría Inteligente sin necesidad de realizar todo el esfuerzo de implementación desde el principio, y que facilitase la adopción de la arquitectura propuesta.

Del análisis de los precedentes propios y ajenos, teóricos y empíricos, se recopiló una lista de los elementos presentes en distintos sistemas que se consideraron más importantes para ser incluidos en la propuesta de solución:

- De este análisis surge la necesidad de considerar como componentes básicos de la arquitectura elementos que permitan incluir tanto la *estrategia de tutoría* como el *modelado del estudiante*, así como el

conocimiento experto sobre el dominio del problema, que son los elementos que conforman la estructura básica de un Sistema Inteligente de Tutoría. También es un elemento básico el *entorno virtual*, sin el cual el sistema pierde gran parte de su utilidad como entorno de entrenamiento.

- Además, por el hecho de estar realizando un entrenamiento dentro de un Entorno Virtual, se hace necesario modelar cierta información semántica que represente las acciones que el estudiante puede realizar en él, así como los objetos con los que puede interactuar y su estado. Para ello se considera necesaria la inclusión de un elemento que maneje la *información del mundo virtual*.

- La necesidad de realizar el entrenamiento en entornos complejos en los que se interactúa con elementos que no están controlados directamente por los usuarios, sino por algún tipo de simulación, marca la necesidad de incluir otro elemento que se haga cargo de la interacción con dicha *simulación*.

- Para poder realizar una tutoría flexible y que dé libertad a los estudiantes, algunos sistemas plantean la posibilidad de no dirigir al estudiante por un único rumbo prefijado, sino que se le da la posibilidad de seguir el camino que considere más adecuado y el sistema adapta la solución a la situación creada por el estudiante cuando es necesario. Con el objetivo de llevar esta posibilidad hasta los extremos más amplios, se propone la utilización de un *planificador* que trace la solución a un ejercicio en la medida en que un estudiante vaya avanzando por el camino que elija para su resolución.

- Finalmente, también se identifican las *interacciones* que típicamente tienen lugar en estos sistemas para darles soporte dentro de la arquitectura, lo que nos lleva a identificar la realización de *acciones* y de *preguntas* y la *navegación* por el Entorno Virtual.

Una vez identificadas estas características, se valoran las distintas opciones disponibles para diseñar e implementar el sistema. Se llega a la conclusión de que la utilización de agentes software constituye una opción prometedora que puede facilitar la tarea que se aborda y que puede incluso aportar propiedades de interés a la arquitectura.

Al tomar esta decisión, se opta también por acudir al área de la ingeniería de software orientada a agentes para desarrollar la arquitectura, eligiendo una organización jerárquica para el diseño de la arquitectura.

A medida que la arquitectura va siendo diseñada e implementada se hace patente que hay algo que no se ha considerado, ya que la arquitectura y el sistema implementados no presentan las propiedades que se esperaba obtener de ellos.

Se decide realizar un análisis de la arquitectura, que muestra que el enfoque jerárquico y la manera de llevarlo a la práctica no constituyen una buena solución para el problema planteado.

Tras un periodo de estudio, se acude al área de las arquitecturas software como camino para resolver los problemas encontrados en la solución inicial. Se identifican métodos de diseño y evaluación de arquitecturas, y se opta por utilizar los desarrollados en el *SEI* por cubrir todo el espectro de diseño, documentación y evaluación de las arquitecturas.

Se recopila un conjunto de escenarios de calidad que pone sobre la mesa los atributos de calidad que se le quieren dar a la arquitectura y que, aunque conocidos de forma implícita, nunca se han enunciado de forma explícita. Es en este momento cuando cobran forma y permiten dirigir el diseño con unos objetivos claros.

Después de considerar la posibilidad de no usar agentes en el nuevo diseño de la arquitectura, pues los expertos en el área dicen que la arquitectura es independiente de estas cuestiones, se llega a la conclusión de que su inclusión facilita la utilización de determinados mecanismos presentes en el paradigma de la orientación a agentes. Esta decisión, a la postre, conduce a la solución descrita en el capítulo 5, cuya evaluación confirma que posee las propiedades de las que se le pretendía dotar en la fase de diseño de la arquitectura.

El diseño de esta arquitectura se ha visto acompañado del desarrollo de una segunda implementación, que actualmente se encuentra en un estado bastante avanzado y que en los próximos meses se pretende poner a disposición del público, para que pueda utilizarse en el desarrollo de Entornos Virtuales de Entrenamiento. De él se espera obtener una retroalimentación positiva respecto a características que se deban seguir incluyendo, lo que hará evolucionar el sistema hacia lo que pueda demandar la comunidad de usuarios.

8.3. Aportaciones

En esta sección se presentan los resultados obtenidos con la elaboración de la presente tesis doctoral, haciendo especial énfasis en las aportaciones originales y en las mejoras proporcionadas por la solución desarrollada.

Al igual que los objetivos expuestos en el capítulo 3, estas aportaciones se han producido en tres áreas, como son el diseño de la arquitectura, la

elaboración de las recomendaciones metodológicas y la implementación de una infraestructura a partir de las aportaciones anteriores.

8.3.1. Resultados Relativos a la Arquitectura

El resultado de la investigación expuesta es una arquitectura software para Entornos Virtuales de Entrenamiento con Tutoría Inteligente. En esta arquitectura se han definido los elementos que deben formar parte del Entorno Virtual de Entrenamiento, así como los que deben conformar el Sistema Inteligente de Tutoría, que es uno de los elementos integrantes del sistema de entrenamiento.

Como primera aportación relevante del diseño de la arquitectura se encuentra el hecho de que incluye elementos que no contemplan otras arquitecturas de manera global, como son el planificador, el planificador de rutas, el modelo semántico del mundo virtual y el simulador, además de los componentes básicos de un SIT, como son el módulo de tutoría, el experto y el de modelado de los estudiantes.

Con esto se consigue que la arquitectura admita la utilización de un mayor número de funciones que otras arquitecturas propuestas anteriormente, y por lo tanto permite el desarrollo de sistemas más complejos y completos que los analizados en el capítulo 2.

Además, se ha preparado la arquitectura para que admita la inclusión, de manera sencilla, de otros elementos de importancia, como un agente tutor virtual que controle la presencia física del tutor en el Entorno Virtual.

También se han definido unos comportamientos básicos para cada uno de los agentes, junto con los servicios que presta cada uno de ellos, de manera que sirvan como base para desarrollar la funcionalidad que debe llevar a cabo cada uno de los agentes que forman el sistema.

Otra aportación importante de esta tesis se encuentra en la aplicación crítica de métodos de diseño y evaluación de arquitecturas software a la elaboración de una arquitectura software basada en agentes.

Actualmente, la comunidad que investiga sobre sistemas multiagente ha prestado poca atención al trabajo realizado en el área de las arquitecturas software, obviando los resultados obtenidos en un área que también debería ser relevante para el desarrollo de sistemas multiagente y que cuenta con mayor experiencia en el desarrollo de software.

En la elaboración del estado de la cuestión se han encontrado muy pocos ejemplos de trabajos que hayan realizado incursiones en ambas áreas. Destacan (Boucké et al., 2006; Boucké y Holvoet, 2008) y (Woods y Barbacci, 1999), los cuales se han centrado más en el proceso de evaluación de la arquitectura que en su diseño.

Sin embargo, es precisamente al diseño a lo que hay que prestarle mayor atención inicialmente, ya que la evaluación nos puede decir si una arquitectura presenta unos determinados atributos de calidad o no, pero la manera de hacer que los presente es precisamente a través del diseño.

Este trabajo constituye, por tanto, una aportación novedosa en lo que a la aplicación de métodos de diseño de arquitecturas software a los sistemas multiagente se refiere.

Esta experiencia ha proporcionado un resultado no esperado inicialmente, como es la conveniencia de considerar la utilización de agentes en el diseño de la arquitectura. Esta decisión permite incluir mecanismos de diseño arquitectónico que posiblemente no se utilizarían de no haberse considerado explícitamente el uso de agentes en el diseño de la arquitectura.

Otra aportación original de este trabajo consiste en la aplicación al desarrollo de Sistemas Inteligentes de Tutoría basados en agentes de los mecanismos proporcionados por estos últimos para la consecución de una mayor modificabilidad de la arquitectura. De esta manera, la combinación de tres elementos como son las páginas amarillas, un esquema editor-suscriptor y la prestación de servicios por parte de los agentes, ha conducido a una solución que, de forma contrastada, proporciona al sistema la modificabilidad buscada.

Estas son algunas propiedades con las que se ha dotado a la arquitectura:

- No es una arquitectura genérica que valga para cualquier tipo de Entorno Virtual de entrenamiento. Es válida para Entornos Virtuales de Entrenamiento procedimental, independientemente de su naturaleza, siempre que admitan la definición de las actividades como una serie de acciones que deben realizarse de manera secuencial.

- Permite el entrenamiento de varios estudiantes de forma simultánea realizando tareas complementarias en una misma actividad.

- Permite la incorporación de aplicaciones distintas a las consideradas durante el diseño de la arquitectura a través del Centro de Mensajes. Estas aplicaciones podrán enviar mensajes, que serán recibidos por el resto de las aplicaciones que forman el sistema de entrenamiento si son de los tipos a los que se han suscrito. De igual manera, esta aplicaciones se podrán suscribir a los tipos de mensajes que deseen recibir. El resto de aplicaciones ignoran por completo tanto la identidad de quien les envía los mensajes como de quien recibe los que ellos envían, lo que permite su sustitución sin afectar al resto de elementos del sistema.

- Permite deshabilitar los elementos que no sean necesarios en una determinada situación sin afectar al resto de elementos del sistema.

- Dentro del SIT, permite la inclusión de nuevos agentes que hagan uso de los servicios proporcionados por los existentes sin necesidad de realizar modificaciones en el sistema.
- Permite deshabilitar comportamientos de los agentes que no realicen tareas de utilidad en una actividad determinada.
- Permite el intercambio de agentes por otros que proporcionen servicios análogos sin necesidad de modificar los agentes que hagan uso de estos servicios.

El resultado de esta aportación es que se han racionalizado las afirmaciones que proclaman que los sistemas basados en agentes son más modificables que los que utilizan tecnologías más antiguas como los objetos. Además, se ha definido un marco que permite determinar en qué situación y a través de la utilización de qué mecanismos puede afirmarse que un sistema basado en agentes es flexible.

Así pues, tras la realización de este trabajo, se dispone de argumentos sólidos para afirmar que se ha desarrollado una arquitectura software basada en agentes que además, y no como consecuencia de ello, es modificable en unos determinados puntos porque se ha diseñado para que así sea.

Este trabajo también constituye una experiencia novedosa, si bien no es la primera vez que se lleva a cabo, en lo que se refiere a la evaluación de una arquitectura software basada en agentes utilizando ATAM. Esta evaluación no ha puesto de manifiesto problemas relevantes relacionados con la arquitectura, puesto que el proceso de diseño ha permitido controlar bastante ese aspecto.

Sin embargo, lo que sí ha permitido esta evaluación de la arquitectura es la elaboración de una lista de características relevantes de la misma, tanto de sus puntos fuertes como, lo que resulta de mayor interés, de sus posibles puntos sensibles o comprometidos.

Esta lista puede servir para tratar de atenuar los efectos de alguno de los inconvenientes del diseño, así como para dirigir las modificaciones que a buen seguro sufrirá este diseño en un futuro. No resulta necesario, ni tan siquiera conveniente, intentar dar solución a muchas de las debilidades identificadas, ya que las mejoras de algunas de ellas conducirían inevitablemente a empeorar otros aspectos de mayor importancia para los objetivos actuales (Bass et al., 2003).

8.3.2. Resultados Relativos a las Recomendaciones Metodológicas y de Diseño

Aunque la aportación principal de este trabajo de tesis lo constituye la arquitectura software para entornos virtuales de entrenamiento, la relativa complejidad de la misma junto con la cantidad de documentación que la acompaña pueden causar que su utilización se vea entorpecida.

Por este motivo, también resulta relevante la aportación introducida por las recomendaciones metodológicas y de diseño, en cuanto a que proporcionan una ayuda para comprender los fundamentos que han guiado el diseño de la arquitectura, por un lado, y guían a los usuarios de la misma en el proceso de modificación de sus distintas partes para adaptarla a sus necesidades.

De esta manera, las recomendaciones metodológicas constituyen una aportación relevante en cuanto que guían al usuario de la arquitectura en la utilización y modificación de la misma.

Por una parte, las recomendaciones metodológicas descritas facilitan la adopción de la arquitectura por parte de otros grupos, lo cual puede ser un punto fuerte de cara a su expansión. Esto se ha realizado a través de la elaboración de una lista de comprobación que sugiere los elementos de la arquitectura a utilizar en función de los requisitos de la aplicación a construir. También se ha diseñado una vista combinada de la arquitectura que muestra todos sus elementos, sobre la cual se pueden ir marcando los componentes necesarios y que permite, de manera visual, comprobar si se han seleccionado los elementos requeridos.

Por otra parte, al indicar cómo se deben llevar a cabo las modificaciones, tanto del sistema en su totalidad como del SIT en particular, se proporciona un medio para que la arquitectura no se degrade por causa de las modificaciones realizadas. Este es un aspecto que se debe evitar a toda costa, ya que la degradación de la arquitectura es uno de los principales motivos que llevan a la sustitución de un sistema software.

Por su parte, las recomendaciones de diseño aportan la visión de conjunto de los criterios de diseño utilizados en la elaboración de la arquitectura, los cuales constituyen un elemento que permite mantener la integridad conceptual de la misma. Esto representa uno de los principales factores que permiten mantener los atributos de calidad con los que se ha dotado a la arquitectura descrita.

Por lo tanto, la principal aportación realizada con la elaboración de las recomendaciones metodológicas y de diseño es su contribución a mantener la integridad conceptual de la arquitectura, lo que retrasa el inevitable proceso de deterioro causado por la sucesivas modificaciones que es de esperar que tengan lugar.

8.3.3. Resultados Relativos a la Implementación de la Arquitectura

La definición de la arquitectura objeto del trabajo se ha utilizado para diseñar e implementar Entornos Virtuales de Entrenamiento que utilizan la estructura y los mecanismos definidos en ella.

El resultado de esta implementación es un *framework* que ya se ha probado con éxito en dos entornos de entrenamiento sencillos de complejidad creciente. Gracias a estos resultados, se está elaborando un tercer ejemplo de aplicación real, de complejidad más elevada, para el entrenamiento de personal que trabaja en centrales nucleares.

Actualmente, la mayor complejidad de la aplicación de la implementación realizada a distintos entornos radica principalmente en lo laborioso que resulta el proceso de definición del entorno de entrenamiento. Por una parte, se debe definir el dominio del problema y los operadores que debe utilizar el planificador. Por otra, se debe elaborar toda la información semántica relativa al Entorno Virtual que debe utilizar el Agente Mundo. Ambas labores son las que, una vez implementado el *framework*, requieren la mayor parte del esfuerzo de desarrollo de nuevas tareas de entrenamiento.

Por ello, una de las asignaturas pendientes se encuentra en la elaboración de herramientas de autor que faciliten la creación de nuevas actividades de entrenamiento (ver Figura 8.1).

En lo tocante a la implementación del sistema en sí, ésta se ha llevado a cabo utilizando distintos lenguajes de programación para cada uno de los subsistemas que integran la aplicación. Todos ellos han podido comunicarse entre sí a través del Centro de Mensajes, lo que constituye una primera prueba de la utilidad de este medio de comunicación y de la flexibilidad que introduce en el sistema.

Durante el desarrollo de la aplicación se hace necesario realizar numerosas pruebas para comprobar su correcto funcionamiento. Para probar las distintas funciones del SIT, resulta excesivamente complejo tener que arrancar el sistema completo y realizar todo el procedimiento de entrenamiento para ve-

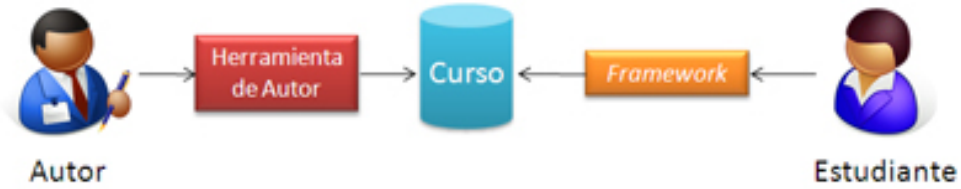

Figura 8.1: Herramienta de Autor

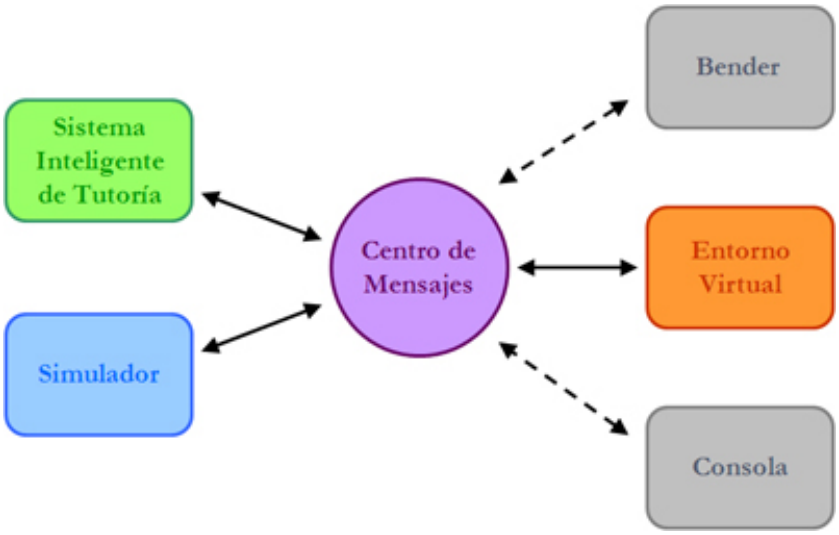

Figura 8.2: Clientes utilizados para la realización de pruebas

rificar una funcionalidad que quizás no aparece hasta pasado un determinado tiempo.

Para facilitar esta tarea, se han desarrollado dos pequeños clientes programables que sustituyen al Entorno Virtual y que funcionan de manera autónoma, lo que ha permitido agilizar la realización de dichas pruebas (ver Figura 8.2).

Adicionalmente, la utilización de estos sustitutos ha supuesto una evidencia más de la flexibilidad que proporciona el Centro de Mensajes al sistema, ya que han podido utilizarse los dos clientes de manera indistinta sin necesidad de realizar ninguna modificación en el resto de aplicaciones.

En cuanto a la implementación del SIT, tal como queda recogido en las recomendaciones metodológicas, se ha realizado en pequeños incrementos que se han iniciado con el Agente de Comunicación Global y el de Tutoría, para verse después complementados por un conjunto de agentes sin funcionalidad. A medida que se ha hecho necesario, se ha ido dotando a estos agentes de comportamientos y servicios, inicialmente vacíos y posteriormente con mayor funcionalidad.

Este método de desarrollo ha servido también para comprobar el funcionamiento del sistema de suscripciones, la utilización de servicios y, sobre todo, la manera de incluir nuevos agentes en el sistema, bien sea proporcionando nuevos servicios que han utilizado otros agentes, como ha sido el caso del agente de planificación, bien utilizando servicios existentes

para llevar a cabo sus propias responsabilidades, como ha sido el caso de la inclusión de un agente de modelado de estudiante sencillo.

Así pues, la implementación de este sistema ha servido para poner a prueba los mecanismos de modificabilidad incluidos en la arquitectura, con resultados, por el momento, bastante satisfactorios.

8.4. Conclusiones

Como colofón a los resultados obtenidos con la realización de esta tesis doctoral se puede concluir que se han resuelto satisfactoriamente los objetivos planteados al inicio del trabajo: se ha definido una arquitectura software basada en agentes para Entornos Virtuales de Entrenamiento con Tutoría Inteligente, se ha elaborado un conjunto de recomendaciones metodológicas que guíen su utilización y modificación y se ha implementado un sistema que ha puesto de manifiesto la validez no sólo de la propuesta de arquitectura sino también de las recomendaciones metodológicas.

A la vista de lo expuesto, puede concluirse que el presente trabajo ha cumplido con los objetivos planteados inicialmente, ha contribuido a la madurez de los Sistemas Inteligentes de Tutoría, a la de los sistemas basados en agentes y a la de los Entornos Virtuales de Entrenamiento, y lo ha hecho aportando ventajas significativas respecto a los sistemas analizados en el estado de la cuestión.

Capítulo 9

Líneas de Trabajo Futuro

"Cómo voy a creer que el horizonte es la frontera"
— *Utopías*, Mario Benedetti

9.1. Introducción

En muchos sentidos, la realización del presente trabajo de tesis doctoral ha supuesto para el autor un reto considerable, no sólo por lo que supone llevarlo a cabo, sino también por el esfuerzo que requiere no considerar las nuevas vías que van definiéndose a medida que se avanza en la resolución del problema y ceñirse al planteamiento inicial.

En efecto, son numerosas las nuevas líneas de investigación que se van perfilando. De éstas se han considerado las que a continuación se describen por ser las que mayor interés han despertado en el autor. Algunas son líneas de investigación abiertas por otros investigadores con los que se colabora, como es el caso de la inclusión de ciertas funcionalidades en la arquitectura, que se quieren probar en el sistema implementado con la arquitectura propuesta. Por tratarse de características surgidas en el proceso de evaluación de la arquitectura de la solución, se dispone ya de información acerca de la manera de llevar a cabo la integración y sobre el efecto que ésta puede tener sobre la arquitectura del sistema.

Hay otras que están más relacionadas con los intereses del autor y la curiosidad surgida de la revisión de algunos de los trabajos citados en la bibliografía.

De este modo, las siguientes secciones describen las líneas de trabajo futuro que, a criterio del autor, resultan más prometedoras, ya sea por interés personal o por la posibilidad de llevarlas a cabo en un futuro cercano, así como por su relevancia para el tipo de aplicación considerada.

9.2. Distintas Aproximaciones al Diseño de la Arquitectura

9.2.1. Arquitecturas Orientadas a Servicios

Aunque el planteamiento inicial del diseño de la arquitectura no buscaba una solución basada en servicios, lo cierto es que los mecanismos arquitectónicos utilizados en el diseño para obtener modificabilidad han dado como resultado una arquitectura que presenta similitudes con las Arquitecturas Orientadas a Servicios (SOA, *Service Oriented Architecture*).

Una de las razones para no haber utilizado plenamente este tipo de arquitectura es que el concepto de arquitectura orientada a servicios ha aparecido tradicionalmente asociado a aplicaciones web, a través del uso de servicios web, y algunas de las definiciones de SOA así lo ponen de manifiesto, como la que da el autor de (Erl, 2005):

> *Contemporary SOA represents an open, extensible, federated, composable architecture that promotes service-orientation and is comprised of autonomous, QoS-capable, vendor-diverse, interoperable, discoverable, and potentially reusable services, implemented as web services.*

Otra de las razones es el hecho de que el concepto de SOA se ha asociado más a determinado tipo de aplicaciones, de ámbito empresarial, que actualmente es más proclive al uso de tecnologías web que a los agentes, como lo demuestra, por ejemplo, la definición proporcionada por (Josuttis, 2007):

> *SOA is an architectural paradigm for dealing with business processes distributed over a large landscape of existing and new heterogeneous systems that are under the control of different owners.*

Sin embargo, poco a poco se está desligando el concepto de SOA tanto del ámbito empresarial como de los servicios WEB, y ya existen trabajos que cuentan experiencias en la aplicación de SOA en otros entornos, y en concreto en conjunción con agentes (Akbari y Karbalaee, 2005; Xiang, 2007; Gasmelseid, 2007). De hecho, a comienzos de Julio de 2008 se ha anunciado la colaboración de FIPA y OMG para avanzar hacia la estandarización de los agentes a través de UML, lo que incluye la elaboración de un estándar de SOA que introduce a los agentes en esta tecnología (SOA-Pro) (OMG, 2008).

Entre las características que se le atribuyen a las arquitecturas orientadas a servicios se encuentran (Erl, 2005; Erl, 2008):

- Los servicios se pueden reutilizar: aunque no exista una necesidad inmediata, los servicios se diseñan para ser reutilizables.
- Los servicios comparten un contrato formal: para que dos servicios puedan interactuar, sólo necesitan conocer un contrato que describa cada servicio y los requisitos de intercambio de información.
- Los servicios están poco acoplados: los servicios deben diseñarse sin que existan dependencias fuertes respecto a otros servicios.
- Los servicios abstraen la implementación: la única parte visible de un servicio es su interfaz, a través del contrato.
- Los servicios se pueden componer: esto permite representar los servicios con distinto nivel de detalle y facilita la reutilización.
- Los servicios son autónomos: la lógica de un servicio reside dentro de unos límites que están bajo el control de ese servicio.
- Los servicios no tienen estado: esto permite mantener la autonomía y el bajo acoplamiento de los servicios
- Los servicios se pueden descubrir: esta característica facilita su reutilización y composición.

