UNIVERSIDAD POLITÉCNICA DE MADRID

FACULTAD DE INFORMÁTICA

TRABAJO FIN DE CARRERA

INTERACTIVIDAD EN UN ENTORNO VIRTUAL INTELIGENTE:

MANEJO DE DISPOSITIVOS DE REALIDAD VIRTUAL Y GESTIÓN DE INTERACCIONES

AUTOR: **Raúl Rodríguez Juárez**

TUTORES: **Angélica de Antonio Jiménez**

Gonzalo Méndez Pozo

Todo lo que he admitido hasta el presente como más seguro y verdadero, lo he aprendido de los sentidos o por los sentidos; ahora bien, he experimentado a veces que tales sentidos me engañaban, y es prudente no fiarse nunca por entero de quienes nos han engañado una vez.

R. DESCARTES, Meditaciones.

Índice

1. INTRODUCCIÓN 9

1.1. PLANTEAMIENTO INICIAL 9
1.2. OBJETIVOS 12
1.3. ESTRUCTURA DEL TRABAJO 13

2. ESTADO DE LA CUESTIÓN 15

2.1. ANTECEDENTES Y DEFINICIONES PREVIAS 15
2.2. MOTIVACIONES PARA EL DESARROLLO DE ENTORNOS VIRTUALES PARA EL ENTRENAMIENTO 16
2.3. TECNOLOGÍAS RELACIONADAS CON ENTORNOS VIRTUALES 19
2.4. INTERFACES DE ACCESO PARA REALIDAD VIRTUAL 22
2.5. ÁREAS DE INVESTIGACIÓN SOBRE ENTORNOS VIRTUALES 31
2.5.1. Inteligencia en VETs 32
2.5.2. Metodologías de desarrollo de sistemas multi-agente 36
2.6. TÉCNICAS DE INTEGRACIÓN EN LA CONSTRUCCIÓN DE ENTORNOS VIRTUALES 45

3. DESCRIPCIÓN GENERAL DEL PROYECTO 47

3.1. DESCRIPCIÓN DEL PROYECTO MAEVIF 47
3.2. SUB-MODELO PARA LA GESTIÓN DE INTERFACES DE ACCESO E INTERACCIONES EN MAEVIF. . 56

4. DISPOSITIVOS DE REALIDAD VIRTUAL 59

4.1. INTRODUCCIÓN 59
4.2. SENSORES DE POSICIONAMIENTO Y ORIENTACIÓN EN EL ESPACIO 60
4.2.1. Introducción 60
4.2.2. Arquitectura Hardware 67
4.2.3. Arquitectura Software 71
4.2.4. Desarrollo de una librería software genérica para Flock of Birds 74
4.2.4.1. Diseño de bajo nivel 75
4.2.4.2. Implementación 83
4.3. DISPOSITIVO DE INTERACCIÓN MANUAL CON UN ENTORNO VIRTUAL 85
4.3.1. Introducción 85
4.3.2. Arquitectura Hardware 86
4.3.3. Arquitectura Software 89
4.3.4. Desarrollo de una librería genérica para Immersion CyberGlove 90
4.3.4.1. Diseño de bajo nivel 90
4.3.4.2. Implementación 92
4.4. DISPOSITIVOS DE NAVEGACIÓN: JOYSTICKS 93
4.4.1. Introducción 93