A ojos del autor de la presente tesis, estos ocho puntos no deben entenderse como una característica inherente a las arquitecturas orientadas a servicios, sino como una guía de criterios a seguir al diseñar la arquitectura.

Los servicios utilizados en la arquitectura que se propone como solución de este trabajo cumplen de manera bastante fiel estos objetivos. Si a ello se le une que se propone este tipo de arquitectura como solución para interconectar componentes heterogéneos de distintos fabricantes (Josuttis, 2007), lo que constituye uno de los objetivos de este trabajo, parece que la arquitectura propuesta se adapta bastante bien a la definición de Arquitectura Orientada a Servicios.

Por ello, se propone como línea de trabajo futuro la exploración de los métodos y técnicas propuestos en este área para comprobar si pueden ofrecer una solución válida para diseñar sistemas como el que ha sido objeto del presente trabajo.

9.2.2. Distribución en un Grid

Uno de los problemas que ha ocupado desde hace años a la comunidad de Entornos Virtuales es la escalabilidad de las aplicaciones, ya que tienden a consumir muchos recursos cuando el número de usuarios se incrementa. Se han propuesto numerosas soluciones que abarcan desde la manera de modelar los entornos a los protocolos que soportan la comunicación en red de estas aplicaciones (Greenhalgh y Benford, 1995; Barrus et al., 1996; Dahmann, 1997; Das et al., 1997; Lèty et al., 2004).

En el caso de los Entornos Virtuales de Entrenamiento, el problema se acentúa si se tiene en cuenta que las tareas que implica la tutoría pueden ser computacionalmente costosas, como es el caso de la planificación, el cálculo de trayectorias o los razonamientos sobre ontologías para responder las preguntas de los estudiantes.

Una de las tendencias actuales para mejorar la eficiencia de las aplicaciones se basa en el uso de la computación con *grids*. Un *grid* es una infraestructura que permite compartir recursos y resolver problemas de forma coordinada entre organizaciones virtuales dinámicas (Foster y Kesselman, 2004).

Los *grids* han evolucionado desde los sistemas iniciales, que se basaban en estándares *de facto* a través del uso del *Globus Toolkit*, a los sistemas actuales, que usan estándares basados en servicios web y en la *Open Grid Services Architecture* (OGSA) (Foster et al., 2002).

En (Foster et al., 2004) los autores trazan las similitudes entre la computación *grid* y los sistemas multiagente, y sugieren cómo los sistemas basados en agentes y la computación *grid* pueden beneficiarse de su integración mutua.

En concreto, señalan el hecho de que la investigación con *grids* se ha centrado principalmente en temas relativos a la infraestructura: cómo organizar comunidades, cómo identificar nuevos miembros, cómo manejar la seguridad o cómo implementar las acciones dentro de una organización. Sin embargo, se ha dejado de lado la manera de utilizar esta tecnología para crear aplicaciones de gran tamaño y comportamiento estable.

Por el contrario, en el área de investigación relacionada con los agentes se ha prestado mayor atención a la creación de organizaciones abiertas y flexibles en las que nuevos comportamientos emergen de la interacción entre sus miembros. Sin embargo, no se han dedicado demasiados esfuerzos a idear formas de llevar estos modelos a la práctica en entornos distribuidos reales.

Más allá de sugerir que las aplicaciones basadas en agentes se ejecuten sobre sistemas *grid*, los autores de (Foster et al., 2004) sugieren fundir ambas tecnologías. Así, los agentes pueden usar los mecanismos existentes en

computación *grid* para, por ejemplo, descubrir servicios, y que los servicios *grid* recurran a los mecanismos de negociación de los agentes para distribuir los recursos en el mismo.

Avanzando en esa línea, los autores de (Zhang y Lin, 2007) proponen un mecanismo para distribuir un Entorno Virtual de gran tamaño en una organización basada en computación grid a través de su descomposición en un grupo de agentes móviles. Cada uno de los agentes es responsable de una tarea del sistema, como el control de la consistencia de los datos o el envío del nuevo estado de la escena. Estos agentes pueden migrar o clonarse de manera dinámica, ejecutándose no sólo en los servidores iniciales, sino también en las máquinas de los clientes que se van uniendo al sistema.

Por otra parte, los autores de (Jonquet et al., 2008) siguen la línea iniciada por (Foster et al., 2004) y proponen un lenguaje basado en ontologías, AGIL (*Agent-Grid Integration Language*), para realizar de manera efectiva la integración entre agentes y sistemas grid.

Como línea de trabajo futuro, se propone la exploración de estas líneas de acción para distribuir los agentes entre diferentes recursos hardware con objeto de proporcionar soporte al entrenamiento de grupos numerosos de estudiantes.

9.2.3. Diseño Alternativo de la Tutoría

La exploración de esta línea de investigación surge a raíz de dos características que aparecen en el presente trabajo.

La primera de ellas es el hecho, mencionado en repetidas ocasiones, de que existe cierta dependencia entre la estrategia de tutoría utilizada y la manera en que se realiza el modelado del estudiante, de forma que la sustitución de cualquiera de los dos requiere que el elemento introducido en el sistema sea compatible, a nivel pedagógico, con el elemento que se mantiene en él.

La segunda es la manera en que se realiza el modelado del estudiante en el sistema I-Help (Vassileva et al., 2003), descrito en el capítulo 2. Esta aplicación es un sistema multiagente que implementa una ayuda entre estudiantes. Los agentes facilitan la colaboración entre estudiantes, para lo que los distintos tipos de agentes mantienen información acerca del estudiante al que representan y de los estudiantes con los que interactúa.

La particularidad de este sistema es que no se mantiene un modelo único y centralizado para cada estudiante, sino que cada agente realiza y mantiene la parte del modelo del estudiante que le interesa. Así, el modelo de un estudiante puede estar repartido entre varios agentes, y las distintas piezas no tienen por qué ser consistentes entre sí, ya que reflejan distintas visiones sobre un mismo estudiante en distintas situaciones.

En el caso que nos ocupa, no resulta muy habitual que se opte por cambiar de estrategia de tutoría una vez que se lleva algún tiempo utilizando otra. Es posible que esto se deba a que por el momento no resulta demasiado fácil realizar dicha sustitución. Sí resulta más habitual que se intenten añadir nuevas funcionalidades a las que ya ofrece la estrategia de tutoría, lo que implica modificar la estrategia de tutoría y posiblemente el modelo del estudiante, para incluir información que deba ser usada por las nuevas características de la tutoría.

Lo que se propone es distribuir las responsabilidades de la estrategia de tutoría entre varios agentes, de manera que cada uno se encargue de una parte de la misma: gestionar las acciones de los estudiantes, las respuestas a sus preguntas o las pistas que se le ofrecen. Cada uno de los agentes, además, se encargaría de mantener la parte del modelo del estudiante que le hace falta para llevar a cabo sus responsabilidades. Para coordinar las decisiones de todos ellos se propone la utilización de otro agente que realice una planificación con las acciones que le proporcionen los demás, de forma similar a como se propone en otros trabajos (Lester et al., 1999a; Webber y Pesty, 2002; Broersen y Nijholt, 2002).

La ventaja que plantea este enfoque es que explota más la autonomía de los agentes que la solución actualmente propuesta, y facilita la introducción de nuevas características a la tutoría sin necesidad de modificar las ya existentes.

Por contra, plantea también dos inconvenientes. El primero de ellos es que algunas partes del modelo del estudiante estarán repetidas, lo que puede redundar es una mayor carga del sistema. El segundo es que, al no ser una forma muy extendida de construir estos sistemas, se pierde la posibilidad de intercambiar tanto la estrategia de tutoría como la forma de modelar al estudiante, al menos de manera sencilla.

9.2.4. Arquitecturas Software e Ingeniería de Software Orientada a Agentes

Una de las aportaciones más novedosas que ha proporcionado la realización del presente trabajo de tesis se encuentra en la aplicación de principios y técnicas de las arquitecturas software al diseño y evaluación de un sistema basado en agentes.

Existen numerosos trabajos que tratan el diseño arquitectónico de sistemas basados en agentes, pero casi todos ellos, como (Kolp et al., 2002; Bastos y Castro, 2005; Kolp et al., 2006), se basan de una u otra forma en el uso de la teoría de organizaciones (Weiss, 1999). Con esta teoría se han

elaborado tanto arquitecturas basadas en agentes como patrones de diseño basados en agentes (Kolp et al., 2003).

De la misma manera, muchas de las metodologías orientadas a agentes descritas en el capítulo 2 se basan de una u otra forma en el uso de la ingeniería de las vocales, que centra el diseño de la arquitectura del sistema el uso de organizaciones.

Sin embargo, como se ha podido ver, el concepto de organización no es suficiente cuando se desea prestar atención a los atributos de calidad de la arquitectura, además de a la funcionalidad, por lo que es necesario prestar atención a estos aspectos cuando se diseñan sistemas basados en agentes.

Esta necesidad ya se apunta en (Weyns et al., 2006), donde proponen utilizar enfoques más cercanos a la ingeniería de software tradicional para el diseño de sistemas basados en agentes. De igual manera, en (Kolp et al., 2006) ya se apuntan algunos de los atributos de calidad de los que se puede dotar a los sistemas basados en agentes, aunque no se apunta a la manera de conseguirlos dentro del contexto del diseño basado en organizaciones.

Por este motivo, se propone como línea de trabajo futuro la investigación sobre cómo introducir los métodos y técnicas del área de las arquitecturas software dentro del diseño arquitectónico mediante organizaciones de los sistemas basados en agentes.

En concreto, se propone la introducción en la estructura de la organización de mecanismos arquitectónicos orientados a conseguir unos determinados atributos de calidad. Esto permitirá seguir realizando diseños basados en organizaciones, a la vez que se se consideran de manera explícita los mecanismos arquitectónicos que conducen a la obtención de los atributos de calidad deseados.

9.3. Extensiones de la Arquitectura

9.3.1. Compañeros de Equipo Manejados por Agentes

Uno de los aspectos que puede favorecer el uso de Entornos Virtuales de Entrenamiento se basa en la posibilidad de entrenar a grupos de estudiantes para llevar a cabo tareas complejas que impliquen coordinación entre ellos.

Sin embargo, como cualquier otra labor que conlleve la participación de un grupo de personas, puede resultar difícil reunir a todas las personas que necesitan recibir ese entrenamiento.

Lo habitual en estos casos es esperar a que todos los miembros del grupo estén disponibles, provocando una espera larga para algunos de los miembros, o forzar a todos los integrantes del grupo a tomar parte en el entrenamiento

en un momento determinado, con los consiguientes trastornos que esto pueda causar.

Cualquiera de estas dos situaciones puede originar efectos negativos sobre el entrenamiento, por lo que una solución para facilitarlo consiste en integrar en el grupo a miembros controlados por agentes, lo que incluso posibilita que un único miembro del grupo se entrene en solitario para mejorar sus habilidades sin necesidad de involucrar a nadie más.

Ya existen algunos trabajos en los que se ha probado este tipo de solución (Rickel y Johnson, 2002; Traum et al., 2003; Chang et al., 2005), si bien la literatura referenciada pone de manifiesto que no se han realizado muchas iniciativas en este área, las soluciones aún son simples y presentan problemas relativos a la planificación y a la interacción agente-humano. También se han encontrado soluciones fuera de los Entornos Virtuales de Entrenamiento (Grosz y Kraus, 1996), pero se debe probar su validez en el área de la que se ocupa este trabajo.

Se propone como línea de trabajo futuro el estudio de una manera de representar las actividades que permita la planificación de tareas que deben llevar a cabo varios participantes, lo que permitirá formar equipos mixtos de personas y agentes que colaboren en una sesión de entrenamiento.

9.3.2. Integración de una Arquitectura Cognitiva

Como se describió en el capítulo de Estado de la Cuestión, una de las características que se requieren de los agentes virtuales que habitan un EV es que posean atributos similares a los humanos, de forma que mejoren la percepción de los usuarios respecto a los agentes y aumenten su motivación para utilizar la aplicación. Así, se puede conseguir que los agentes actúen de distinta manera en función de su estado interno, y podrán hacer uso de comunicación no verbal a través de expresiones faciales, corporales y de la entonación que impriman a sus frases.

Uno de los integrantes del grupo en el que se ha realizado la investigación descrita en el presente trabajo ha desarrollado una arquitectura cognitiva para agentes que da cabida, entre otros, al uso de rasgos de personalidad y emociones, lo que permite dotar a los agentes de comportamientos más similares a los humanos.

Esta arquitectura, llamada *Cognitiva* (Imbert, 2005), está estructurada en tres niveles, Reactivo, Deliberativo y Social, que reflejan tres facetas distintas del comportamiento humano.

Por un lado, el nivel reactivo da cabida a la parte del comportamiento más influenciado por las emociones, en forma de comportamientos reflejos y reacciones conscientes. Por otro, el nivel deliberativo contempla comporta-

mientos manejados por decisiones orientadas a alcanzar metas a través de planes. Finalmente, el nivel social da cabida a las influencias causadas por la relación del agente con otros agentes y humanos.

La utilización de esta arquitectura cognitiva se puede realizar con tres objetivos. El primero de ellos consiste en dotar al tutor de un modelo de personalidad que le proporcione un comportamiento más creíble de cara al alumno y que, además, pueda adaptarse a las características del alumno para proporcionarle una tutoría mejor adaptada a su forma de aprendizaje.

El segundo consiste en dotar de rasgos de personalidad a otros agentes con los que se deba interactuar, ya sean compañeros de equipo en un entrenamiento o simples personajes con los que interactuar en momentos puntuales. El hecho de que posean rasgos de personalidad y emociones también puede mejorar la experiencia del usuario dentro del EV y, por tanto, proporcionará beneficios al proceso de entrenamiento.

El tercer objetivo, quizá más complejo que los anteriores, consiste en utilizar esta arquitectura para realizar un modelado cognitivo de los estudiantes, lo que puede ayudar a predecir su reacción en una situación determinada y, además, permitirá adaptar aún mejor la tutoría a sus características.

Se proponen como líneas de trabajo futuro la integración de la arquitectura *Cognitiva* en cada una de las tres modalidades descritas.

9.3.3. Integración de Mecanismos de Percepción

El mismo objetivo que en el apartado anterior, dotar a los personajes presentes en el EV de un comportamiento similar al humano, se persigue con la propuesta de integrar mecanismos de percepción en los agentes virtuales que pueblen el EV.

Una de las maneras de conseguir que un agente presente un comportamiento similar al de un humano consiste en *limitar* sus capacidades, de manera que en lugar de ser consciente de todo lo que pasa en el entorno, viendo y oyendo todo lo que sucede, presente un comportamiento más parecido al de los humanos, de manera que sólo vea y oiga los estímulos que se encuentren lo suficientemente cerca y no estén ocultos por algún otro elemento.

En el ámbito del grupo de investigación en el que se ha desarrollado el presente trabajo, otro investigador elaboró un modelo de percepción para agentes, similar al humano, que incorpora elementos de percepción visual y auditiva (Herrero, 2003).

La percepción visual se articula en torno al uso de tres elementos principales, a los que se denomina *focus*, *nimbus* y *claridad de percepción*.

El *focus* es un doble cono que representa el campo de visión del agente, con un zona interna en la que la visión es más nítida y una zona periférica donde es más borrosa. El *nimbus*, por su parte, es una esfera que rodea a un objeto y que representa la zona del espacio en la que ese objeto puede ser percibido.

Un agente podrá ver un objeto si existe una intersección entre el *focus* del agente y el *nimbus* del objeto, y la claridad de percepción dará una medida de la calidad de dicha percepción, teniendo en cuenta la distancia y la zona del focus con la que se produce la intersección.

Por su parte, la percepción auditiva modela tanto la ubicación de la fuente de un sonido como la atenuación del volumen debida a la distancia y a los obstáculos.

Este mecanismo se ha utilizado para modelar enemigos en aplicaciones de entrenamiento militar (Herrero y de Antonio, 2005), donde el realismo en los mecanismos de percepción de los enemigos aumenta notablemente tanto el realismo de la aplicación como los beneficios que proporciona a los usuarios.

En las aplicaciones de entrenamiento procedimental se aprecia una gran utilidad en el empleo de mecanismos de percepción visual. Así, por ejemplo, el tutor virtual puede seguir al estudiante en sus desplazamientos por el entorno, o decidir dónde colocarse al ofrecer una explicación para no bloquearle al alumno la visión del lugar u objeto al que se refiere dicha explicación.

La percepción auditiva puede utilizarse para dotar al tutor virtual de un mayor realismo a la hora de, por ejemplo, girarse hacia el alumno cuando éste le haga una pregunta, ya que la localización de la fuente del sonido le permitirá girarse o volver la cabeza hacia el lado correcto. Asimismo, en aquellos dominios de entrenamiento en los que los sonidos jueguen un papel importante, la capacidad de percepción auditiva será fundamental para que el tutor pueda hacer una tutoría de utilidad.

Se propone, por tanto, dotar a los agentes virtuales, tanto tutores como compañeros de equipo, de mecanismos de percepción para mejorar la experiencia de los usuarios en las sesiones de entrenamiento.

9.3.4. Interacción mediante Lenguaje Natural

Una de las maneras en que se puede lograr un aumento de la sensación de presencia del estudiante en el entorno es a través de la mejora de la interacción con el mismo.

Para ello, una de las soluciones a las que apuntan algunos autores consiste en dotar a los sistemas de interacción a través del uso de lenguaje natural. En concreto, esta es una alternativa a la que recurren los sistemas basados

en diálogos, como (Graesser et al., 2008), que inicialmente utilizaba una interacción textual para pasar después a una interfaz basada en el habla.

Sin embargo, como ya se ha apuntado anteriormente en esta tesis, estos sistemas parecen recibir buenas críticas por parte de usuarios noveles, pero no tanto por parte de usuarios avanzados. Esto es debido, por un lado, a lo complejo de modelar diálogos con humanos, y por otro, a la dificultad de reconocer el habla si no se utilizan unas estructuras gramaticales bastante rígidas.

Por lo tanto, lo que se plantea es es hacer uso únicamente de la parte generativa para ofrecer al estudiante una interacción más rica mediante el uso de estructuras narrativas medianamente complejas. Esto se plantea en tres posibles campos de aplicación.

El primero de ellos se centra en la generación de indicaciones que permitan a un estudiante encontrar un lugar del EV al que no sabe cómo llegar. Para ello, es necesario contar con un modelo del mundo rico en semántica, para manejar el mismo tipo de información que el estudiante. Además, hace falta poder elegir referencias que el estudiante pueda utilizar para orientarse, y que permitan dar instrucciones precisas acerca del camino a seguir. Finalmente, es necesario generar estas instrucciones de manera natural, en lugar de como una serie de instrucciones atómicas que pueden sonar forzadas. Aunque ya existen aplicaciones que realizan funciones similares (Calle et al., 2006), la generación de lenguaje aún se realiza de manera un poco rudimentaria, por lo que en muchos casos las indicaciones quedan poco naturales.

El segundo campo de aplicación posible se basa en la utilización del conocimiento que se posee sobre la actividad a resolver para generar respuestas ricas en contenido ante las preguntas realizadas por los estudiantes. Al disponer de un planificador jerárquico como JSHOP2, existe la posibilidad de descender por la jerarquía del plan para ofrecer a los estudiantes respuestas más detalladas a las preguntas relativas a la tarea a realizar. La dificultad en este caso reside en enlazar en una estructura narrativa compleja los diferentes pasos correspondientes a distintas acciones que el estudiante debe llevar a cabo. Esta dificultad hace que algunos sistemas actuales, como los descrito en (Rickel y Johnson, 2002; Gratch et al., 2002), opten por proporcionar las respuestas en base a textos prediseñados.

Finalmente, el tercer campo de aplicación se basa en la generación de resúmenes y explicaciones en base a las acciones realizadas por los alumnos y las que deberían haber llevado a cabo de acuerdo a lo planificado. Aunque existen trabajos a este respecto, como (McQuiggan et al., 2008), los resultados no parecen de momento muy prometedores, ya que parece que los beneficios son inferiores a los de los medios tradicionales. Sin embargo, un repaso a la literatura muestra que que esto puede ser debido a que el

sistema con el que se trabaja está orientado a la enseñanza en lugar de al entrenamiento, el cual suele ofrecer mejores resultados en líneas generales.

Por lo tanto, con las tres áreas de aplicación propuestas, se plantea seguir una línea de investigación que integre la generación de lenguaje natural con estructuras narrativas complejas junto con los Entornos Virtuales de Entrenamiento.

9.4. Reflexiones

La realización del presente trabajo de tesis ha supuesto una experiencia notablemente enriquecedora para el autor, tanto en lo que se refiere al desarrollo de la labor investigadora como a la escritura de la memoria de tesis doctoral, al proporcionar un medio para organizar ideas y pulir detalles.

Uno de los resultados más atractivos de esta tesis son las numerosas líneas de investigación que quedan abiertas para su exploración futura, permitiendo que este trabajo no sea un fin en sí mismo sino el punto de partida como investigador de su autor.

Algunas de las líneas propuestas están muy relacionadas entre sí y con el trabajo realizado, como puede ser la utilización de arquitecturas orientadas a servicios y la distribución del sistema en un *grid*. En opinión del autor, este es un aspecto importante ya que se cuenta con el firme apoyo de la investigación ya realizada, y además se centran los esfuerzos en unos aspectos concretos de la investigación, permitiendo así disponer de objetivos claros de cara a proseguir la labor investigadora.

Quedan también puertas abiertas a otras líneas que, por sí solas, pueden dotar al trabajo actual de características atractivas y útiles, además de posibilitar la colaboración con otros investigadores.

Aunque existen muchos otros caminos susceptibles de ser explorados, los que se han presentado anteriormente son los que se consideran más prometedores, tanto por la trayectoria y las inquietudes del autor como por los intereses que suscitan actualmente en las áreas de investigación relacionadas.

Parte IV

Apéndices

Apéndice A

Sobre la Implementación de la Arquitectura

"A fuerza de construir bien, se llega a buen arquitecto"
— Aristóteles

A.1. Introducción

Una de las aportaciones que se ha realizado con la elaboración del presente trabajo de tesis es la implementación de una infraestructura que, siguiendo las directrices marcadas por la arquitectura propuesta, permita desarrollar Entornos Virtuales de Entrenamiento en diferentes dominios.

El propósito de esta implementación, además de probar la validez de la arquitectura propuesta, consiste en facilitar su adopción por parte de otros grupos de investigación. Esto, a su vez, proporcionará información relevante que permita hacer evolucionar el sistema en una dirección que resulte de utilidad a la comunidad relacionada con los sistemas objeto de estudio.

En las siguientes secciones se proporciona información relativa a la implementación de distintas partes de la arquitectura. Aunque no se dan detalles de la implementación completa, se han seleccionado aquellos elementos implementados que resultan más significativos para la funcionalidad del sistema. En concreto, se muestra una descripción de la implementación del Entorno Virtual, el Centro de Mensajes, el Agente de Comunicación y el Agente de Tutoría.

Estas implementaciones se han llevado a cabo en el marco de la realización de Proyectos de Fin de Carrera de alumnos del Laboratorio Decoroso Crespo (Núñez, 2008; Rodríguez, 2008; Sánchez, 2006).

A.2. Implementación del Entorno Virtual

A.2.1. El Sistema Gestor de Entidades

Actualmente, la mayoría de simulaciones, sistemas de realidad virtual y videojuegos (que, aunque parezcan mucho menos "académicos", en muchos aspectos se encuentran mucho más avanzados) son diseñados y programados desde cero (o casi desde cero) ajustándose a los requisitos del problema que desean resolver.

La proliferación de componentes, bibliotecas y sistemas de desarrollo para la programación de este tipo de sistemas hace pensar que en algún momento se podrá desarrollar sistemas de realidad virtual tal y como se desarrollan las aplicaciones de escritorio.

La diferencia entre las aplicaciones de escritorio y los sistemas de realidad virtual es que los ingenieros de las primeras aún tienen que desarrollar lo que se denominan las "reglas de negocio" de cada una de las aplicaciones, mientras que partes como las capas de presentación, persistencia y comunicación entre procesos están disponibles para ellos como bibliotecas (del propio sistema operativo o bien externas a él). Los ingenieros que construyen los sistemas de realidad virtual tienen que desarrollar su aplicación en todos sus aspectos, lo cual es poco productivo.

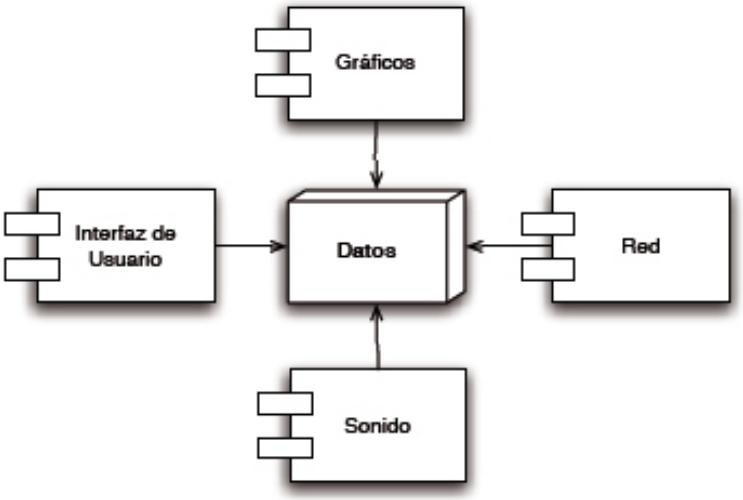

Figura A.1: Diseño general del EV

La intención principal de esta arquitectura es intentar que el uso de sistemas de desarrollo, bibliotecas y componentes sea lo más sencillo posible,

que el cambio de uno de estos componentes no implique más que un pequeño "dolor de cabeza" para la parte que trata con esos componentes (es decir, aislar los cambios de un componente de los demás componentes), y que esta arquitectura sea lo más abierta posible para resolver el mayor número de problemas que se presenten a los sistemas de realidad virtual.

Después de evaluar las diferentes posibilidades, la arquitectura que más se ajusta a las características deseadas según nuestros criterios es una arquitectura fundamentalmente centrada en los datos como la que se puede ver en la Figura A.1.

Como se puede ver esta arquitectura refleja muchos de nuestros deseos y contempla muchos de los aspectos comentados durante el análisis realizado anteriormente: los subsistemas no contienen dependencias entre ellos, pero todos ellos actuarán según unos datos comunes. Esta independencia de subsistemas significa que la sustitución de los componentes debería ser una tarea sencilla.

Pero podemos observar un problema obvio en esta arquitectura: ¿están los diferentes subsistemas sincronizados con los datos? Para responder a esta pregunta hay que estudiar si esta arquitectura es capaz de funcionar de forma que los cambios en los datos sean conocidos por los subsistemas cuando estos los necesiten.

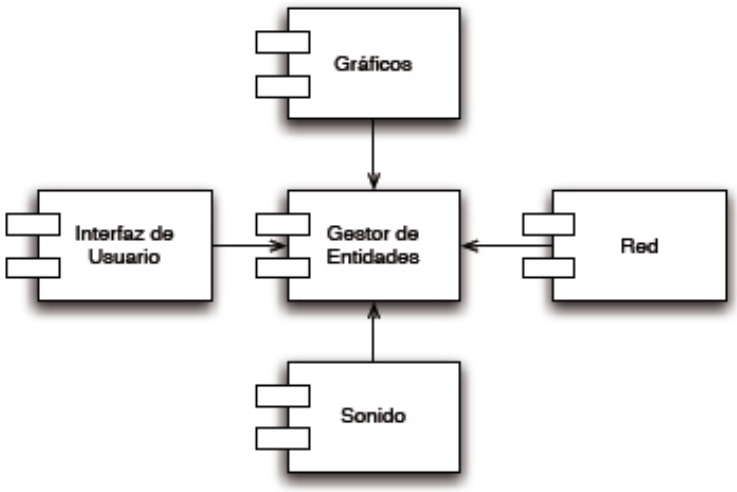

Figura A.2: Componentes del EV

La opción de utilizar el mecanismo del repositorio de datos activo es la que menos carga debería suponer a los diferentes subsistemas y como complemento se han implementado sistemas de publicación–subscripción

a nivel de objeto del repositorio, de forma que se reduzca el número de comunicaciones innecesarias entre los subsistemas y el repositorio (también se han implementado el sistema de publicación–subscripción a nivel de repositorio, de forma que los subsistemas puedan elegir cuál de las dos aproximaciones quieren utilizar).

Se puede ver que la idea inicial de un "simple" repositorio de objetos se ha ido complementando con sistemas de publicación–subscripción, y en previsión de que las responsabilidades del repositorio de objetos aumentarán en el futuro se decidió abandonar la denominación "repositorio de objetos" para que no sonara extraña ante los posibles cambios. Al final se comprobó que fue la decisión correcta, ya que el sistema de gestión empezó a adquirir otras responsabilidades como la gestión de subsistemas o la gestión de mensajes. El resultado final se puede observar en la Figura A.2.

A.2.2. El Simulador

El simulador es el encargado de controlar el funcionamiento del sistema de gestión de entidades, ya que éste, por si solo, no hace mucho más que existir y guardar entidades. Sólo cuando contamos con un "director de orquesta" (en nuestro caso el simulador) el sistema de gestión de entidades se vuelve útil, invocando a los diferentes submódulos y actuando en consecuencia a sus acciones.

En nuestra implementación, el simulador es básicamente un autómata finito determinista que define una máquina de estados y una serie de eventos que hacen avanzar el estado de la máquina. Los eventos pueden ser acciones del usuario (pinchar con el ratón en un botón, por ejemplo), mensajes llegados por la red, u otras clases de mensajes que sean enviados a través del sistema de gestión de entidades y que el simulador decida responder. Para implementar esta arquitectura se ha optado por una solución intermedia entre el patrón *Strategy* y el patrón *State*, como se puede ver en la Figura A.3.

El patrón *Strategy* nos permite definir una familia de algoritmos, encapsular cada uno de ellos, y hacer posible que sean intercambiables. El patrón *Strategy* permite que el algoritmo pueda ser modificado independientemente de los "clientes" que lo utilizan. El patrón *Strategy* es útil en situaciones donde es necesario intercambiar dinámicamente los algoritmos que han de ser utilizados en la aplicación.

El patrón *State*, mientras tanto, permite a un objeto alterar su comportamiento cuando su estado interno se modifica. El objeto parecerá que ha cambiando de clase, pero lo único que habrá cambiando habrá sido su estado interno. Muchos autores consideran al patrón *State* como un patrón *Strategy* encapsulado y "auto-modificable" (el patrón *Strategy* no

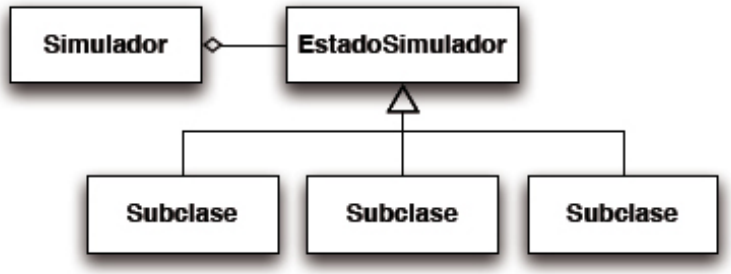

Figura A.3: Diseño del Simulador

está diseñado para modificar el comportamiento del objeto dinámicamente, aunque podría hacerlo).

En nuestro caso el patrón seguido no se ajusta completamente al diseño o los principios de ninguno de los dos: nuestro patrón, al igual que el patrón *State* es "auto-modificable", pero también es cierto que puede ser modificado externamente; nuestro patrón también es asíncrono (algo que no son ninguno de los dos patrones), porque el cambio de estado del simulador no se produce cuando el usuario lo requiere, sino cuando el simulador lo vea oportuno.

El hecho de que la modificación del estado sea asíncrona es necesario en un problema como al que nos enfrentamos, ya que la sustitución de un estado por otro no debería producirse en cualquier momento, pues el estado actual puede estar realizando tareas que no deberían ser interrumpidas. Por ello se diseñó el sistema de forma que se pidiera el cambio de estado, pero que fuera el simulador el que decidiera cuando su estado interno debería ser modificado. El cambio del estado está garantizado (y en circunstancias normales se producirá de forma casi instantánea), pero no está garantizado que todos los cambios de estados sean satisfechos.

Tal y como está diseñada la arquitectura, se pensó que mantener una cola de estados a los que cambiar sería poco eficiente y sobre todo innecesario: los estados en la cola durarían un único cuadro o tick de la simulación, haciendo que sus efectos fueran prácticamente invisibles a los usuarios.

Si entre la petición de un cambio de estado y su cambio efectivo se produce otra petición de cambio de estado, será esta última la que finalmente se utilice. Esto podría producir que algunos estados no fuesen utilizados, pero durante la construcción del software no han sucedido tales casos, y, de cualquier forma, tales casos son sencillos de evitar.

En nuestra arquitectura, el Simulador es un objeto con pocas responsabilidades. Por supuesto, es el encargado de mantener su estado y de modificarlo cuando se lo pidan, pero los objetos que realmente son los encargados de de-

finir el comportamiento de la aplicación son las diversas implementaciones de los estados del simulador.

El diagrama de estados del autómata finito determinista del simulador se puede ver en la Figura A.4. El flujo del simulador actualmente es bastante lineal, pero la arquitectura está preparada para implementar sobre ella flujos con decisiones, saltos de estados condicionales y demás necesidades que puedan tener los desarrolladores de herramientas. En la figura podemos ver cómo unas líneas estás marcadas claramente, mientras que otras utilizan un trazado rayado: las primeras representan el flujo "normal" del programa, mientras que las últimas representan opciones no muy usuales que puede tomar el usuario. En este caso, el usuario puede decidir cancelar el proceso de simulación en cualquier momento.

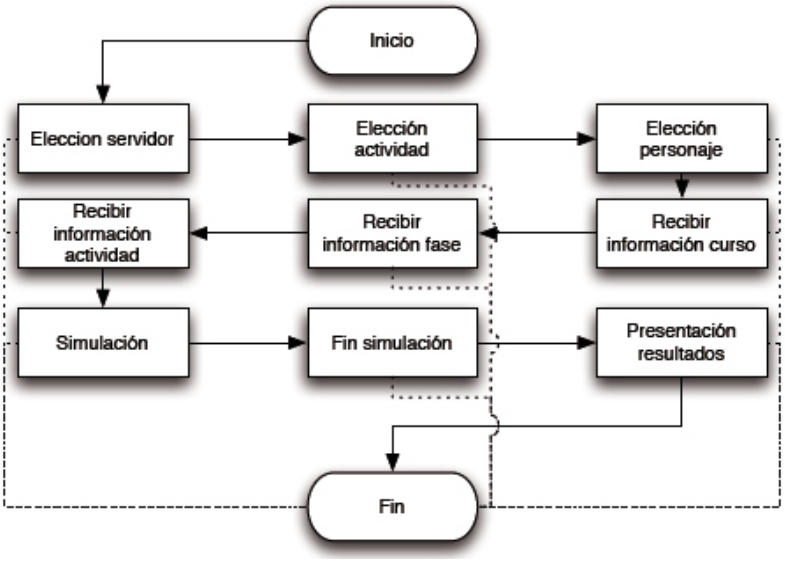

Figura A.4: Diagrama de estados del Simulador

A.2.3. Implementación de la Entidad

Dentro del contexto de este proyecto, una entidad es cualquier objeto que participe en la simulación. No es necesario que una entidad sea "representable" para el usuario (ya sea de forma visual o auditiva), simplemente que signifique algo en la simulación. Ejemplos de elementos que participen en la simulación, pero no son "representables" para el usuario, son una luz para el sistema gráfico, de la cual el usuario vería sus efectos, pero no a la luz misma; o una zona donde se aplique alguna ley física como una

zona de viento, invisible para el usuario, pero cuyos efectos deben tenerse en cuenta para la simulación.

La definición de entidad es muy abstracta para ajustarse a cualquier idea que pueden tener de entidad los diseñadores de la simulación. Como guía podemos decir que cualquier "cosa" que uno o más subsistemas deben gestionar durante la simulación debería ser una entidad.

Como se puede ver en la Figura A.5 las entidades tienen sus responsabilidades separadas en tres clases diferentes. La clase `IEntidad` es la interfaz que todas las entidades que pretendan utilizarse en el gestor de entidades deberían implementar. Se proporcionan además dos implementaciones concretas de la interfaz: `Entidad` y `EntidadModulo`. La primera es una implementación canónica de `IEntidad` que puede ser utilizada sin ningún problema en la mayoría de situaciones (de hecho, excepto en el caso de verdaderos problemas con las implementaciones canónicas, se recomienda su uso).

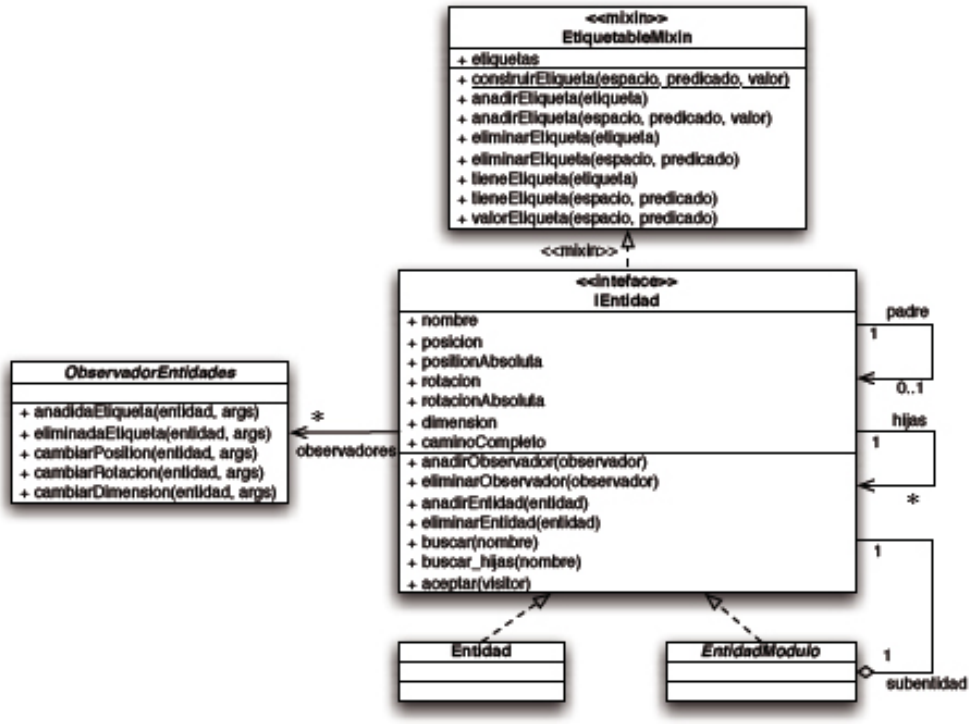

Figura A.5: Diseño de la Entidad

`EntidadModulo`, mientras tanto, es una versión de `IEntidad` que agrega a otra `IEntidad`. Esta clase, como su nombre indica, está orientada a utilizarse desde los módulos que hagan uso del gestor, pero no debería ser

utilizada directamente. Su propósito es servir como implementación base para los diferentes módulos, que deberán personalizar el funcionamiento de esta entidad con sus propias necesidades. Esta entidad permite la separación de los datos propios de cada módulo (que formarán parte de la clase derivada de `EntidadModulo`) y los datos generales de las entidades (que se almacenan en la `IEntidad` agregada).

Los módulos que utilicen sus propias versiones de `EntidadModulo` deben recordar que deben tratar con los datos comunes de `IEntidad` de forma especial. Por ejemplo, en el caso de un sistema gráfico en dos dimensiones en una rejilla, la representación de los campos x e y de `IEntidad` puede ser números enteros positivos, pero esta representación sólo puede ser visible en las interacciones hacia el subsistema, y no debe ser visible para el gestor u otros subsistemas, que sólo verán la visión general donde x e y son números en coma flotante según el estándar que esté definido para la simulación.

A.2.4. Implementación del Gestor de Entidades

Las entidades por si solas no son demasiado "funcionales". Es cierto que podemos organizarlas en un árbol y actuar sobre ellas (utilizando *Visitors* y observadores), pero sería difícil utilizarlas desde varios subsistemas al mismo tiempo, sobre todo si tenemos en cuenta que muchos de estos subsistemas ni siquiera conocerán la existencia del otro. Por ello necesitamos una serie de clases que tengan la responsabilidad de coordinar y regular el uso que hacen los subsistemas de las entidades, de forma que todos ellos realicen sus modificaciones en las entidades a través de métodos que permitan a los demás subsistemas conocer las modificaciones realizadas.

La principal de estas clases es la "jerarquía" de `IGestorEntidades`, que como se ve en la Figura A.6 es muy similar a "jerarquía" de `IEntidad`, no por una casualidad, sino porque una estructura similar facilita la comprensión de las responsabilidades y las funciones que tiene cada clase.

Las responsabilidades de los gestores, al igual que las responsabilidades de las entidades, está separadas en tres clases. La clase `IGestorEntidades` (equivalente a `IEntidad`) es la interfaz que los demás gestores deberían implementar. Junto con la interfaz se proporcionan una implementación canónica del gestor en `GestorEntidades` (similar a la idea de `Entidad`) y la implementación pensada para facilitar la creación de subsistemas específicos: `GestorEntidadesModulo` (relacionada con `EntidadModulo`).

En el caso de los gestores, debido a su forma de uso, no es necesario implementar un gestor por cada subsistema, y en el caso de ser necesario, no es obligatorio atenerse a las interfaces proporcionadas. La idea de las interfaz `IEntidad` está diseñada para que mediante composición y agregación

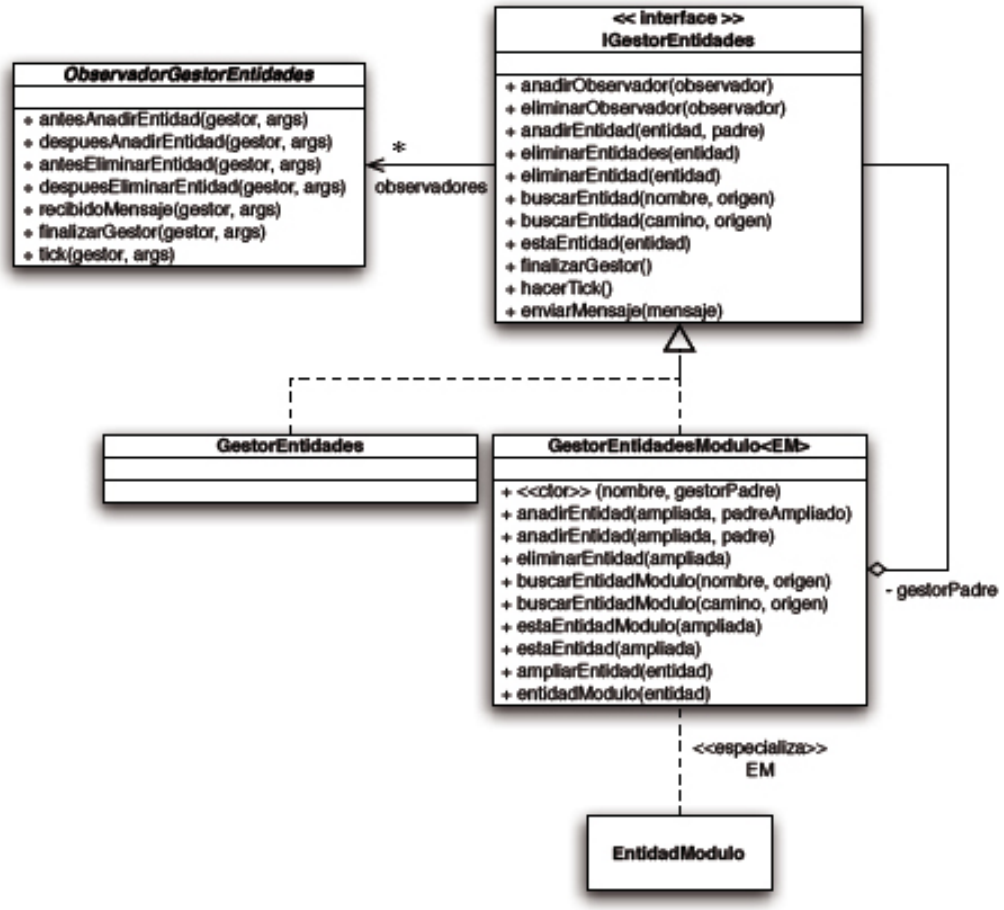

Figura A.6: Diseño del Gestor de Entidades

se vaya especializando el uso de una `IEntidad`, incluso contemplando la posibilidad de especializar una entidad ya especializada (siguiendo una suerte de patrón *Decorator*), para lo que es necesario forzar a que la primera especialización tenga el mismo interfaz que la `IEntidad`, de forma que la segunda especialización sea aplicable sobre ella.

En el caso de los gestores no parece que existan muchas situaciones en las que un gestor deba especializar a otro gestor ya especializado, por lo que, a pesar de que existe la arquitectura para ello, no debería seguirse como una regla para los subsistemas, sino más bien como una "directriz". De hecho, algunos de los subsistemas ya implementados, al considerar que la implementación de un gestor específico era innecesario, se comunican con el gestor "central" únicamente mediante observadores y visitors.

`GestorEntidades`, la implementación canónica de `IGestorEntidades`, es una implementación completa de un gestor de entidades, que es totalmente utilizable en la mayoría de simulaciones que se podrían presentar (por ahora no hemos encontrado una simulación que necesite suposiciones diferentes a las que realiza `GestorEntidades`). La implementación canónica, además, muestra como deben ser implementados los eventos que debe generar el gestor para informar a los observadores, según la especificación de `IGestorEntidades`. Al igual que en el caso de la implementación canónica de `IEntidad` es recomendable que, excepto en el caso de verdaderos problemas con ellas, se utilice la implementación canónica y se evite volver a implementar otro gestor desde el principio.

En el caso de `GestorEntidadesModulo`, a pesar de ser en cierto modo equivalente a `EntidadModulo`, difieren en su forma de implementarse. Mientras que no era posible utilizar directamente `EntidadModulo` y debíamos especializarlo implementado una nueva clase que heredase de `EntidadModulo`, la implementación de `GestorEntidadesModulo` permite que, gracias a las facilidades de meta-programación de C++ utilizando templates, una vez creada nuestra especialización de Entidad-Modulo dispongamos de un completo `GestorEntidadesModulo` especializado para nuestra nueva entidad sin necesidad de ninguna línea de código.

La implementación que proporciona `GestorEntidadesModulo` incluye una implementación completa de la interfaz de `IGestorEntidades` y además varios métodos especialmente adaptados para trabajar de forma diferente en el caso de utilizar instancias de `EntidadModulo` o de `IEntidad` (por ejemplo `anadirEntidad`).

También proporciona métodos muy útiles para el caso de que tengamos que añadir una nueva `EntidadModulo` basándonos en una `IEntidad` del gestor "central" y para interrogar al gestor "ampliado" sobre las instancias de `EntidadModulo` en él (`estaEntidadModulo` y `ampliarEntidad`). Finalmente la implementación de `GestorEntidadesModulo` se encarga de eliminar las entidades relacionadas con las entidades eliminadas en el gestor "central", de forma que los datos de ambos siempre estén sincronizados.

Por supuesto, si es necesario especializar el funcionamiento proporcionado por el template es posible utilizando los mecanismos clásicos de C++, pero al igual que en el caso de `EntidadModulo` se debe tener muy en cuenta que los demás subsistemas esperan que los gestores cumplan cierto "contrato" (la especificación de `IGestorEntidades`). En el caso de los gestores las partes importantes del "contrato" es la generación correcta de los eventos, ya que muchos subsistemas dependen de la correcta generación de los eventos para funcionar.

A.2.5. Mensajes entre Subsistemas

Hasta ahora se han mostrado varios sistemas de comunicación de datos y eventos entre subsistemas: los subsistemas pueden comunicarse con el sistema central mediante acceso directo a las propiedades de las entidades, pero estas propiedades son limitadas y están definidas desde el momento en que se diseñaron las clases. Añadir una nueva propiedad a una entidad exigiría modificar todo el sistema y los subsistemas. El sistema central se puede comunicar con los subsistemas mediante el mecanismo de observadores y protocolos de publicación–subscripción, de forma que el sistema central no tenga que conocer a cada uno de los subsistemas, pero de nuevo los eventos que son transmitidos por el sistema central están definidos en tiempo de diseño, a pesar de que ampliar el número de eventos generado únicamente afectaría al sistema central y al subsistema que requisiera el nuevo evento. Pero hasta ahora no hemos visto la posibilidad de comunicar unos subsistemas con otros.

El sistema de mensajes se encarga de esta tarea proporcionando un mecanismo de paso de mensajes síncrono entre todos los subsistemas que se hayan registrado como observadores de un gestor central. Los mensajes se reciben a través del método `enviarMensaje`, en cuyo argumento (`ArgumentosEnviarMensaje`) va contenido el mensaje en una instancia de la clase `Mensaje` (ver Figura A.7).

Un `Mensaje` tiene un nombre y una serie de argumentos (datos asociados que aportan más significado según la semántica del mensaje). La clase `Mensaje` proporciona métodos para manipular la lista de argumentos de un mensaje, y métodos para convertir un `Mensaje` en un formato textual denominado *JSON* ideal para almacenar en archivos de texto o enviar a través de la red.

Un mensaje, al incluir `EtiquetableMixin`, puede ser etiquetado, lo que añade otra dimensión de datos al mensaje, aunque esta última dimensión no forma parte de los datos que son transformados al formato *JSON*.

Los argumentos de un mensaje son subclases de la clase abstracta `ArgumentoMensaje`. La clase base establece que los argumentos deben tener un nombre y deben ser capaces de convertirse a sí mismos en formato *JSON*, de forma que el mensaje pueda delegar en ellos la conversión (la diferencia entre el método `toJson` y el método `toJsonValue` es que el primero proporciona una representación en formato cadena, mientras que el segundo proporciona una representación interna, más sencilla de manipular mediante programación). Las subclases de `ArgumentoMensaje` proporcionan un pequeño sistema de tipos que cubre muchas de las necesidades que se puedan tener al transmitir mensaje y sus datos asociados.

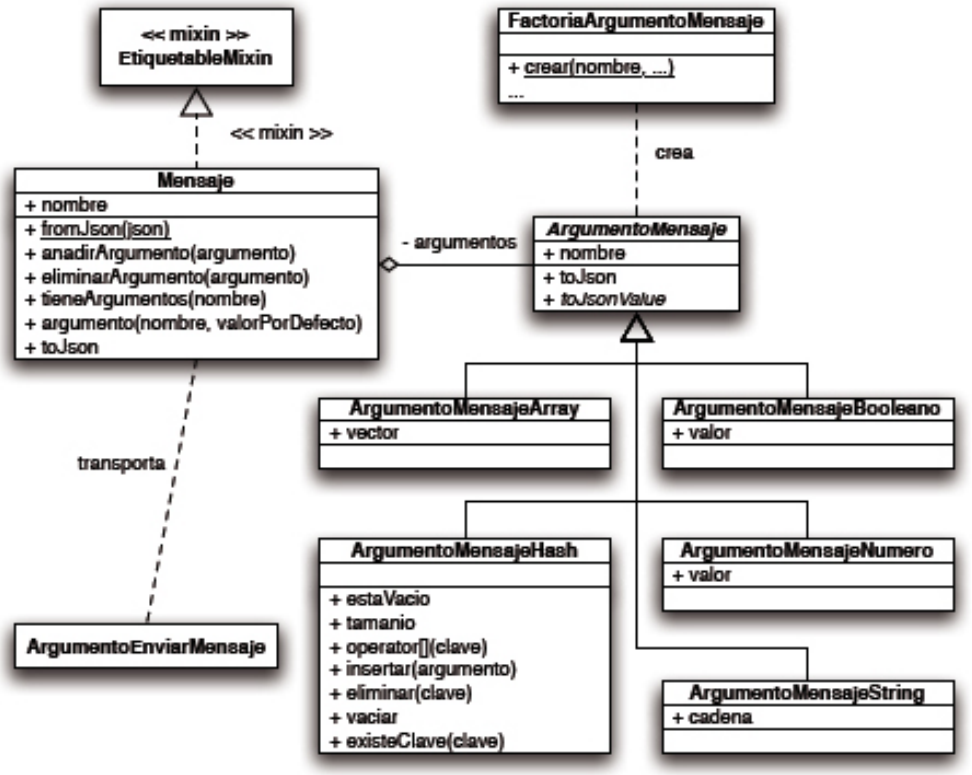

Figura A.7: Diseño de los mensajes

La creación de los diferentes tipos de argumentos se realiza a través de la clase `FactoriaArgumentoMensaje`, pero a pesar de su nombre no sigue el patrón *Factory*, si no que se trata de un objeto factoría, es decir un objeto que contiene funciones factoría, que crean objetos a petición del programador. En el caso de `FactoriaArgumentoMensaje` se dispone de un único método de clase (estático) denominado crear, que mediante los mecanismos de sobrecarga disponibles en C++, dependiendo de sus argumentos, creará una instancia determinada de `ArgumentoMensaje`, de forma que los programadores no tengan que preocuparse de realizar los pasos necesarios para crearlos.

A.2.6. Implementación del Simulador

Hasta ahora hemos descrito las partes del sistema y cómo los subsistemas pueden utilizarlas para implementar sus características específicas, se ha visto como interactúan las diferentes partes del sistema y como puede ser

ampliado para ajustarse a las necesidades de nuestra simulación. Pero falta la descripción de una de las partes: el encargado de iniciar los subsistemas, enlazarlos y añadir los datos necesarios para representar la simulación. En nuestra arquitectura esta tarea la cumple el Simulador y sus clases asociadas.

El `Simulador` tiene la tarea de director de orquesta: sitúa a sus instrumentista, les da ciertas instrucciones, y deja que cada uno toque su instrumento según esas instrucciones, interviniendo, de vez en cuando, para sincronizar el funcionamiento de cada uno de los instrumentistas. Pero, a pesar de la analogía, el `Simulador` no realiza todas las tareas del "director de orquesta", sino que delega muchas de ellas a otras clases, las cuales se pueden ver en la Figura A.8. Cada una de estas clases tiene una tarea bien definida y son utilizadas por el simulador en momentos específicos.

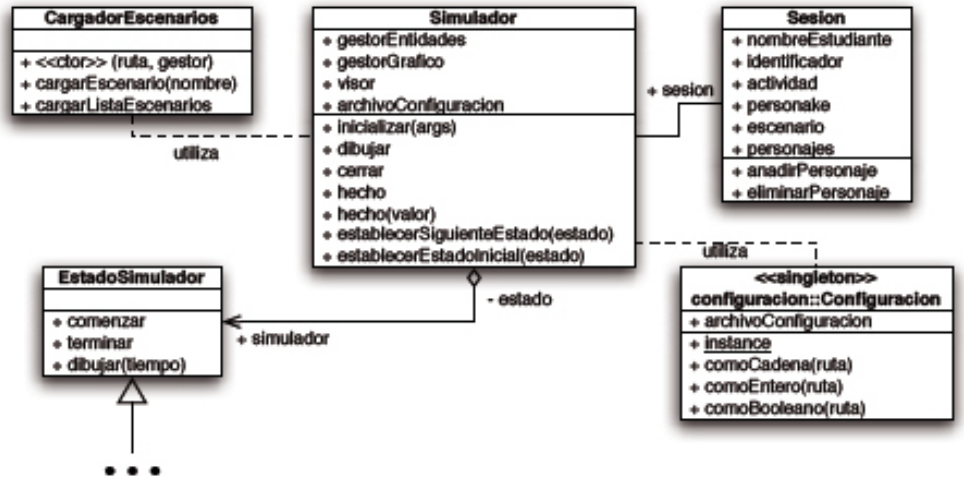

Figura A.8: Diseño del Simulador

Al `Simulador`, después de separar las diferentes tareas en diferentes clases, le quedan dos responsabilidades: mantener el estado de la simulación y hacerlo avanzar. Para ello, el simulador utiliza un concepto parecido al de una máquina de estados finita, aunque nunca se define una como tal.

Formalmente, una máquina de estados finita es un modelo de comportamiento compuesto de un número finito de estados, transiciones entre esos estados, y acciones. Una máquina de estados finita puede verse como un modelo abstracto de una máquina con una memoria interna primitiva.

Un estado almacena información del pasado, ya que refleja los cambios en la entrada del autómata desde el inicio hasta el presente. Una transición, mientras tanto, indica un cambio de estado, y se describe por una condición que debe cumplirse para que la transición pueda llevarse a cabo. Finalmente,

una acción es la descripción de una actividad que debe realizarse en un momento dado. Existen varios tipos de acciones: acciones de entrada a un estado, acciones de salida de un estado, acciones de reacción a la entrada del autómata, y acciones de transición.

En nuestra implementación se podría ver al `Simulador` como la máquina de estados finita y a las clases derivadas de `EstadoSimulador` como los estados. Sin embargo, nuestra implementación no define las transiciones de estado explícitamente, sino que están definidas por los propios estados del simulador, en reacción a las entradas del simulador y a su memoria interna.

El simulador delega en la clase `Sesion` el almacenamiento de algunos datos relativos a la simulación, pero no específicos a ninguno de los subsistemas, principalmente aquellos datos necesarios antes de que se haya creado realmente una simulación.

La clase `CargadorEscenarios` es la clase a la que el simulador delega las tareas de encontrar los archivos de definición de escenarios (los mismos archivos de definición de escenarios que se utilizan en el sistema inteligente de tutoría, para simplificar), y cargar los datos necesarios en el gestor de entidades designado.

Finalmente, la clase `Configuracion` permite acceder desde cualquier parte del sistema a los datos de la configuración global, principalmente datos necesarios para iniciar el funcionamiento de los diferentes subsistemas.

A.2.7. Bibliotecas de Terceras Partes Utilizadas

OpenSceneGraph es una biblioteca de gráficos tridimensionales de alto rendimiento de código abierto utilizada por desarrolladores de aplicaciones en campos como la simulación visual, juegos de ordenador, realidad virtual, visualización científica y modelado. Está escrita totalmente en C++ estándar y OpenGL, por lo que se ejecuta en gran variedad de sistemas operativos como Microsoft Windows, Mac OS X, Linux, IRIX, Solaris y FreeBSD.

Simple DirectMedia Layer (SDL) es una biblioteca multimedia de código abierto multiplatarforma escrita en C que crea una abstracción sobre los gráficos, sistema de sonido y sistemas de entrada de cada plataforma, permitiendo a los desarrolladores escribir aplicaciones multimedia (usualmente videojuegos) una sola vez y ejecutarlos en muchos sistemas operativos incluyendo Linux, Windows y Mac OS X. SDL tiene versiones para casi cualquier lenguaje de programación que se utiliza, desde los populares (C++, Perl, Python, Pascal, etcétera) a los menos conocidos (como Euphoria y Pliant).

VR Juggler es un conjunto de herramientas que escalan desde sistemas de escritorios como PC a sistemas complejos multipantalla que se ejecutan en clusters o en estaciones de trabajo de gama alta y superordenadores. La flexibilidad de VR Juggler permite que las aplicaciones se ejecuten en muchas configuraciones de sistemas de realidad virtual desde realidad virtual de escritorio a cascos de realidad virtual, dispositivos CAVE y PowerwallTM. La portabilidad del código de VR Juggler asegura que cualquier sistema operativo pueda ser utilizado. VR Juggler 2.0 funciona bajo IRIX, Linux, Windows, FreeBSD, Solaris y Mac OS X.

Crazy Eddie's GUI System (también conocida como CEGUI) es una biblioteca de código abierto que proporciona ventanas y controles a los motores y bibliotecas gráficas donde tales funcionalidades no están disponibles nativamente, o son insuficientes. La biblioteca está escrita utilizando C++ orientado a objetos, y está orientada a los desarrolladores de videojuegos que deberían gastar su tiempo en programar la lógica de la aplicación en vez de diseñando subsistemas de interfaces gráficas de usuario. CEGUI es la interfaz gráfica de usuario que se utiliza por defecto en el proyecto Ogre3D, pero también puede utilizarse directamente con DirectX 8.1 y 9 de Microsoft y OpenGL, así como con el motor Irrlicht.

Bibliotecas Boost para C++ son una colección de bibliotecas de código abierto que extienden las funcionalidades de C++. La mayoría de las bibliotecas están protegidas por la licencia Boost Software License, diseñada para permitir que Boost pueda ser utilizado tanto con proyectos de código abierto como con proyecto propietarios. Muchas de las bibliotecas de Boost han sido aceptadas para su incorporación al siguiente estándar de C++. Las bibliotecas están destinadas a un amplio rango de usuarios y aplicaciones de C++. Sus funciones van desde bibliotecas de propósito general como la biblioteca de punteros inteligentes, a abstracciones de los sistemas operativos como la biblioteca de acceso al sistema de ficheros, y a bibliotecas principalmente destinadas a otros desarrolladores de bibliotecas y usuarios avanzados de C++, como la biblioteca de meta-plantillas (*meta-templates*, en inglés).

A.3. Implementación de la Comunicación

El Centro de Mensajes es la entidad software por donde pasan todos los mensajes enviados por los diferentes subsistemas. La diversidad de componentes conectados al Centro de Mensajes de forma simultánea requiere un diseño capaz de gestionar múltiples conexiones sin bloquearlas entre sí. Es decir, el Centro de Mensajes debe comunicarse de forma concurrente con todos sus clientes, estableciendo en tiempo real los canales de comunicación necesarios. La solución a este problema, empleada en múltiples implementaciones de servidores que han de atender más de una conexión al mismo tiempo con sus clientes (un ejemplo de ello es el servidor http de Apache), es realizar un servidor multi-hilo (*multithreaded server*).

Un servidor multi-hilo es un servidor que acepta conexiones a través de la red, y asigna un hilo de ejecución (thread) para atender cada una de las conexiones. De este modo, todos sus clientes pueden interactuar de forma simultánea con el servidor, y, de hecho, los clientes perciben al servidor como una entidad dedicada a ellos únicamente. Sin embargo, el servidor está ejecutando múltiples instancias del código que ofrece el servicio a los clientes.

Los servidores multi-hilo son posibles gracias a que los sockets son descriptores de fichero. Ya que un proceso puede tener varios descriptores de fichero abiertos y sus hijos heredarlos, un servidor multi-hilo puede ceder los descriptores de fichero (los sockets) a cada uno de los hilos. A cada conexión abierta con el socket servidor se le asigna un socket asociado a esa comunicación y, de este modo, cada conexión entrante al Centro de Mensajes es asignada a un socket y cedida a un thread.

A.3.1. El Protocolo de Comunicación

La estructura de los mensajes que maneja el Centro de Mensajes puede verse en la Figura A.9.

En primer lugar, el elemento principal que identifica el tipo de mensaje es el nombre. Éste, que es el tipo del mensaje, puede ser un nombre representativo del contenido, o simplemente una orden. El hecho de emplear un nombre u otro para un mensaje queda a elección del programador.

Para complementar al nombre, el mensaje incluye un contenido. El contenido del mensaje es una lista de argumentos con nombre y valor. El nombre del argumento sirve para identificar y recuperar cada uno de los argumentos del contenido. El valor puede ser de cualquier clase, siempre que pueda ser convertido a texto y recuperado posteriormente en su forma

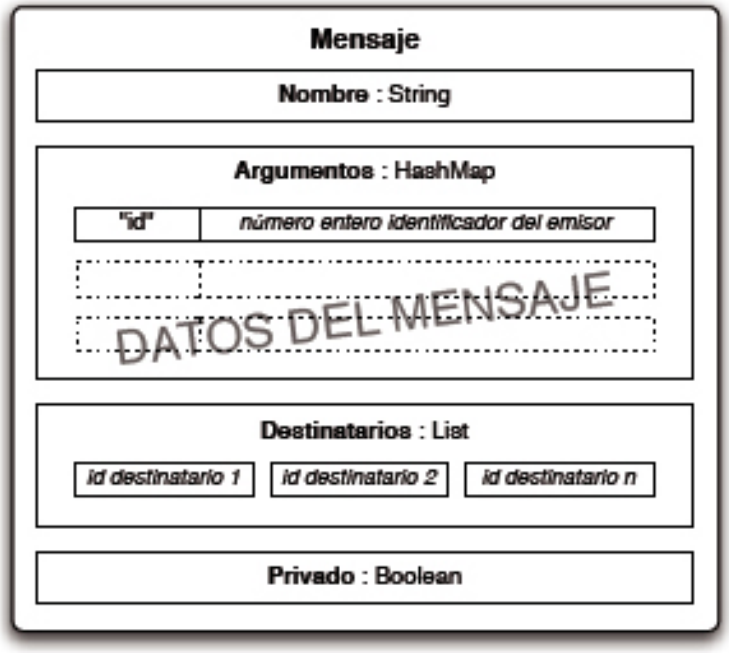

Figura A.9: Diseño de los mensajes de red

original. Un argumento puede, por tanto, ser una estructura compleja (podría ser una lista de nuevos argumentos).

El nombre y los argumentos del mensaje definen el propósito y el contenido del mensaje. Pero esto no es suficiente, se necesita información relativa al emisor y receptor del mensaje que permita el encaminamiento desde el centro de mensajes.

A cada entidad del sistema distribuido conectada al Centro de Mensajes se le asigna un identificador numérico. Este identificador se incluye siempre como un argumento más en la lista de argumentos del mensaje, para identificar a su emisor. También se podría haber incluido como un parámetro más del mensaje, pero se tomó la decisión de añadirlo como argumento para aumentar la compatibilidad con el sistema interno de paso de mensajes del subsistema gráfico, que ya lo hacía de esta forma. Aunque el emisor no incluya su identificador en la lista de argumentos del mensaje, o incluso si introduce un identificador que no se corresponde con el suyo, el mensaje llegará al receptor con el identificador de emisor correcto, ya que el Centro de Mensajes corrige o añade el identificador del emisor al mensaje si es necesario.

Otro parámetro importante para el encaminamiento es la lista de destinatarios. Esta lista enumera cada uno de los receptores a los que

obligatoriamente se les debe enviar el mensaje, aunque no estén suscritos al tipo del mismo. Un mensaje puede tener una lista de receptores vacía, algo muy común en los mensajes de tipo evento, que serán recibidos por los clientes interesados mediante el mecanismo de suscripción. El envío de un mensaje sin receptores se corresponde con el mecanismo de publicación.

Por último, el mensaje contiene un valor booleano que indica si el mensaje es privado. Este campo está pensado para poder saltar las reglas de los mecanismos de publicación/suscripción, y hacer el sistema más flexible. Para comprender la utilidad de este parámetro, veremos antes, de forma breve, el mecanismo de suscripción. Los clientes del Centro de Mensajes tienen varias posibilidades de suscripción. Una de ellas es suscribirse a un tipo de mensaje, esto es, solicitar al Centro de Mensajes que le envíe todos los mensajes de un determinado tipo. Un emisor que emita un mensaje de un tipo en el que están interesados varios clientes del Centro de Mensajes podría querer, sin embargo, enviar ese mensaje con ese tópico a un receptor o conjunto de receptores determinado sin que el resto de clientes suscritos tenga constancia del mensaje.

Cuando el parámetro privado es verdadero, el comportamiento del sistema es precisamente ése, ignorando los clientes suscritos a ese tipo de mensaje y enviándolo solamente a los receptores especificados de forma explícita en el mensaje.

El mecanismo de suscripción sirve para que el cliente haga conocer al Centro de Mensajes su interés por un determinado tipo de mensaje. Para realizar una suscripción, el cliente deberá enviar un mensaje del tipo `Suscripcion` a través del canal de comunicación.

Si un cliente desea modificar la suscripción a un mensaje para cambiar la lista de emisores deseados, no es necesario que se desuscriba al mensaje para volver a realizar la suscripción. El envío de un nuevo mensaje de suscripción que contenga como argumento tipo el mismo nombre de un tipo de mensaje ya suscrito, reemplaza la lista de identificadores de emisores permitidos en la configuración del centro de mensajes.

Además, es posible suscribirse a los mensajes de tipo `Suscripcion`, al igual que es posible suscribirse a cualquiera de los otros tipos de mensaje reservados por el Centro de Mensajes.

Ya que esto es posible, también puede darse el caso en el que el emisor no desee que los demás clientes tengan conocimiento de la suscripción. Para ello, bastará con marcar el campo privado del mensaje con el valor true. También es posible indicar en la lista de destinatarios los identificadores de clientes que se desea que reciban una notificación de la suscripción, aunque no estén suscritos a este tipo de mensaje.

A.3.2. Diseño del Centro de Mensajes

El diseño orientado a objetos del Centro de Mensajes refleja la estructura multi-hilo ya comentada, y provee la infraestructura necesaria para encaminar mensajes y gestionar las suscripciones de los clientes, como vemos en el diagrama de clases representado en la Figura A.10.

La instancia de `CentroMensajes` será el punto de partida para las conexiones de nuevos clientes, a los que se les asignarán objetos de la clase `ClienteConectado`. Las instancias de `ClienteConectado` se comunicarán con sus clientes gracias a la librería de sockets orientada a eventos. Esta librería, diseñada de forma parecida a un patrón Observer (patrón Observador), obliga a la clase cliente a implementar ciertos métodos asociados con eventos del socket.

Vemos también una clase `Mensaje`, usada para representar un mensaje en forma de objeto, así como para proporcionar métodos para traducir mensajes *JSON* en objetos y viceversa. La `TablaClientesConectados` mantiene la lista de clientes conectados al sistema. La instancia de la clase `TablaRutas` mantendrá las estructuras que definen las suscripciones de los clientes, representadas por la clase `TablaSuscripcion`, y encaminará los mensajes hacia los objetos `ClienteConectado` oportunos según la configuración de estas estructuras. Es visible que la aferencia de la clase `ClienteConectado` supera a la del resto de clases, seguida, de cerca por la tabla de rutas. Es evidente, por tanto, que sobre estas clases recaerá la mayor carga computacional del sistema, y que ambas son las clases más importantes.

A.3.3. Agentes de Comunicación

El diseño de los agentes de comunicación, ilustrado en la Figura A.12, se estructura en una serie de comportamientos, los cuales derivan de las clases `Behaviour` ofrecidas por *JADE*.

Con el fin de facilitar la ampliación del conjunto de mensajes de red disponibles, se ha incluido un sistema que permite tratar nuevos tipos de mensaje sin tener que modificar el código del Agente de Comunicación del estudiante. Mediante la técnica de *reflexión*, el Agente de Comunicación es capaz de establecer de forma dinámica qué clase de tratamiento debe darse al mensaje que acaba de llegar a través de la red. Para ello se usa la clase `Tratamiento`, la cual incluye atributos con información relevante sobre el agente. De este modo, las subclases de `Tratamiento` pueden acceder a información del agente que pueda ser necesaria para realizar las operaciones requeridas (enviar un mensaje *ACL* a otro agente, por ejemplo).

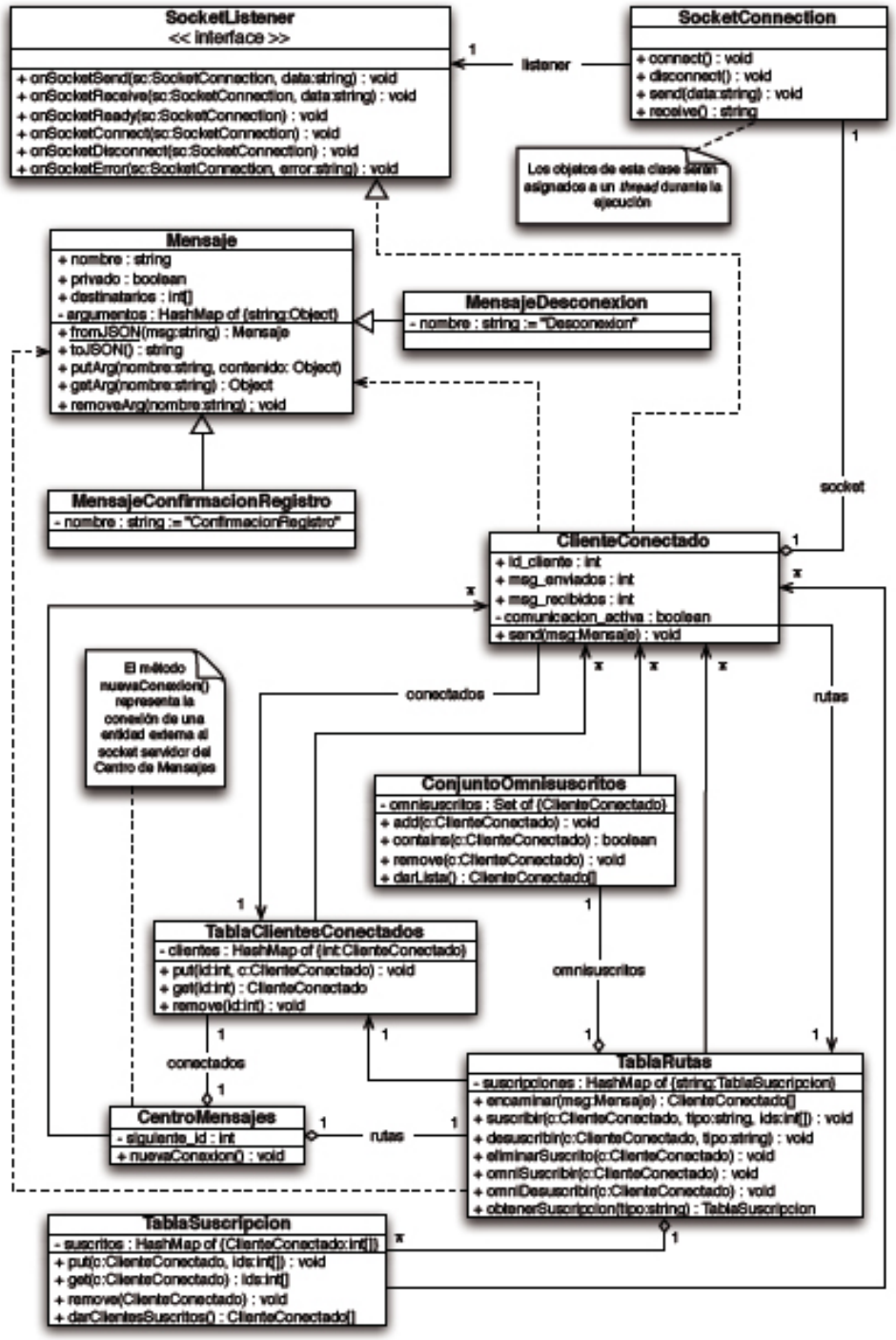

Figura A.10: Diseño del Centro de Mensajes

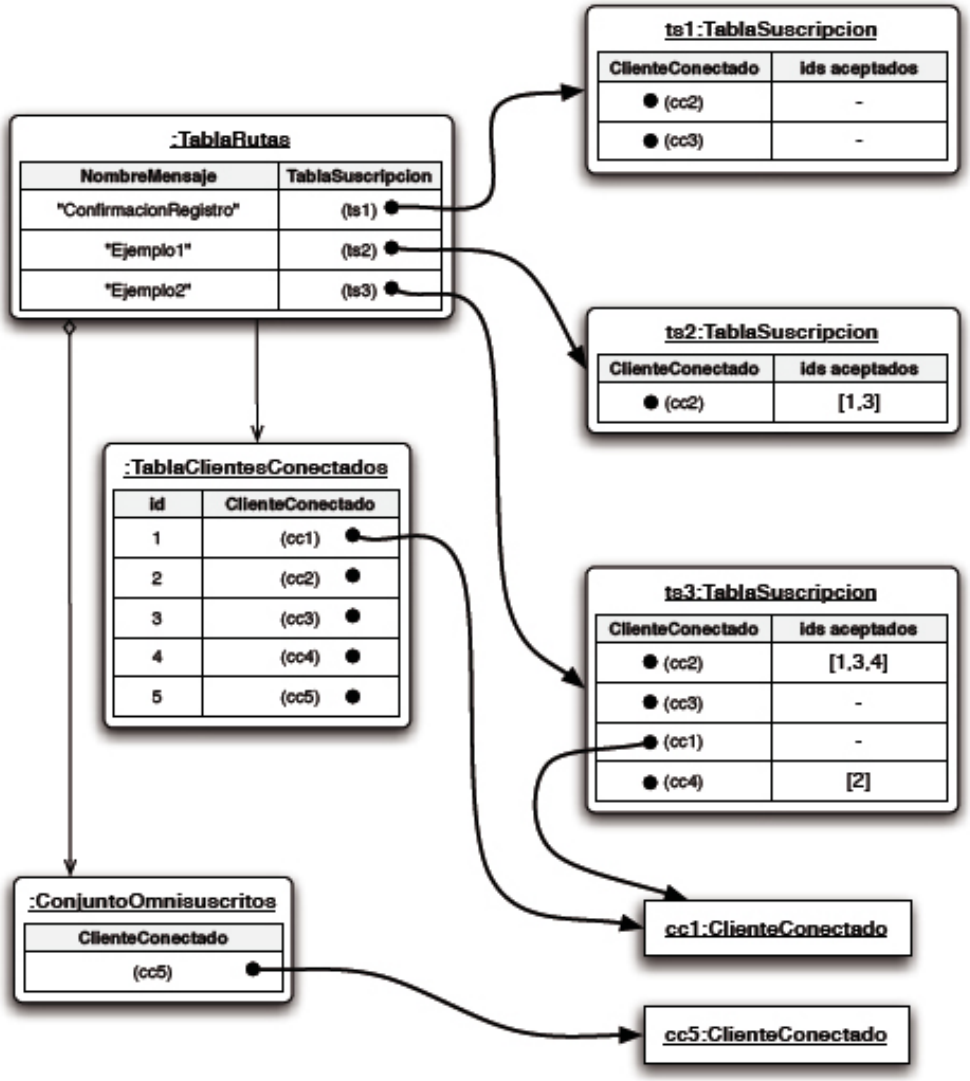

Figura A.11: Diseño de la Tabla de Rutas

Cuando un mensaje llega desde la red, el Agente de Comunicación deduce y construye, a partir del nombre del mensaje, el nombre de la subclase de `Tratamiento` que debe utilizar (mediante reflexión), incluyendo en el constructor de la clase los argumentos del mensaje, y a continuación llama a su método `tratar()`.

Cada Agente de Comunicación de estudiante guarda en sus atributos dos identificadores:

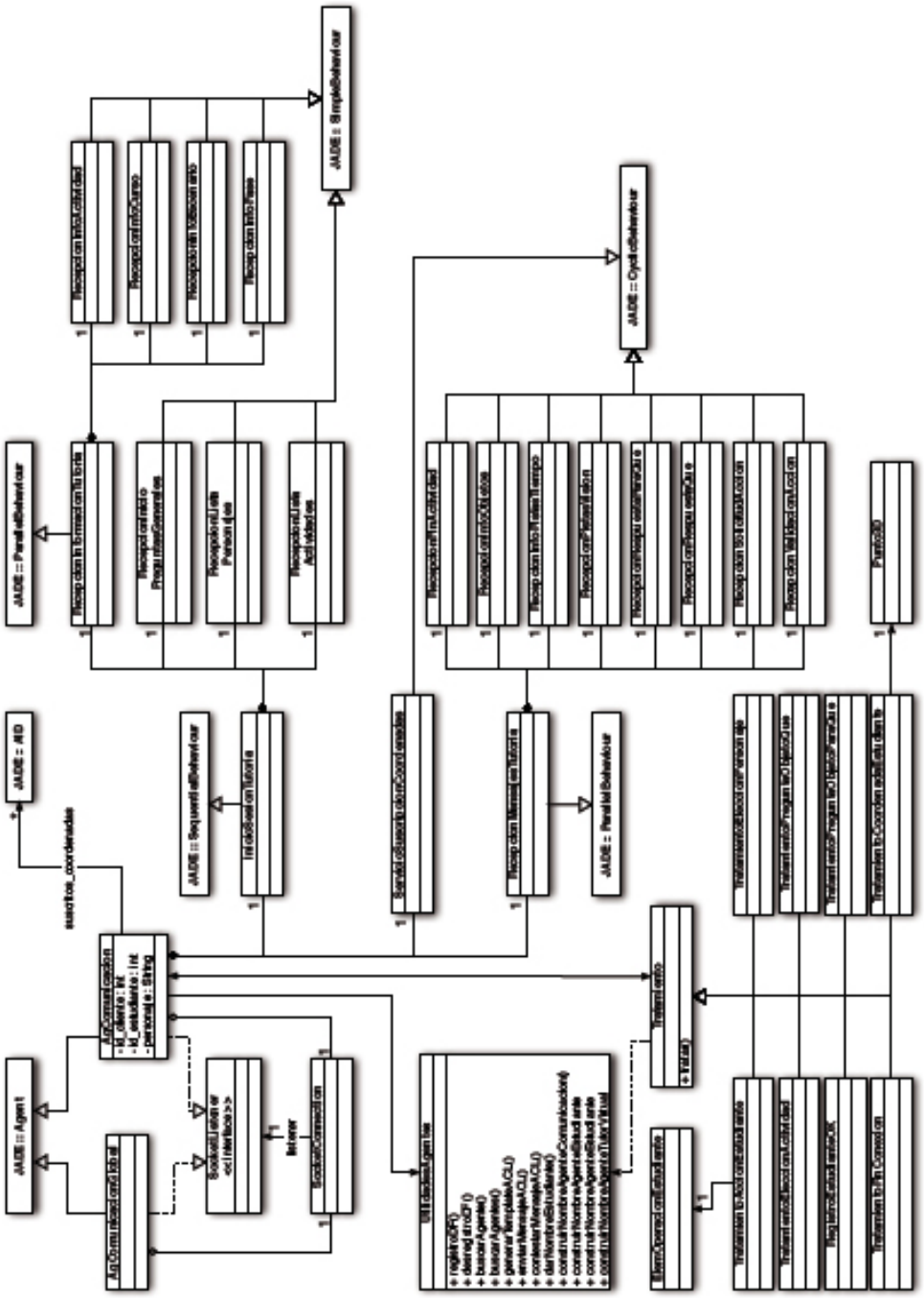

Figura A.12: Diseño de los Agentes de Comunicación

- id estudiante: es el identificador asignado por el Centro de Mensajes en el mensaje `ConfirmacionRegistro` enviado al Agente de Comunicación de estudiante tras su conexión.
- id cliente: es el identificador correspondiente al subsistema gráfico asociado al Agente de Comunicación, y le es comunicado al Agente de Comunicación de estudiante por el Agente de Comunicación Global, el cual recibió ese identificador ya que se suscribe a los mensajes del tipo `ConfirmacionRegistro`, detectando así las entradas al sistema de los subsistemas gráficos.

La clase `UtilidadesAgentes` es una clase común a todos los agentes que facilita tareas como la inicialización del agente o el envío de mensajes *ACL*. El diagrama de clases, al contener un gran número de clases, no detalla los métodos ya ilustrados en otras figuras, como pueden ser los de las clases pertenecientes a la librería de sockets.

A.3.4. Diseño del Observador de Red en el EV

El diagrama de clases de la Figura A.13 muestra el diseño del observador de red. El subsistema gráfico ofrece una superclase llamada `ObservadorGestorEntidades`, que permite, heredando de la misma, recibir y comunicar datos al Gestor de Entidades Central del subsistema gráfico.

Esta clase ofrece en su interfaz dos métodos que nos interesan, `recibidoMensaje()` y `tick()`, y mediante una referencia al Gestor de Entidades tiene acceso a otro método interesante, `enviarMensaje()`.

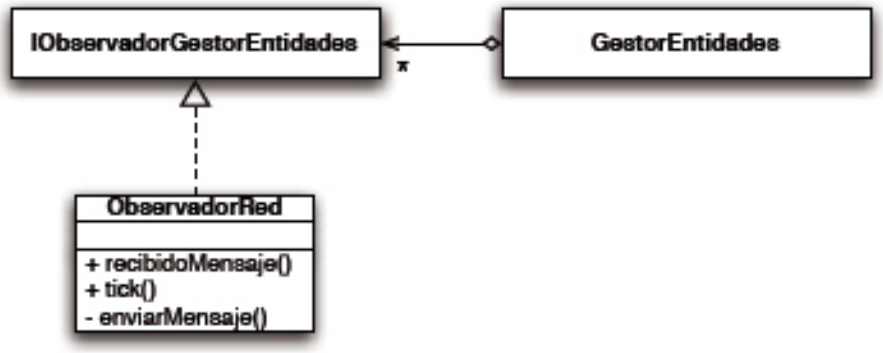

Figura A.13: Diseño del observador de red

El método `recibidoMensaje()` es un método virtual que debe ser implementado en aquellos observadores con subclases que hereden de

`ObservadorGestorEntidades` que quieran hacer uso de este modo, y que ha sido implementado, por tanto, en el Observador de Red. Cuando el Gestor de Entidades Central genera un mensaje, éste es comunicado a todos los observadores registrados invocando el método `recibidoMensaje()` de cada uno de los observadores. Por lo tanto, es necesario tratar el mensaje recibido en la implementación de este método en la subclase del observador.

Cada observador puede realizar un filtro del mensaje según el tipo del mensaje. El Observador de Red filtra los mensajes de tal forma que sólo trate aquellos mensajes que están relacionados con la red. Los mensajes de red son tratados (transformados de objeto a cadena de caracteres en formato *JSON*) y enviados a través del socket que lo conecta con el Centro de Mensajes. En concreto, al observador de red le interesan los mensajes de `RegistroCliente`, `EleccionActividad`, `EleccionPersonaje`, `CoordenadaEstudiante`, `AccionEstudiante`, `PreguntaObjetoQue` y, finalmente, `PreguntaObjetoParaQue`. El Observador de Red desconoce en todo momento qué módulo del subsistema gráfico genera cada mensaje, puesto que le son comunicados desde el Gestor de Entidades.

Cuando el subsistema gráfico comienza su ejecución, el Observador de Red se conecta al Centro de Mensajes, el cual comunicará al observador el identificador asignado mediante el mensaje `ConfirmacionRegistro`. A continuación, el Observador de Red se suscribirá a los mensajes de tipo `CoordenadaEstudiante`, para de este modo poder recibir las coordenadas generadas por otros subsistemas gráficos y actualizar las posiciones de los avatares correspondientes.

El resto de mensajes procedentes del SIT son enviados con destinatario explícito por su Agente de Comunicación asociado, y por lo tanto no es necesario que el subsistema gráfico se suscriba a ellos.

El método `tick()` es llamado por el Gestor de Entidades Central del Cliente Gráfico en cada paso de la simulación, con una frecuencia que se corresponde con la tasa de generación de frames gráficos. Esto sirve para comprobar el estado del canal de entrada del socket conectado a la red para verificar si hay nuevos mensajes. En caso afirmativo, el `ObservadorRed` inyecta el mensaje *JSON* en el subsistema gráfico a través del `GestorEntidades` para que la entidad pertinente lo recoja.

A.3.5. Envío de Mensajes

Para complementar los diseños anteriormente expuestos, se describirá de forma gráfica el viaje de un mensaje desde que es generado hasta que llega a su destino. Los mensajes pueden pasar por varias formas: en el lenguaje de los agentes, los mensajes llevan el formato de un mensaje *ACL*; en el lenguaje del protocolo de red, los mensajes están representados en formato *JSON*.

La Figura A.14 muestra el camino y los métodos invocados para encaminar un mensaje desde la entidad emisora del subsistema gráfico hasta el agente del SIT receptor de la información. Aunque no tiene que ver con el viaje del mensaje, se ha incluido en el diagrama la llamada que hace el objeto `SocketConnection` al objeto que llama a su método `send()`, para mostrar el funcionamiento de la librería.

La Figura A.15 muestra cómo un mensaje viaja desde un agente del SIT al subsistema gráfico. Puede observarse aquí cómo el Gestor de Entidades del subsistema gráfico no recibe el mensaje hasta que llama al método `tick()` del Observador de Red, dentro del cual se recoge el mensaje del socket y se invoca el método `enviarMensaje()` de `GestorEntidades`.

La Figura A.16 ilustra la conexión de un nuevo subsistema gráfico al sistema. Tras la conexión al Centro de Mensajes, el Observador de Red recibe de éste el mensaje `ConfirmacionRegistro`. Una vez el estudiante haya escogido su nombre de usuario, el Gestor de Entidades lo hará saber al Observador de Red, usando un mensaje `RegistroCliente`. Este mensaje es enviado al Centro de Mensajes, que a su vez lo reenviar á al Agente de Comunicación Global, que en el inicio de su ejecución se ha suscrito a este tipo de mensaje. Cuando el Centro de Mensajes transmite el mensaje al Agente de Comunicación Global, éste crea los tres agentes asociados al subsistema gráfico: el de comunicación, el de estudiante, y el tutor virtual. El Agente de Comunicación recién creado procederá a conectarse al Centro de Mensajes con los datos proporcionados por el Agente de Comunicación Global.

La Figura A.17 muestra el recorrido de un mensaje que viaja de un cliente a otro, como una coordenada generada por una acción de movimiento del estudiante.

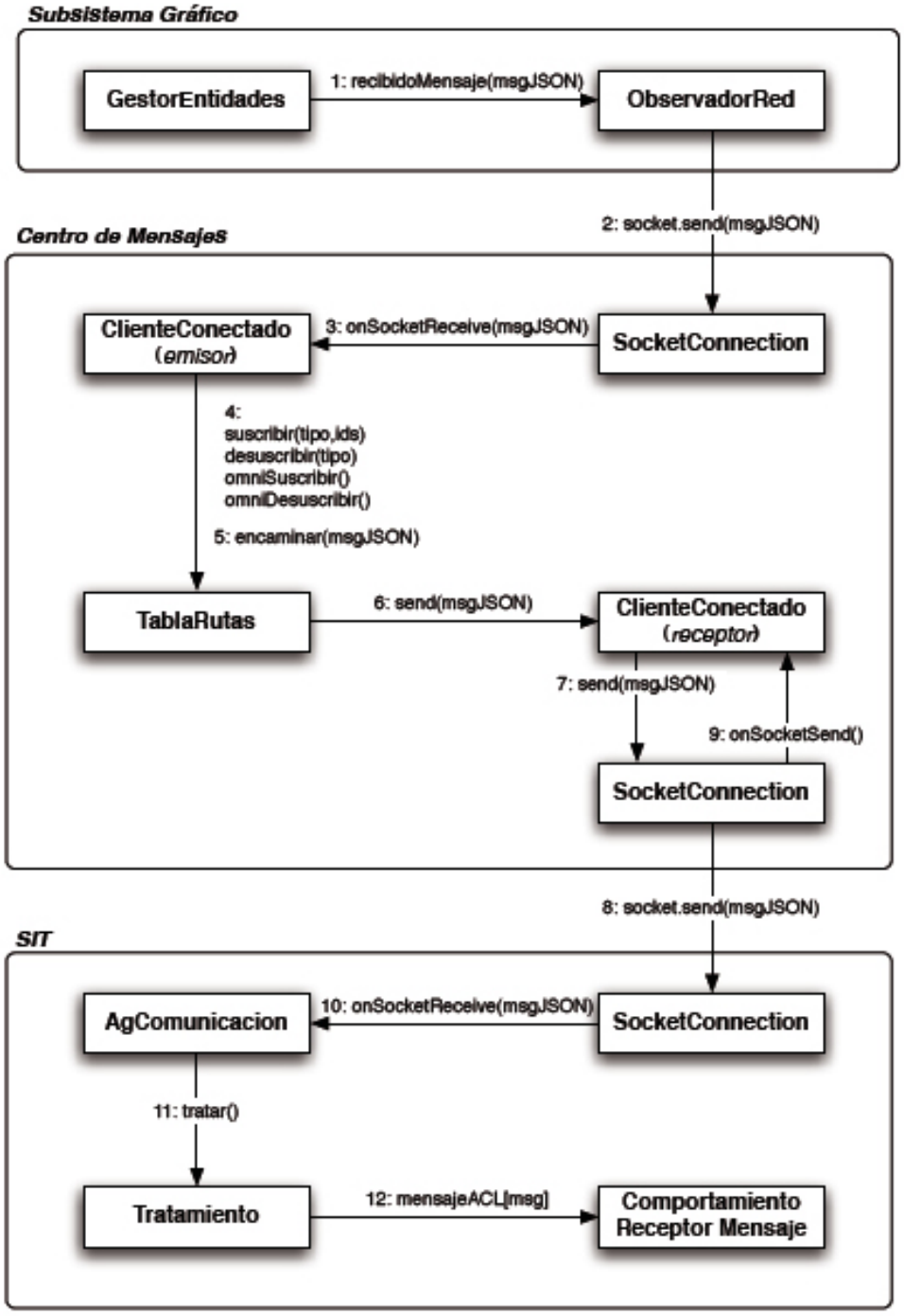

Figura A.14: Mensaje del EV al SIT

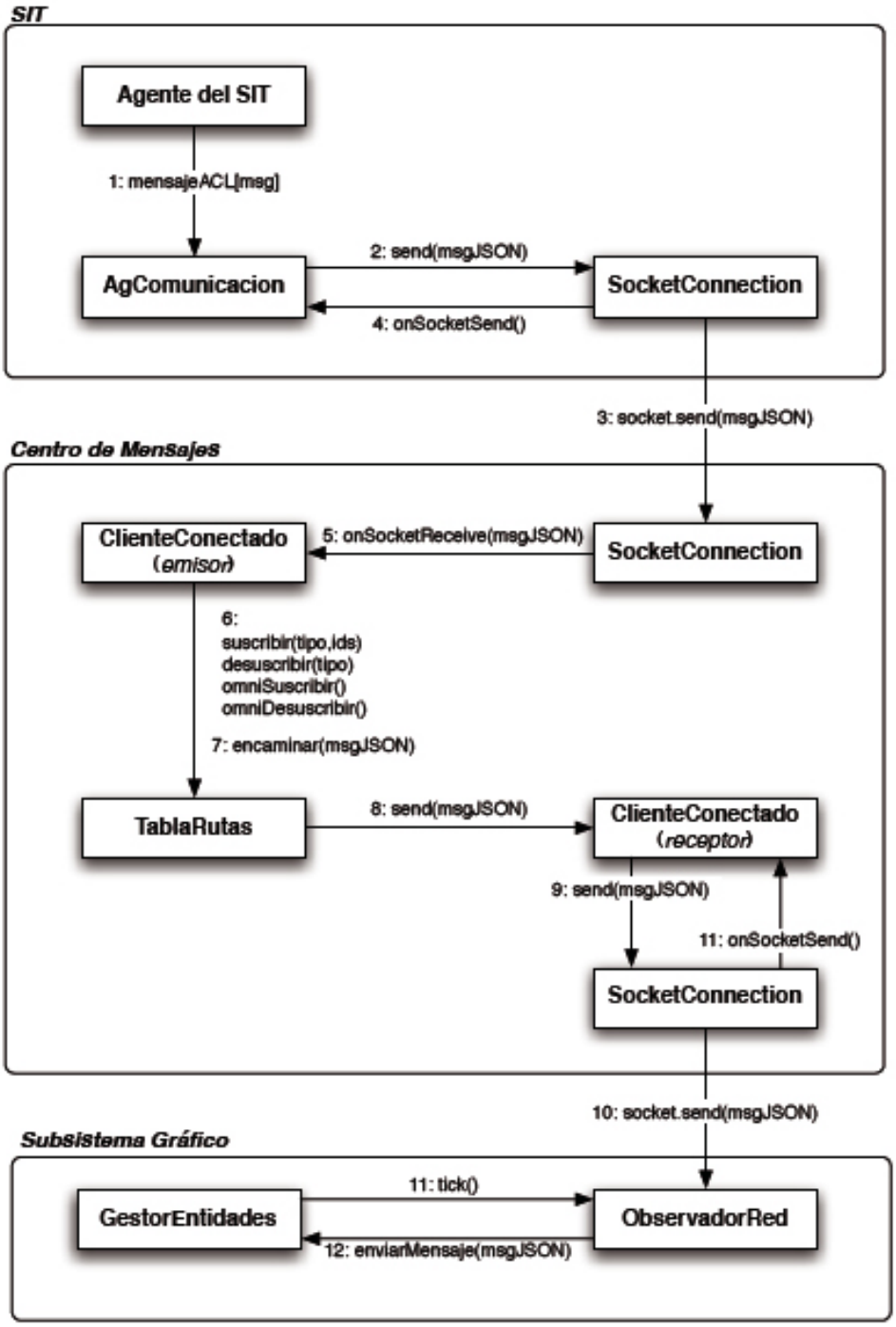

Figura A.15: Mensaje del SIT al EV

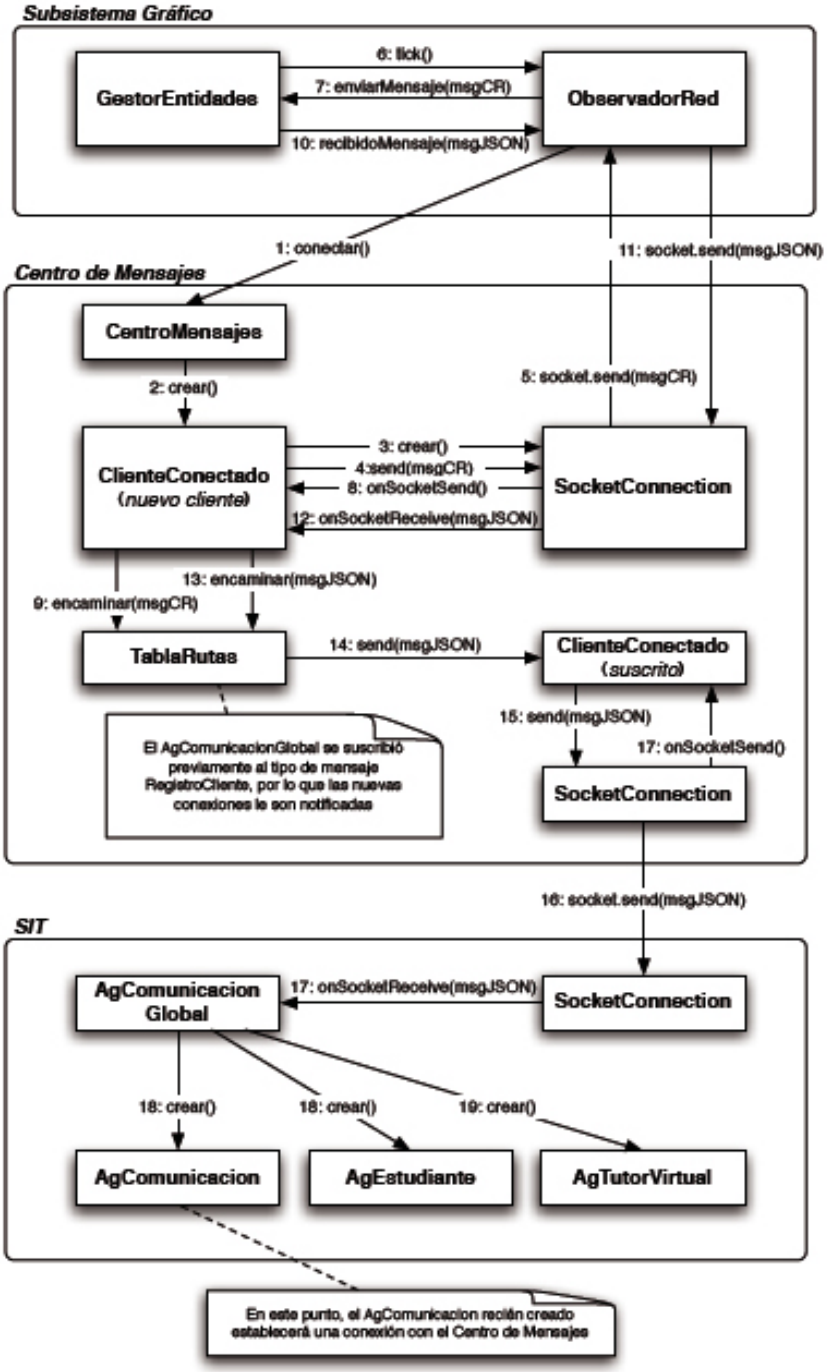

Figura A.16: Conexión de un EV al sistema

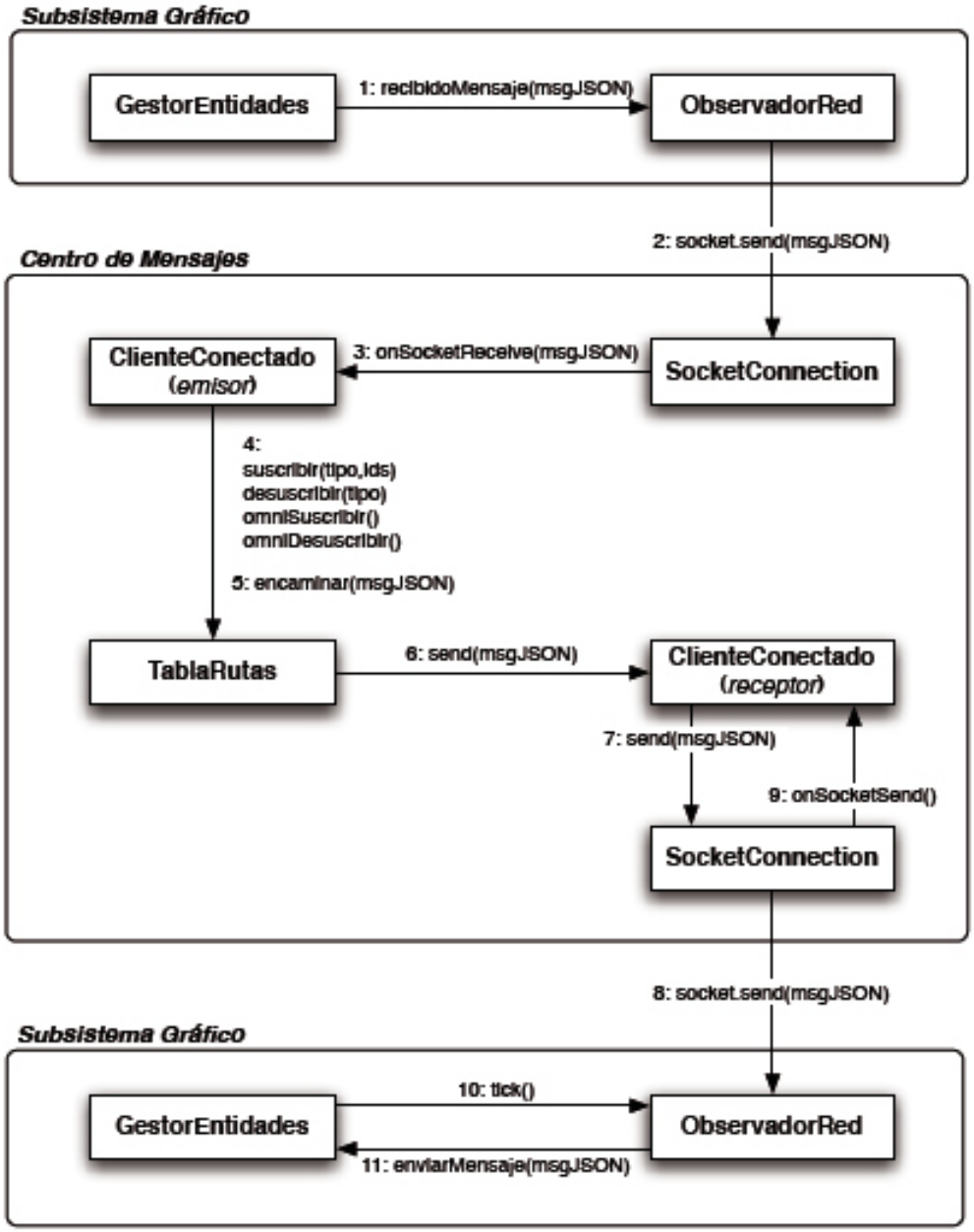

Figura A.17: Mensaje entre dos clientes

A.4. Diseño del Agente de Tutoría

A continuación se muestran de forma detallada las clases que representan al Agente de Tutoría y a los comportamientos que se pueden lanzar dentro de él. El método `setup()` del Agente de Tutoría se ejecuta nada más comenzar, y dentro de dicho método se pueden lanzar nuevos comportamientos. Se ha de tener en cuenta que los comportamientos también tienen la capacidad de iniciar a otros, y que pueden ser varios los que se encuentren en ejecución en un mismo instante de tiempo. En general, al añadir comportamientos creamos un nuevo hilo de ejecución cooperativo dentro del agente. Es un planificador, invisible para el programador, el que lleva a cabo una política de planificación *round-robin* entre todos los comportamientos que estén en cada momento en la cola de ejecución. En cuanto al resto de métodos del Agente de Tutoría, pueden ser invocados desde cualquier comportamiento del agente.

Se puede observar que todas las clases que representan a los comportamientos tienen tres métodos. Además del constructor, todos poseen un método `action()` y otro `done()`. El planificador del que se ha hablado antes ejecuta el método `action()` de cada comportamiento presente en la cola de ejecución. Cuando termina dicho método, se invoca al método `done()` para comprobar si el comportamiento ha terminado su tarea, y si es éste el caso se elimina de la cola. Los comportamientos internamente tienen una estructura de autómata finito determinista (*AFD* a partir de ahora).

En los diagramas que se mostrarán de aquí en adelante no se detallarán los parámetros de los métodos de las clases para favorecer su legibilidad.

A.4.1. Plan de Estudios

En el diagrama que se muestra en este punto queda representada la estructura del plan de estudios que maneja el Agente de Tutoría. Estas estructuras se completan a partir de la información proporcionada por la herramienta de autor al sistema de agentes que, como se explicará en los siguientes capítulos, queda recogida en una serie de ficheros sobre los que se tendrá que hacer parsing.

El Agente de Tutoría tendrá disponibles un conjunto de cursos. Cada uno de ellos tendrá asociada una determinada estrategia de tutoría, en la que se definirán entre otras cosas el tratamiento de los fallos y la forma de evaluar al alumno. Cada curso contiene una lista de fases, a su vez compuestas por un conjunto de actividades.

Las actividades se realizan en un escenario determinado que contiene una serie de objetos y ocupan una posición concreta dentro de él. El estado

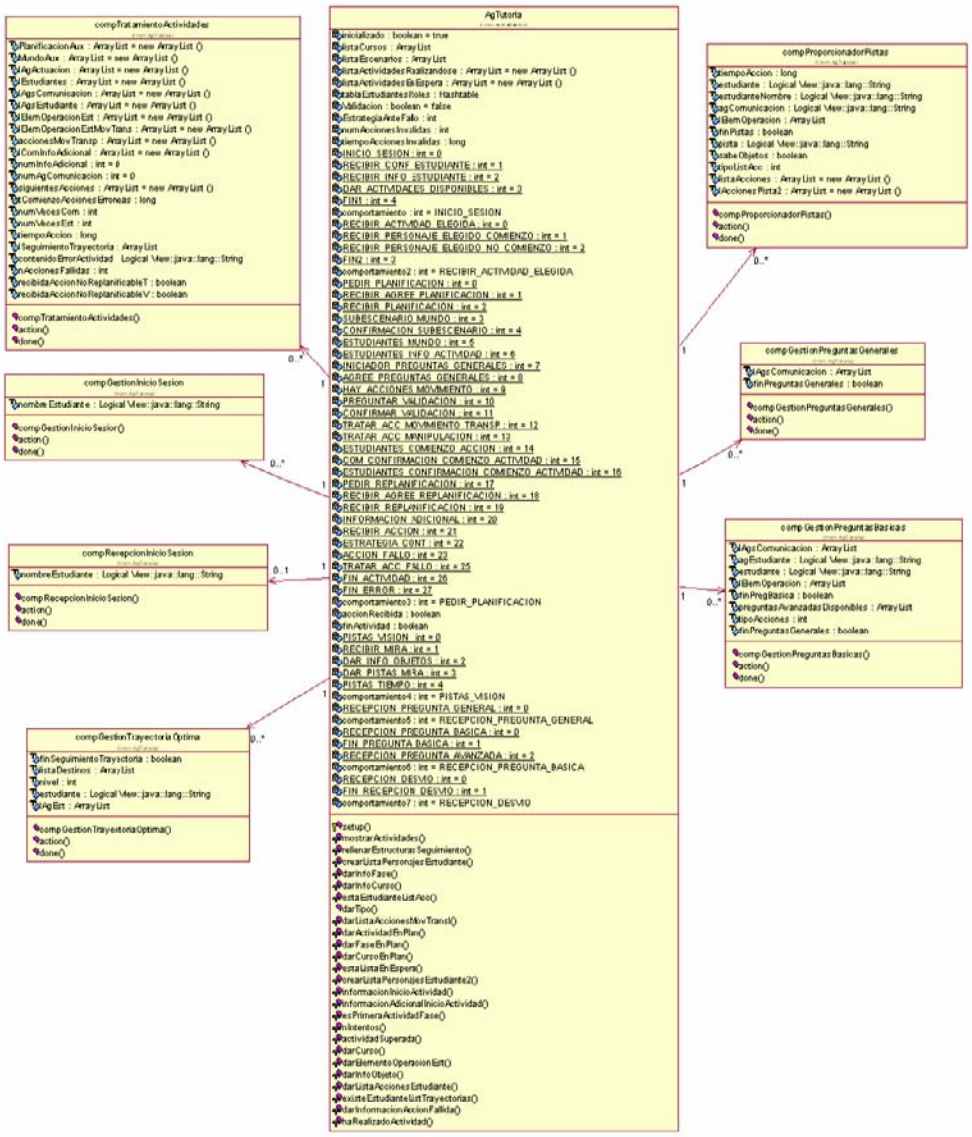

Figura A.18: Comportamientos del Agente de Tutoría

inicial de los objetos podría variar según el tipo de actividad que se vaya a desarrollar; dicha información se almacena en la clase `EstadoObjeto`. Por cada actividad, se detalla también el conjunto de personajes que tienen que participar en ella, junto con el rol que representan y la posición que ocupan dentro del escenario.

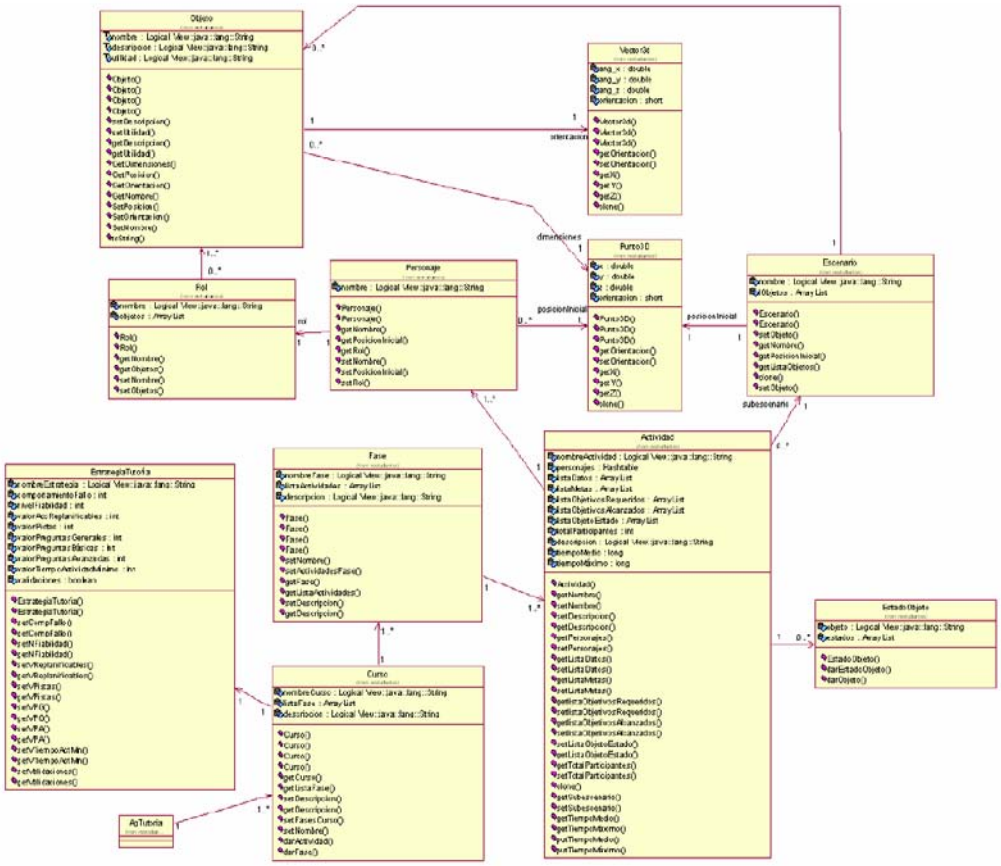

Figura A.19: Estructura del plan de estudios

A.4.2. Plan de la Actividad

En el siguiente fragmento de diagrama queda representada la estructura del plan que proporciona al Agente de Tutoría el Agente de Planificación. Como se puede observar, un plan está compuesto por uno o varios elementos (clase `Elem`), que a su vez pueden ser de dos tipos, acciones simples (clase `ElemOperacion`) o bloques de acciones múltiples (clase `ElemOperacionMult`). Los bloques de acciones están compuestos por un conjunto de elementos y no quedan realizados hasta que los estudiantes no ejecutan todos ellos. Se ha de tener en cuenta que los elementos de los que se habla pueden ser acciones u otros bloques de acciones múltiples. Los bloques de acciones también pueden ser de dos tipos, dependiendo de si se tiene en cuenta el orden en que se ejecutan los elementos que lo componen (`ElemOperacionMultOrd`), o no (`ElemOperacionMultUnord`), de ahí la jerarquía en el diagrama.

Las acciones tienen un atributo de la clase `Predicate` de JSHOP2 (*Java Simple Hierarchical Ordered Planner*, Planificador Jerárquico HTN utilizado por el Agente de Planificación para crear el plan de una actividad), de donde se obtendrá información como la lista de objetos se tienen que manipular, o los lugares a los que hay desplazarse.

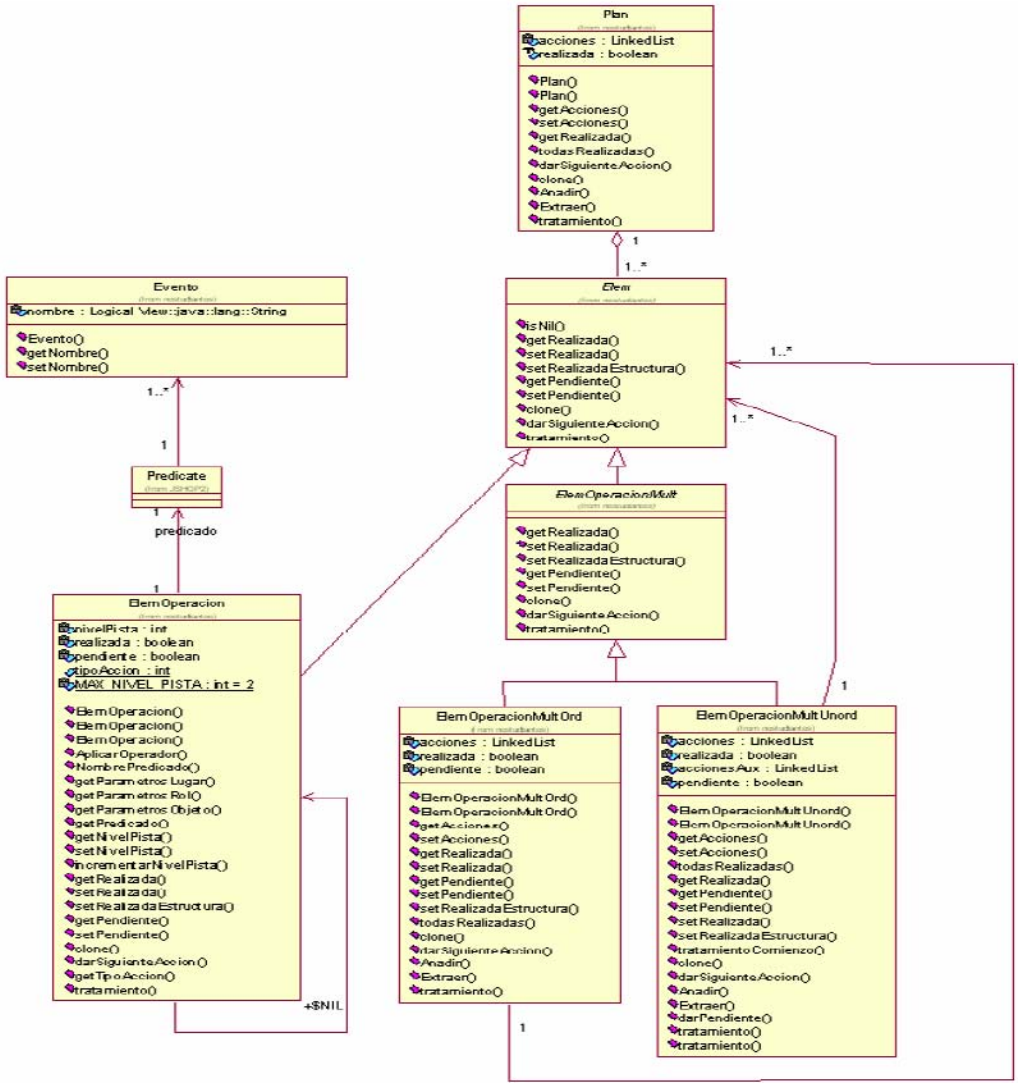

Figura A.20: Estructura del plan de actividad

A.4.3. Seguimiento del Estudiante

Por cada estudiante que inicie una sesión se debería hacer un seguimiento individualizado de su proceso de aprendizaje. El sistema está pensado para que el alumno reciba una serie de cursos de formación durante un periodo de tiempo, por lo que se debería recoger su evolución al finalizar cada sesión para partir desde el punto en que lo dejó cuando decida continuar con los cursos. El encargado de almacenar estos datos es el Agente Estudiante. Por tanto, el Agente de Tutoría le tendría que pedir dicha información por cada inicio de sesión. Durante la realización de la actividad, esta información se tendrá que ir actualizando a partir de las acciones que vaya realizando el alumno y el Agente Estudiante recogerá los resultados obtenidos.

La estructura para el seguimiento del estudiante va a ser la misma que la que se creó para almacenar el plan de estudios, pero la información que se recoge será distinta. Al igual que antes, por cada estudiante se guardará la lista con todos los cursos que forman parte del plan, que, a su vez, están formados por una serie de fases que contienen un conjunto de actividades. Por cada uno de estos elementos se almacena si se ha superado, el número de intentos (en el caso de las actividades), si se ha comenzado (en el caso de las fases y los cursos, si se ha realizado alguna de las actividades que forman parte de ellos), y la evaluación conseguida.

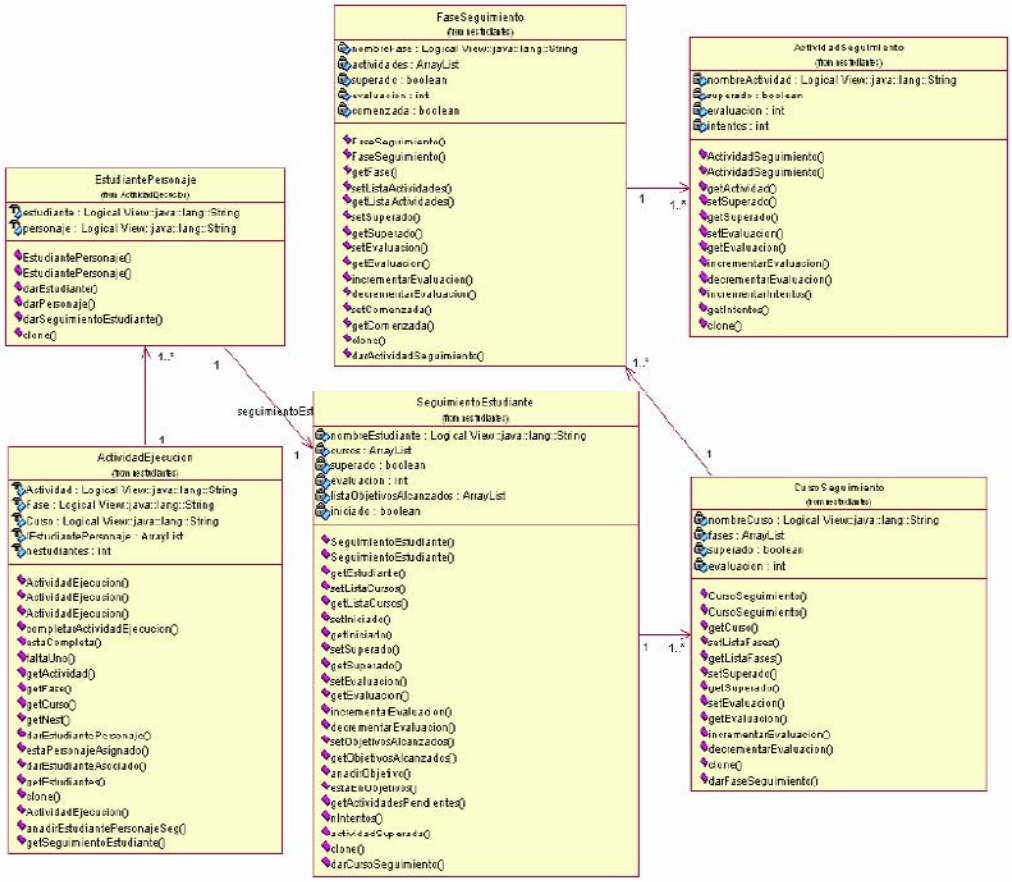

Figura A.21: Estructura para el seguimiento del estudiante

Parte V

Bibliografía

Bibliografía

Agre, P. E. (2003) Hierarchy and History in Simon's "Architecture of Complexity". *Journal of the Learning Sciences*, 12(3): págs. 413–426.

Akbari, M. y Karbalaee, O. (2005) Characteristics of SOA for virtual agent-based electronic marketplaces (VABEM). *En International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, tomo 2, págs. 934–939.

Alem, L. y Keeling, R. (1996) Intelligent Simulation Environment an Application to Air Traffic Control Training. *En Proceedings of the first international conference on Advancing simulation technology and training, SimTecT'96*, págs. 323–328.

Alem, L. et al. (1997) FILIP and intelligent simulator based framework for skill acquisition a case in air traffic control. *En* W. van Jooligen, A. Munro y L. Alem (eds.), *Proceedings of AI-ED'97 workshop on Architecture for intelligent simulation based learning environment*.

André, E. (ed.) (1997) *Proceedings of the IJCAI Workshop on Animated Interface Agents: Making Them Intelligent*.

André, E. et al. (1996) The PPP persona: a multipurpose animated presentation agent. *En AVI '96: Proceedings of the Workshop on Advanced Visual Interfaces*, págs. 245–247. ACM Press, New York, NY, USA.

André, E. et al. (1998) Integrating reactive and scripted behaviors in a life-like presentation agent. *En Proceedings of the Second International Conference on Autonomous Agents*, págs. 261–268.

André, E. et al. (1999) Employing AI Methods to Control the Behavior of Animated Interface Agents. *Applied Artificial Intelligence*, 13(4-5): págs. 415–448.

André, E. et al. (2000) The Automated Design of Believable Dialogues for Animated Presentation Teams. *En* J. Cassell, J. Sullivan, S. Prevost y E. Churchill (eds.), *Embodied Conversational Agents*. MIT Press, Cambridge, MA.

Arcos, J. L. L. et al. (2000) LAHYSTOTRAIN: Integration of Virtual Environments and ITS for Surgery Training. *En* G. Gauthier, C. Frasson y K. VanLehn (eds.), *Intelligent Tutoring Systems (2000)*, nº 1839 en LNCS, págs. 43–52. Springer-Verlag.

Asay-Davis, X. S. et al. (2000) Virtual Explorer: Interactive Virtual Environment for Education. *Presence: Teleoperators and Virtual Environments*, 9(6): págs. 505–523.

Aurenhammer, F. (1991) Voronoi diagrams — a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3): págs. 345–405.

Bachmann, F. et al. (2007) Modifiability Tactics. Inf. Téc. CMU/SEI-2007-TR-002, Software Engineering Institute - Carnegie Mellon University, Pittsburg, PA, USA.

Badler, N. I. et al. (1993) *Simulating Humans*. Oxford University Press, New York.

Ball, G. et al. (1997) Lifelike Computer Characters: the Persona Project at Microsoft. *En* J. Bradshaw (ed.), *Software Agents*. AAAI/MIT Press.

Bansler, J. P. y Bodker, K. (1993) A reappraisal of structured analysis: design in an organizational context. *ACM Transactions on Information Systems*, 11(2): págs. 165–193.

Baranauskas, M. C. C. y Borges, M. A. F. (1997) Jonas - The Jonah agent in a computer-based learning system. *En Proceedings of the International Conference: Information Technology for Competitiveness - Experiences and Demands for Education and Vocational Training*.

Baranauskas, M. C. C. et al. (1997) Learning by creating models: a computer-based environment for industrial application. *En Proceedings of the 8th Conference on Artificial Intelligence in Education (AIED97)*, págs. 426–433.

Barbacci, M. R. et al. (2003) Quality Attribute Workshops, Third Edition. Inf. Téc. CMU/SEI-2003-TR-016, Software Engineering Institute - Carnegie Mellon University, Pittsburg, PA, USA.

Barrus, J. W. et al. (1996) Locales: Supporting Large Multiuser Virtual Environments. *IEEE Computer Graphics and Applications*, 16(6): págs. 50–57.

Bass, L. et al. (2000) Quality Attribute Design Primitives. Inf. Téc. CMU/SEI-2000-TN-017, Software Engineering Institute - Carnegie Mellon University, Pittsburg, PA, USA.

Bass, L. et al. (2001) Quality Attribute Design Primitives and the Attribute Driven Design Method. *En* F. van der Linden (ed.), *PFE '01: Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, tomo 2290 de *Lecture Notes in Computer Science*, págs. 169 – 186. ACM, Springer-Verlag.

Bass, L. et al. (2003) *Software Architecture in Practice*. Addison-Wesley, Boston, MA, second edition ed$^{\underline{\text{ón}}}$.

Bastos, L. R. D. y Castro, J. F. B. (2005) From requirements to multi-agent architecture using organisational concepts. *SIGSOFT Software Engineering Notes*, 30(4): págs. 1–7.

Baylor, A. L. (2001) Investigating multiple pedagogical perspectives through MIMIC (Multiple intelligent mentors instructing collaboratively). *En Proceedings of Artificial Intelligence in Education (AI-ED) International Conference*.

Bellifemine, F. L. et al. (2007) *Developing Multi-Agent Systems with JADE*. Wiley Series in Agent Technology. Wiley.

Borges, M. A. F. y Baranauskas, M. C. C. (1997) A User-centered Approach to the Design of an Expert System for Training. *En Proceedings of the 8th International PEG Conference*, págs. 74–81.

Boucké, N. y Holvoet, T. (2008) View composition in multiagent architectures. *International Journal of Agent-Oriented Software Engineering*, 2(1): págs. 3–33.

Boucké, N. et al. (2006) Applying the ATAM to an Architecture for Decentralized Control of a Transportation System. *En Second International Conference on Quality of Software Architectures*, tomo 4214 de *Lecture Notes in Computer Science*, págs. 180–198. Springer-Verlag.

Bresciani, P. et al. (2004) TROPOS: An Agent-Oriented Software Development Methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, 8: págs. 203–236.

Bricken, W. (1990a) Learning in Virtual Reality. Inf. téc., University of Washington, Human Interface Technology Laboratory.

Bricken, W. (1990b) Training in Virtual Reality. Inf. téc., University of Washington, Human Interface Technology Laboratory.

Broersen, A. y Nijholt, A. (2002) Developing a Virtual Piano Playing Environment. *En* K. . I. G. V. Petrushin, P. Kommers (ed.), *Proceedings IEEE International Conference on Advanced Learning Technologies (ICALT 2002)*, págs. 278–282. IEEE.

Brown, J. S. y Burton, R. R. (1978) Diagnostic models for procedural bugs in basic mathmatical skills. *Cognitive Science*, 2: págs. 155–192.

Brusilovsky, P. et al. (1996) ELM-ART: An intelligent tutoring system on world wide web. *En* C. Frasson, G. Gauthier y A. Lesgold (eds.), *Proceedings of the Third International Conference on Intelligent Tutoring Systems*, págs. 261–269. Springer-Verlag.

Buche, C. et al. (2004) MASCARET: A Pedagogical Multi-Agent System for Virtual Environments for Training. *International Journal of Distance Education Technologies*, 2(4): págs. 41–61.

Burton, R. R. (1982) Diagnosing bugs in a simple procedural skill. *En* D. Sleeman y J. S. Brown (eds.), *Intelligent Tutoring Systems*. Academic Press.

Buschmann, F. et al. (1996) *Pattern-Oriented Soaftware Architecture: A System of Patterns*, tomo 1 de *Software Design Patterns*. John Wiley & Sons.

Byrne, C. (1993) Virtual Reality and Education. *En Proceedings of IFIP WG3.5 International Workshop Conference*.

Caire, G. et al. (2001) Agent Oriented Analysis Using Message/UML. *En Agent-Oriented Software Engineering*, págs. 119–135.

Calle, J. et al. (2006) Intentional Processing as a Key for Rational Behaviour through Natural Interaction. *Interacting With Computers*, 18(6): págs. 1419–1446.

Capuano, N. et al. (2000) ABITS: An Agent Based Intelligent Tutoring System for Distance Learning. *En Proceedings of the International Workshop on Adaptive and Intelligent Web-Based Education Systems. ITS 2000*.

Carbonell, J. R. (1970) AI in CAI: An Artificial-Intelligence Approach to Computer-Assisted Instruction. *IEEE Transactions on Man Machine Systems*, 11(4): págs. 190–202.

Cassell, J. (2000) Nudge Nudge Wink Wink: Elements of Face-to-Face Conversation for Embodied Conversational Agents. *En* J. Cassell, J. Sullivan, S. Prevost y E. Churchill (eds.), *Embodied Conversational Agents*, págs. 1–27. MIT Press, Cambridge, MA.

Cassell, J. et al. (eds.) (2000) *Embodied Conversational Agents*. MIT Press.

Cassell, J. et al. (2007) Trading Spaces: How Humans and Humanoids use Speech and Gesture to Give Directions. *En* T. Nishida (ed.), *Conversational Informatics*, págs. 133–160. John Wiley & Sons.

Chan, T. W. (1996) Learning companion systems, social learning systems, and the global social learning club. *Journal of Artificial Intelligence in Education*, 7: págs. 125–159.

Chan, T. W. y Baskin, A. B. (1990) Learning companion systems. *En* C. Frasson y G. Gauthier (eds.), *Intelligent tutoring systems: At the crossroads of artificial intelligence in education*.

Chang, P. H.-M. et al. (2005) A knowledge-based scenario framework to support intelligent planning characters. *En IVA 2005: Inteligent Virtual Agents*, tomo 3661 de *Lecture Notes in Computer Science*, págs. 134–145. Springer-Verlag.

Chastek, G. y Ferguson, R. (2006) Toward Measures for Software Architectures. Inf. Téc. CMU/SEI-2006-TN-013, Software Engineering Institute - Carnegie Mellon University, Pittsburg, PA, USA.

Cheikes, B. A. (1995) GIA: An Agent-Based Architecture for Intelligent Tutoring Systems. *En* T. Finin y J. Mayfield (eds.), *Proceedings of the CIKM'95 Workshop on Intelligent Information Agents*. Baltimore, Maryland.

Choi, S. y Clark, R. E. (2006) Cognitive and affective benefits of animated pedagogical agents in learning English as a second language. *Journal of Educational Computing Research*, 33(2): págs. 455–480.

Chou, C.-Y. et al. (2003) Redefining the learning companion: the past, present, and future of educational agents. *Computers & Education*, 40(3): págs. 255–269.

Clements, P. (2005) Comparing the SEI's Views and Beyond Approach for Documenting Software Architectures with ANSI-IEEE 1471-2000. Inf. Téc. CMU/SEI-2005-TN-017, Software Engineering Institute - Carnegie Mellon University.

Clements, P. y Northrop, L. (2001) *Software Product Lines: Practices and Patterns*. The SEI Series in Software Engineering. Addison-Wesley Professional.

Clements, P. et al. (2002a) *Documenting Software Architectures: Views and Beyond*. The SEI Series in Software Engineering. Addison Wesley Professional, 1 ed$^{\underline{\text{ón}}}$.

Clements, P. et al. (2002b) *Evaluating Software Architectures: Methods and Case Studies*. The SEI Series in Software Engineering. Addison Wesley Professional, 1 ed$^{\underline{\text{ón}}}$.

Clements, P. C. (2000) Active Reviews for Intermediate Designs. Inf. Téc. CMU/SEI-2000-TN-009, Software Engineering Institute - Carnegie Mellon University, Pittsburg, PA, USA.

Collins, J. A. et al. (1997) Inspectable User Models for Just-In-Time Workplace Training. *En* A. Jameson, C. Paris y C. Tasso (eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, págs. 327–337. Springer.

Collis, J. C. et al. (1998) The ZEUS Agent Building Toolkit. *BT Technology Journal*, 16(3): págs. 60–68.

Cugini, J. et al. (2000) Design of 3D Visualization of Search Results: Evolution and Evaluation. *En Proceedings of IST/SPIE's 12th Annual International Symposium: Electronic Imaging 2000: Visual Data Exploration and Analysis*.

Dahmann, J. S. (1997) High Level Architecture for Simulation. *En DIS-RT '97: Proceedings of the 1st International Workshop on Distributed Interactive Simulation and Real-Time Applications*, pág. 9. IEEE Computer Society, Washington, DC, USA.

Das, T. K. et al. (1997) NetEffect: a network architecture for large-scale multi-user virtual worlds. *En VRST '97: Proceedings of the ACM symposium on Virtual reality software and technology*, págs. 157–163. ACM, New York, NY, USA.

DeLoach, S. A. et al. (2001) Multiagent Systems Engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3): págs. 231–258.

Demazeau, Y. et al. (2001) Systems Development as Societies of Agents. *En* J. Cuena, Y. Demazeau, A. Garcia y J. Treur (eds.), *Knowledge Engineering and Agent Technology*, Frontiers in Artificial Intelligence and Applications. IOS Press.

Dijkstra, E. W. (1959) A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1): págs. 269–271.

Drettakis, G. et al. (2007) Design and Evaluation of a Real-World Virtual Environment for Architecture and Urban Planning. *Presence: Teleoperators and Virtual Environments*, 16(3): págs. 318–332.

Elliott, C. D. et al. (1999) Lifelike Pedagogical Agents and Affective Computing: An Exploratory Synthesis. *Artificial Intelligence Today*, págs. 195–212.

Erl, T. (2005) *Service-Oriented Architecture. Concepts, Technology and Design*. Service Oriented Computing. Prentice Hall.

Erl, T. (2008) *SOA. Principles of Service Design*. Service Oriented Computing. Prentice Hall.

Etzioni, O. (1996) Moving Up the Information Food Chain: Deploying Softbots on the World Wide Web. *En* T. A. P. . T. M. Press (ed.), *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, págs. 1322–1326.

Evers, M. (2000) Jacob project - Documentation. Inf. téc., Department of Computer Science - University of Twente.

Evers, M. y Nijholt, A. (2000) Jacob - An animated instruction agent in virtual reality. *En* T. Tan, Y. Shi y W. Gao (eds.), *Advances in Multimodal Interfaces - ICMI 2000, Proceedings of the Third International Conference on Multimodal Interfaces*, nº 1948 en LNCS, págs. 526–533. Springer-Verlag.

Feynman, R. P. (2007) *¿Está usted de broma, Sr. Feynman?*. Alianza Editorial.

Finin, T. et al. (1993) Specification of the KQML Agent Communication Language.

Foster, I. y Kesselman, C. (eds.) (2004) *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2nd edón.

Foster, I. et al. (2002) Grid Services for Distributed Systems Integration. *IEEE Computer*, 35(6): págs. 37–46.

Foster, I. et al. (2004) Brain meets brawn: Why grid and agents need each other. *En Autonomous Agents and Multi-Agent Systems*, págs. 8–15.

Franklin, S. y Graesser, A. (1996) Is It an Agent, or Just a Program?: A Taxonomy for Autonomous Agents. *En Intelligent Agents III. Agent*

Theories, Architectures and Languages (ATAL'96), tomo 1193. Springer-Verlag.

Frederick P. Brooks, J. et al. (1990) Project GROPE - Haptic displays for scientific visualization. *En SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, págs. 177–185. ACM, New York, NY, USA.

Freedman, R. (2000) What is an Intelligent Tutoring System? *Intelligence*, 11(3): págs. 15–16.

Férey, N. et al. (2004) A Distributed Multimedia Database Visualization within an Immersive Environment for Bioinformatics. *En IEEE Sixth International Symposium on Multimedia Software Engineering*.

Furness, T. A. (1986) The super cockpit and its human factors challenges. *En Proceedings of the Human Factors Society*, tomo 30, págs. 48–52.

Gamma, E. et al. (1993) Design Patterns: Abstraction and Reuse of Object-Oriented Design. *En European Conference on Object-Oriented Programming Proceedings*, tomo 707 de *Lecture Notes in Computer Science*, págs. 406–431. Springer-Verlag.

Gamma, E. et al. (1995) *Design Patterns: Elements of Reusable Object-Oriented Software*. Professional Computing. Addison-Wesley.

Álvaro García y Medinilla, N. (2007) The Ambiguity Criterion in Software Design. *En International Workshop on Living with Uncertainties*.

Garijo, F. J. (2002) Tecnología de Agentes: Experiencias y perspectivas para el desarrollo de nuevos servicios y aplicaciones. *Bole.tic*, 24: págs. 1–9.

Gasmelseid, T. M. (2007) A Multiagent service-oriented Modeling of E-Government Initiatives. *International Journal of Electronic Government Research*, 3(3): págs. 87–106.

Genesereth, M. (1993) An agent-based approach to software interoperability. Inf. Téc. Logic–91–6, Logic Group, Computer Science Department, Stanford University.

Gherbi, R. y Hérisson, J. (2001) 3d Modelling tools for spatial-based in silico analysis of DNA. *In International Electronic Journal on Computer Graphics and Geometry*.

Gigante, M. (1993) Virtual Reality: Definitions, History and Applications. *En Virtual Reality Systems*, págs. 3–14. Academic-Press.

Giorgini, P. et al. (2004) The Tropos Methodology: an overview. *En Methodologies And Software Engineering For Agent Systems*. Kluwer Academic Publishing.

Gómez-Sanz, J. J. y Pavón, J. (2006) Implementing Multi-agent Systems Organizations with INGENIAS. *En Programming Multi-Agent Systems*, tomo 3862 de *LNCS*, págs. 236–251. Springer-Verlag.

Graesser, A. C. et al. (2001) Teaching Tactics and Dialog in AutoTutor. *International Journal of Artificial Intelligence in Education*, 12: págs. 257–279.

Graesser, A. C. et al. (2005a) AutoTutor: A cognitive system that simulates a tutor that facilitates learning through mixed-initiative dialogue. *En* C. Forsythe, M. Bernard y T. Goldsmith (eds.), *Cognitive systems: Human cognitive models in systems design*. Erlbaum, Mahwah, NJ, USA.

Graesser, A. C. et al. (2005b) AutoTutor: An intelligent tutoring system with mixed-initiative dialogue. *IEEE Transactions in Education*, 48: págs. 612–618.

Graesser, A. C. et al. (2008) The relationship between affect states and dialogue patterns during interactions with AutoTutor. *Journal of Interactive Learning Research*, 19: págs. 293–312.

Gratch, J. et al. (2002) Creating Interactive Virtual Humans: Some Assembly Required. *IEEE Intelligent Systems*, 17(4): págs. 54–63.

Greenhalgh, C. y Benford, S. (1995) MASSIVE: a distributed virtual reality system incorporating spatial trading. *En ICDCS '95: Proceedings of the 15th International Conference on Distributed Computing Systems*. IEEE Computer Society, Washington, DC, USA.

Grosz, B. J. y Kraus, S. (1996) Collaborative Plans for Complex Group Action. *Artificial Intelligence*, 86(2): págs. 269–357.

Ha, T. y Woo, W. (2006) Garden Alive: an Emotionally Intelligent Interactive Garden. *The International Journal of Virtual Reality*, 5(4): págs. 21–30.

Haddadi, A. y Sundermeyer, K. (1996) *Foundations of Distributed Artificial Intelligence*, cap. Belief-Desire-Intention Agent Architectures, págs. 169–186. Wiley-Interscience.

Hayden, S. C. et al. (1999) Architectural Design Patterns for Multi-Agent Coordination. *En Proceedings of the International Conference on Agent Systems '99. (Agents'99)*. Seattle, WA, USA.

Hayes-Roth, B. y Doyle, P. (1998) Animate characters. *Autonomous Agents and Multi-Agent Systems*, 1(2): págs. 195–230.

Hayes-Roth, R. (1995) An Architecture for Adaptive Intelligent Systems. *Artificial Intelligence: Special Issue on Agents and Interactivity*, 72: págs. 329–365.

Haythorn, W. (1994) What is Object-Oriented Design? *Journal of Object Oriented Programming*, 7(1): págs. 67–78.

He, Q. y Sycara, K. (1998) Towards a secure agent society. *En Proceedings of the Workshop on Deception, Fraud and Trust in Agent Societies, ACM AA'98*.

Heng, P. A. et al. (2006) A Haptic Needle Manipulation Simulator for Chinese Acupuncture Learning and Training. *International Journal of Image and Graphics*, 6(2): págs. 205–230.

Herrero, P. (2003) *Modelo de Percepción para Agentes Virtuales Inteligentes*. Tesis Doctoral, Facultad de Informática, Universidad Politécnica de Madrid.

Herrero, P. y de Antonio, A. (2005) Intelligent virtual agents keeping watch in the battlefield. *Virtual Reality*, 8(3): págs. 185–193.

Herrero, P. et al. (2005) Modelling the Sensory Abilities of Intelligent Virtual Agents. *Autonomous Agents and Multi-Agent Systems*, 11(3): págs. 361–385.

Hirose, M. (2006) Virtual Reality Technology and Museum Exhibit. *The International Journal of Virtual Reality*, 5(2): págs. 31–36.

Hodgins, W. (2002) IEEE Standard for Learning Object Metadata.

Hofmeister, C. et al. (2000) *Applied Software Architecture*. Addison Wesley.

Hospers, M. et al. (2004) *An Agent-based Intelligent Tutoring System for Nurse Education*, cap. 9, págs. 143–159. Whitestein Series in Software Agent Technologies. Birkhauser Publishing Ltd.

Hosseini, M. et al. (2002) A haptic virtual environment for industrial training. *IEEE International Workshop on Haptic Virtual Environments and Their Applications (HAVE)*, págs. 25–30.

Hérisson, J. et al. (2004) DNA in Virtuo: Visualization and Exploration of 3D Genomic Structures. *En 3rd ACM-SIGRAPH. International Conference on Virtual Reality, Computer Graphics, Visualization and Interaction*. Stellenbosch (Cape Town), South Africa.

Hughes, C. E. y Moshell, J. M. (1997) Shared Virtual Worlds for Education: the ExploreNet Experiment. *Multimedia Systems*, 5(2): págs. 145–154.

IEEE (2000) Recommended Practice for Architectural Description of Software-Intensive Systems.

IEEE (2002) IEEE 730-2002 Standard for Software Quality Assurance Plans.

IEEE-FIPA (2002a) FIPA Abstract Architecture Specification.

IEEE-FIPA (2002b) FIPA ACL Message Structure Specification.

Iglesias, C. et al. (1998) Analysis and design of multiagent systems using MAS-CommonKADS. *En Inteligent Agents IV*, tomo 1365 de *LNAI*. Springer-Verlag.

Imbert, R. (2005) *Una Arquitectura Cognitiva Multinivel para Agentes con Comportamiento Influido por Características Individuales y Emociones, Propias y de Otros Agentes*. Tesis Doctoral, Facultad de Informática, Universidad Politécnica de Madrid.

Ivezic, N. et al. (2000) An Analysis of a Supply Chain Management Agent Architecture. *En Fourth International Conference on Multi-Agent Systems (ICMAS'00)*, pág. 401.

Jackson, R. L. y Fagan, E. (2000) Collaboration and Learning within Immersive Virtual Reality. *En Proceedings of the Third International Conference on Collaborative Virtual Environments*, págs. 83–92. ACM.

Jiang, Y. C. et al. (2005) Autonomous trust construction in multi-agent systems: a graph theory methodology. *Advances in Engineering Software*, 36(2): págs. 59–66.

Johnson, A. et al. (1998a) The NICE Project: Learning Together in a Virtual World. *En Proceedings of VRAIS'98*, págs. 176–183. IEEE.

Johnson, A. et al. (1999a) Learning and Building Together in an Immersive Virtual World. *Presence: Teleoperators and Virtual Environments*, 8(3): págs. 247–263. Special Issue on Virtual Environments and Learning.

Johnson, A. et al. (1999b) The Round Earth Project: Deep Learning in a Collaborative Virtual World. *En Proceedings of Virtual Reality*, págs. 164–171. IEEE.

Johnson, W. L. (2007) Serious Use of a Serious Game for Language Learning. *En* R. Luckin, K. R. Koedinger y J. Greer (eds.), *Artificial Intelligence in Education - Building Technology Rich Learning Contexts That Work*, tomo 158 de *Frontiers in Artificial Intelligence and Applications*, págs. 67–74. IOS Press.

Johnson, W. L. y Valente, A. (2008) Tactical Language and Culture Training Systems: Using Artificial Intelligence to Teach Foreign Languages and Cultures. *En Proceedings of IAAI 2008*.

Johnson, W. L. et al. (1998b) Integrating Pedagogical Agents into Virtual Environments. *Presence: Teleoperators and Virtual Environments*, 7(6): págs. 523–546.

Johnson, W. L. et al. (2000) Animated Pedagogical Agents: Face-to-Face Interaction in Interactive Learning Environments. *International Journal of Artificial Intelligence in Education*, 11: págs. 47–78.

Jonquet, C. et al. (2008) Agent-Grid Integration Language. *Multiagent and Grid Systems*, 4(2): págs. 167–211.

Josuttis, N. M. (2007) *SOA in Practice. The Art of Distributed System Design*. O'Reilly.

Kazman, R. et al. (1994) SAAM: A Method for Analyzing the Properties Software Architectures. *En Proceedings of the 16th International Conference on Software Engineering*, págs. 81–90. Sorrento, Italy.

Kazman, R. et al. (2000) ATAM: Method for Architecture Evaluation. Inf. Téc. CMU/SEI-2000-TR-004, Software Engineering Institute - Carnegie Mellon University, Pittsburg, PA, USA.

Kazman, R. et al. (2002) Making Architecture Design Decisions: An Economic Approach. Inf. Téc. CMU/SEI-2002-TR-035, Software Engineering Institute - Carnegie Mellon University, Pittsburg, PA, USA.

Kazman, R. et al. (2004) Integrating Software-Architecture-Centric Methods into the Rational Unified Process. Inf. Téc. CMU/SEI-2004-TR-011, Software Engineering Institute - Carnegie Mellon University, Pittsburg, PA, USA.

Khoualdi, K. y Benghezal, R. (2004) Design of an Intelligent Tutor using a Multiagent Approach. *Transactions on Engineering, Computing and Technology*, 3: págs. 25–28.

Kim, Y. et al. (2005) Controlling the focus of perceptual attention in embodied conversational agents. *En AAMAS*, págs. 1097–1098.

Kinny, D. et al. (1996) A Methodology and Modelling Technique for Systems of BDI Agents. *En Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, tomo 1038 de *LNCS*, págs. 56–71.

Kirner, T. G. et al. (2001) Development of a Collaborative Virtual Environment for Educational Applications. *En Proceedings of the 2001*

Conference on 3D Technologies for the World Wide Web, págs. 61–68. ACM.

Klein, M. y Kazman, R. (1999) Attribute-Based Architectural Styles. Inf. Téc. CMU/SEI-99-TR-022, Software Engineering Institute - Carnegie Mellon University, Pittsburg, PA, USA.

Klein, M. H. et al. (1999) Attribute-Based Architecture Styles. *En Software Architecture, Proceedings of the First Working IFIP Conference on Software Architecture (WICSA1)*, págs. 225–243. Kluwer Academic Publishers, San Antonio, Texas, USA.

Kolp, M. et al. (2002) A Goal-Based Organizational Perspective on Multi-agent Architectures. *En ATAL '01: Revised Papers from the 8th International Workshop on Intelligent Agents VIII*, págs. 128–140. Springer-Verlag, London, UK.

Kolp, M. et al. (2003) Organizational patterns for early requirements analysis. *En 15th International Conference on Advanced Information Systems Engineering (CAiSE'03*, págs. 617–632.

Kolp, M. et al. (2006) Multi-Agent Architectures as Organizational Structures. *Autonomous Agents and Multi-Agent Systems*, 13(1): págs. 3–25.

Korze, D. et al. (2001) Virtual delivery room as a teaching tool. *En Proceedings of the 9th Mediterranean Conference on Medical and Biological Engineering and Computing (MEDICON 2001)*, tomo 1, págs. 476–79. Pula, Croacia.

Kroll, P. y Kruchten, P. (2003) *The Rational Unified Process Made Easy. A Practitioner's Guide to the RUP*. Object Technology. Addison Wesley.

Kruchten, P. (1995) The 4+1 View Model of Architecture. *IEEE Software*, 6(12): págs. 45–50.

Kruchten, P. et al. (2006) The Past, Present and Future of Software Architecture. *IEEE Software*, 23(2): págs. 22–30.

Kuljis, J. (2002) A distributed virtual collaborative learning environment. *En Proceedings of AI 2002. 20th IASTED International Multiconference on Applied Informatics*, págs. 464–69. Innsbruck, Austria.

Labrou, Y. y Finin, T. (1997) A Proposal for a new KQML Specification. Inf. Téc. TR CS-97-03, UMBC, Baltimore, MD 21250.

Larijani, L. C. (1994) *The Virtual Reality Primer*. McGraw Hill.

Lester, J. y Stone, B. (1997) Increasing believability in animated pedagogical agents. *En Proceedings of the First International Conference on Autonomous Agents*, págs. 16–21.

Lester, J. et al. (1997a) Animated pedagogical agents and problem solving efectiveness: A large-scale empirical evaluation. *En Proceedings of Eighth World Conference on Artificial Intelligence in Education*, págs. 23–30.

Lester, J. et al. (1997b) Cosmo: A life-like animated pedagogical agent with deictic believability. *En IJCAI97 Workshop on Animated Interface Agents: Making them Intelligent*. Nagoya, Japan.

Lester, J. et al. (1997c) The persona effect: Affective impact of animated pedagogical agents. *En Proceedings of CHI 97*, págs. 359–366. ACM.

Lester, J. et al. (1999a) Deictic Believability: Coordinated Gesture, Locomotion and Speech in Lifelike Pedagogical Agents. *Applied Artificial Intelligence*, 13(4-5): págs. 383–414.

Lester, J. C. et al. (1999b) Explanatory lifelike avatars: performing user-centered tasks in 3D learning environments. *En Proceedings of the Third Annual Conference on Autonomous Agents*, págs. 24–31.

Lester, J. C. et al. (1999c) Lifelike Pedagogical Agents for Mixed-Initiative Problem Solving in Constructivist Learning Environments. *User Modeling and User-Adapted Interaction*, 9(1-2): págs. 1–44.

Li, Z. et al. (2006) A Survey of Emergent Behavior and Its Impacts in Agent-based Systems. *En IEEE International Conference on Industrial Informatics, 2006*, págs. 1295–1300.

van Liere, R. et al. (2002) Virtual reality in biological microscopic imaging. *En Proceedings of IEEE International Symposium on Biomedical Imaging*, págs. 879–882.

Loftin, R. B. et al. (2004) Training in Peacekeeping Operations Using Virtual Environments. *IEEE Computer Graphics and Applications*, 24(4): págs. 18–21.

Lozano, A. et al. (2003) Virtool-D: Entorno virtual educativo para aprendizaje en dominios procedimentales. *En Proceeings of SIIE 2003, International Symposium in Education Computing*.

Lèty, E. et al. (2004) SCORE: a scalable communication protocol for large-scale virtual environments. *IEEE/ACM Transactions on Networking*, 12(2): págs. 247–260.

Lucas, S. et al. (2008) Virtual Reality Training Improves Simulated Laparoscopic Surgery Performance in Laparoscopy Naïve Medical Students. *Journal of Endourology*, 22(5): págs. 1047–1052.

Maes, P. (1995) Artificial Life Meets Entertainment: Life like Autonomous Agents. *Communications of the ACM*, 38(11): págs. 108–114.

Magnenat-Thalmann, N. y Kalra, P. (1995) *The simulation of a virtual TV presenter*, págs. 9–21. World Scientific Press, Singapore.

Marsella, S. y Johnson, W. L. (1997) An instructor's assistant for team-teaching in dynamic multi-agent virtual worlds. *En Proceedings of the Fourth International Conference on Intelligent Tutoring Systems*, LNCS. Springer-Verlag.

Martin, R. C. (1999) *Designing Object Oriented Applications using UML*. Prentice Hall, 2 ed$^{\underline{\text{ón}}}$.

Mas, A. (2005) *Agentes Software y Sistemas Multi-Agente: Conceptos, Arquitecturas y Aplicaciones*. Prentice Hall.

McQuiggan, S. W. et al. (2008) Story-Based Learning: The Impact of Narrative on Learning Experiences and Outcomes. *En Intelligent Tutoring Systems*, tomo 5091 de *Lecture Notes in Computer Science*, págs. 530–539. Springer Berlin / Heidelberg.

Medinilla, N. y Gutiérrez, I. (2007) La Incertidumbre como Herramienta en la Ingeniería de Software. *IEEE Latin America Transactions*, 5(4): págs. 265–270.

Méndez, G. et al. (2001) PRVIR: An Integration Between an Intelligent Tutoring System and a Virtual Environment. *En SCI2001*, tomo VIII, págs. 175–180. IIIS, IEEE Computer Society, Orlando, FL.

Méndez, G. et al. (2003) Steve Meets Jack: the Integration of an Intelligent Tutor and a Virtual Environment with Planning Capabilities. *En Intelligent Virtual Agents. 4th International Workshop, IVA03*, nº 2792 en LNAI, págs. 325–332. Springer-Verlag.

Méndez, G. et al. (2004) Intelligent Virtual Environments for Training in Nuclear Power Plants. *En ICEIS 2004, Proceedings of the 6th International Conference on Enterprise Information Systems*, págs. 204–209. Oporto, Portugal.

Moreno, R. et al. (2001) The case for social agency in computer-based teaching. Do students learn more deeply when they interact with animated pedagogical agents? *Cognition and Instruction*, 19(2): págs. 177–213.

Moya, L. J. y Tolk, A. (2007) Towards a taxonomy of agents and multi-agent systems. *En Proceedings of the 2007 Spring Simulation Multiconference, SpringSim 2007*, tomo 2, págs. 11–18. SCS/ACM.

Muller-Wittig, W. K. et al. (2001) LAHYSTOTRAIN - VR-based Intelligent Training Environment for Laparoscopy and Hysteroscopy. *En* S. Richir (ed.), *Virtual Reality International Conference (VRIC) 2001. Proceedings*, págs. 225–233.

Neo, H. K. et al. (2005) A Grid-Based Mobile Agent Collaborative Virtual Environment. *En CW'05: Proceedings of the 2005 International Conference on Cyberworlds*, págs. 335–339. IEEE Computer Society, Washington, DC, USA.

Núñez, A. (2008) Proyecto ENVIRA: Desarrollo de una Arquitectura de Comunicación para Entornos Virtuales Distribuidos de Entrenamiento. Proyecto de Fin de Carrera, Laboratorio Decoroso Crespo, Facultad de Informática - UPM.

Nilsson, N. J. (1971) *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill.

Noma, T. y Badler, N. (1997) A virtual human presenter. *En Proceedings of the IJCAI '97 Workshop on Animated Interface Agents: Making them Intelligent*, págs. 45–51.

Nord, R. L. et al. (2004) Integrating the Quality Attribute Workshop (QAW) and the Attribute-Driven Design (ADD) Method. Inf. Téc. CMU/SEI-2004-TN-017, Software Engineering Institute - Carnegie Mellon University, Pittsburg, PA, USA.

Odell, J. (2000) Agent Technology. Green Paper. Inf. Téc. OMG Document ec/2000-08-01, Version 1.0, Object Management Group - Agent Working Group.

Odell, J. (2002) Objects and Agents Compared. *Journal of Object Technology*, 2(2): págs. 41–53.

OMG (2008) UML Profile and Metamodel for Services (SOA-Pro). Request for Proposals ad/2008-05-03, Object Management Group.

Paiva, A. y Machado, I. (1998) Vincent, an Autonomous Pedagogical Agent for On-the-Job Training. *En* B. P. Goettl, H. M. Halff, C. L. Redfield y V. J. Shute (eds.), *ITS '98: Proceedings of the 4th International Conference on Intelligent Tutoring Systems*, nº 1452 en LNCS, págs. 584–593. Springer-Verlag, London, UK.

Paiva, A. y Machado, I. (2002) Life-Long Training with Vincent, a Web-based Pedagogical Agent. *International Journal of Continuing Engineering Education and Life-Long Learning*, 12(1).

Pantelidis, V. S. (1996) Suggestions on When to Use and When Not to Use Virtual Reality in Education. *VR in the Schools*, 2(1): pág. 18.

Park, J. S. et al. (2005) Visible Korean Human: Improved Serially Sectioned images of the entire body. *IEEE Transactions on Medical Imaging*, 24: págs. 352–360.

Parnas, D. L. (1972) On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12): págs. 1053 – 1058.

Parnas, D. L. y Weiss, D. (1985) Active Design Reviews: Principles and Practice. *En Proceedings, Eighth International Conference on Software Engineering*, págs. 132–136.

Parnas, D. L. et al. (1984) The Modular Structure of Complex Systems. *En ICSE '84: Proceedings of the 7th International Conference on Software Engineering*, págs. 408–417. IEEE Press, Piscataway, NJ, USA.

Parunak, H. V. D. y Brueckner, S. (2001) Entropy and self-organization in multi-agent systems. *En AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, págs. 124–130. ACM, New York, NY, USA.

Pavón, J. y Gómez-Sanz, J. J. (2003) Agent Oriented Software Engineering with INGENIAS. *En Multi-Agent Systems and Applications III, 3rd International Central and Eastern European Conference on Multi-Agent Systems, (CEEMAS 2003)*, págs. 394–403.

Person, N. K. et al. (2001) Simulating human tutor dialog moves in AutoTutor. *International Journal of Artificial Intelligence in Education*, 12: págs. 23–29.

Piaget, J. (1954) *The Construction of Reality in the Child*. New York: Basic Books.

Popovici, D. M. et al. (2005) VirtualDive - a VR-based educational virtual environment. *Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2005)*.

Poslad, S. y Calisti, M. (2000) Towards improved trust and security in FIPA agent platforms. *En Proceedings of Autonomous Agents 2000 Workshop on Deception, Fraud and Trust in Agent Societies*.

Pressman, R. S. (2004) *Software Engineering: A Practitioner's Approach*. Computer Science. McGraw-Hill, 6 ed$^{\underline{\text{ón}}}$.

Querrec, R. et al. (2004) Multiagents Systems for Virtual Environment for Training. Application to Fire-Fighting. *Advanced Technology for Learning*, 1(1): págs. 25–34.

Ramdane-Cherif, A. et al. (2005) The Platform Based-Agents to Test and Evaluate Software Architecture. *Journal of Object Technology*, 4(1): págs. 67–82.

Randell, B. y Buxton, J. (eds.) (1970) *Software Engineering Techniques: Report of a Conference Sponsored by the NATO Science Committee*. Scientific Affairs Division, NATO.

Rickel, J. y Johnson, W. L. (1997) Intelligent Tutoring in Virtual Reality: A Preliminary Report. *En Proceedings of the International Conference on Artificial Intelligence in Education*, págs. 294–301.

Rickel, J. y Johnson, W. L. (1999) Animated Agents for Procedural Training in Virtual Reality: Perception, Cognition and Motor Control. *Applied Artificial Intelligence*, 13: págs. 343–382.

Rickel, J. y Johnson, W. L. (2000) *Task Oriented Collaboration with Embodied Agents in Virtual Worlds*. MIT Press.

Rickel, J. y Johnson, W. L. (2002) Extending Virtual Humans to Support Team Training in Virtual Reality. *En* G. Lakemayer y B. Nebel (eds.), *Exploring Artificial Intelligence in the New Millenium*, págs. 217–238. Morgan Kaufmann, San Francisco.

Rickel, J. et al. (2002a) Collaborative Discourse Theory as a Foundation for Tutorial Dialogue. *En* S. A. Cerri, G. Gouardères y F. Paraguaçu (eds.), *Proceedings of the Sixth International Conference on Intelligent Tutoring Systems*, tomo 2363 de *LNCS*, págs. 542–551. Springer-Verlag.

Rickel, J. et al. (2002b) Toward a New Generation of Virtual Humans for Interactive Experiences. *IEEE Intelligent Systems*, 17(4): págs. 32–38.

Rodríguez, D. (2008) Proyecto ENVIRA: Arquitectura de un Entorno Virtual Colaborativo para Sistemas Inteligentes de Entrenamiento. Proyecto de Fin de Carrera, Laboratorio Decoroso Crespo, Facultad de Informática - UPM.

Russell, S. J. y Norvig, P. (1995) *Artificial Intelligence: A Modern Approach*. Prentice Hall.

dos Santos, C. T. et al. (2002) DÓRIS – Pedagogical Agent in Intelligent Tutoring Systems. *En* S. A. Cerri, G. Gouardères y F. Paraguaçu (eds.), *Proceedings of the 6th International Conference on Intelligent Tutoring Systems*, nº 2363 en LNCS, págs. 91–104. Springer-Verlag.

Schreiber, G. et al. (1999) *Knowledge Engineering and Management. The CommonKADS Methodology*. MIT Press.

Shaw, E. et al. (1999a) Building a Case for Agent-Assisted Learning as a Catalyst for Curriculum Reform in Medical Education. *En* S. Lajoie y M. Vivet (eds.), *Proceedings of the 9th World Conference on Artificial Intelligence in Education AIED'99, Frontiers in Artificial Intelligence and Applications*, tomo 50, págs. 509–516. IOS Press.

Shaw, E. et al. (1999b) Pedagogical agents on the Web. *En Proceedings of the Third Annual Conference on Autonomous Agents*, págs. 283–290. ACM Press, Seattle, WA, USA.

Shaw, M. y Clements, P. (2006a) The Golden Age of Software Architecture. *IEEE Software*, 23(2): págs. 31–39.

Shaw, M. y Clements, P. (2006b) The Golden Age of Software Architecture: A Comprehensive Survey. Inf. Téc. CMU-ISRI-06-101, Institute for Software Research International, Carnegie Mellon University, Pittsburgh, PA, USA.

Silva, C. et al. (2005) Describing Agent-Oriented Design Patterns in Tropos. *En Proceedings of 19th SBES*, págs. 10–25.

Simon, H. A. (1969) *The Architecture of Complexity: Hierarchic Systems*, cap. 8, págs. 183–217. MIT Press.

Sleeman, D. H. y Brown, J. (eds.) (1982) *Intelligent Tutoring Systems*. Academic Press, London.

Smith, D. C. et al. (1994) KidSim: Programming Agents Without a Programming Language. *Communications of the ACM*, 37(7): págs. 55–67.

Sánchez, L. (2006) Proyecto MAEVIF: Diseño e Implementación de un Módulo de Tutoría Basado en Agentes Software. Proyecto de Fin de Carrera, Laboratorio Decoroso Crespo, Facultad de Informática - UPM.

Sánchez, M. I. (2001) *Senda3D. Una Aproximación Metodológica al Desarrollo de Entornos Virtuales*. Tesis Doctoral, Facultad de Informática, Universidad Politécnica de Madrid.

Sommerville, I. (2007) *Software Engineering*. Addison-Wesley, 8 ed$^{\underline{\text{ón}}}$.

Stone, B. y Lester, J. (1996) Dynamically sequencing an animated pedagogical agent. *En Proceedings of the Thirteenth National Conference on Artificial Intelligence*, págs. 424–431. Portland, Oregon, USA.

Sud, A. et al. (2007) Real-time Path Planning for Virtual Agents in Dynamic Environments. *En Virtual Reality Conference, 2007. VR '07. IEEE*, págs. 91–98.

Sud, A. et al. (2008) Real-Time Path Planning in Dynamic Virtual Environments Using Multiagent Navigation Graphs. *IEEE Transactions on Visualization and Computer Graphics*, 14(3): págs. 526–538.

Sunar, M. S. et al. (2006) Effective Range Detection Approach for Ancient Malacca Virtual Walkthrough. *The International Journal of Virtual Reality*, 5(4): págs. 31–38.

Suppes, P. (1967) Some theoretical models for mathematics learning. *Journal of Research and Development in Education*, 1: págs. 5–22.

Sutherland, I. et al. (1971) Moving Computer Graphic Images Seen from Inside the Vascular System. *Transactions of the American Society of Artificial Internal Organs*, 17: págs. 381–385.

Sutherland, I. E. (1965) The Ultimate Display. *En Proceedings of IFIPS Congress*, tomo 2, págs. 506–508. New York.

Swartout, W. et al. (2001) Towards the Holodeck: Integrating Graphics, Sound, Character and Story. *Fifth International Conference on Autonomous Agents*, págs. 409–416.

Swartout, W. R. et al. (2006) Toward Virtual Humans. *AI Magazine*, 27(2): págs. 96–108.

Toro, C. et al. (2006) Improving Virtual Reality Applications in CAD through Semantics. *The International Journal of Virtual Reality*, 5(4): págs. 39–46.

Traum, D. et al. (2003) Negotiation over Tasks in Hybrid Human-Agent Teams for Simulation-Based Training. *En Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, págs. 441–448. ACM Press.

Uhr, L. (1969) Teaching machine programs that generate problems as a function of interaction with students. *En Proceedings of the 1969 24th ACM National Conference*, págs. 125–134. ACM, New York, NY, USA.

Vassileva, J. et al. (1999) A Multi-agent Approach to the Design of Peer-help Environments. *En* S. Lajoie y M. Vivet (eds.), *Artificial Intelligence and Education*, págs. 38–45. IOS Press.

Vassileva, J. et al. (2003) Multi-Agent Multi-User Modeling in I-Help. *User Modeling and User-Adapted Interaction*, 13(1-2): págs. 179–210.

Viciana-Abad, R. y Reyes-Lecuona, A. (2005) Patient Modelling Using Expert Systems for Medical Training Simulations Based on Virtual Reality. *En 7th International Conference on Virtual Reality, VRIC - LAVAL VIRTUAL.*

Voronoi, G. (1907) Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *Journal für die Reine und Angewandte Mathematik*, 133: págs. 97–178.

Voss, G. et al. (2000) LAHYSTOTRAIN - Intelligent Training System for Laparoscopy and Hysteroscopy. *En* J. Westwood, H. Hoffman y G. Mogel (eds.), *Proceedings of Medicine Meets Virtual Reality*, págs. 359–364. IOS Press.

Wazlawick, R. S. et al. (1999) Virtual Museum: An Authory Tool for the Creation of Virtual Reality Museums to Support the Collaborative Learning on the Internet. Inf. téc., ProTeM Project.

Webber, C. y Pesty, S. (2002) A two-level multi-agent architecture for a distance learning environment. *En* E. de Barros Costa (ed.), *Workshop on Architectures and Methodologies for Building Agent-based Learning Environments (ITS 2002)*, págs. 26–38.

Webber, C. et al. (2001) Baghera project: a multi-agent architecture for human learning. *En Proceedings of the Workshop Multi-Agent Architectures for Distributed Learning Environments (AIED2001)*, págs. 12–17.

Webber, C. et al. (2002) A multi-agent and emergent approach to learner modelling. *En* F. van Harmelen (ed.), *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002)*. IOS Press.

Webster, B. F. (1995) *Pitfalls of Object-Oriented Development*. M&T Books.

Weiss, G. (ed.) (1999) *Multiagent Systems. A Moder Approach to Distributed Artificial Intelligence*. MIT Press.

Wenger, E. (1987) *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge*. Morgan Kaufmann Publishers.

Weyns, D. et al. (2006) Multiagent systems as software architecture: another perspective on software engineering with multiagent systems. *En AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, págs. 1314–1316. ACM, New York, NY, USA.

Wiemer-Hastings, P. et al. (1998) The foundations and architecture of AutoTutor. *En Proceedings of the 4th International Conference on Intelligent Tutoring Systems*, págs. 334–343. Springer-Verlag, Berlin.

Winn, W. (1993) A Conceptual Basis for Educational Applications of Virtual Reality. Tech Report R-93-9, University of Washington, Human Interface Technology Laboratory.

Wirth, N. (1971) Program development by stepwise refinement. *Communications of the ACM*, 14(4): págs. 221–227.

Wojcik, R. et al. (2006) Attribute-Driven Design (ADD), Version 2.0. Inf. Téc. CMU/SEI-2006-TR-023, Software Engineering Institute - Carnegie Mellon University, Pittsburg, PA, USA.

Woods, S. G. y Barbacci, M. R. (1999) Architectural Evaluation of Collaborative Agent-Based Systems. Inf. Téc. CMU/SEI-99-TR-025, Software Engineering Institute - Carnegie Mellon University, Pittsburg, PA, USA.

Wooldridge, M. (1997) Agent-based Software Engineering. *IEE Transactions on Software Engineering*, 144(1): págs. 26–37.

Wooldridge, M. y Jennings, N. R. (1995a) Agent Theories, Architectures, and Languages: a Survey. *Intelligent Agents*, págs. 1–22.

Wooldridge, M. y Jennings, N. R. (1995b) Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2): págs. 115–152.

Wooldridge, M. y Jennings, N. R. (1998) Pitfalls of Agent-Oriented Development. *En* K. P. Sycara y M. Wooldridge (eds.), *Proceedings of the Second International Conference on Autonomous Agents*, págs. 385–391. ACM, ACM Press.

Wooldridge, M. y Jennings, N. R. (1999) Software Engineering with Agents: Pitfalls and Pratfalls. *IEEE Internet Computing*, 3(3): págs. 20–27.

Xiang, L. (2007) A Multi-Agent-Based Service-Oriented Architecture for Inter-Enterprise Cooperation System. *En Second International Conference on Digital Telecommunications*.

Youngblut, C. (1998) Educational Uses of Virtual Reality Technology. Inf. téc., Institute for Defense Analyses.

Zambonelli, F. et al. (2003) Developing Multiagent Systems: The Gaia Methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3): págs. 317–370.

Zhang, D. M. y Alem, L. (1997) From task sequencing to curriculum planning. *En Proceedings of AI-ED'97*.

Zhang, D. M. et al. (1998) Using Multi-agent Approach for the Design of an Intelligent Learning Environment. *En* C. Z. Wayne Wobcke, Maurice Pagnucco (ed.), *Proceedings of the Workshops on Commonsense Reasoning, Intelligent Agents, and Distributed Artificial Intelligence: Agents and Multi-Agent Systems Formalisms, Methodologies, and Applications*, nº 1441 en LNCS, págs. 221–230. Springer-Verlag.

Zhang, L. y Lin, Q. (2007) MACVE: A Mobile Agent Based Framework for Large-Scale Collaborative Virtual Environments. *Presence: Teleoperators and Virtual Environments*, 16(3): págs. 279–292.

Zhu, T. et al. (2008) Plant Modeling Based on 3D Reconstruction and Its Application in Digital Museum. *The International Journal of Virtual Reality*, 7(1): págs. 81–88.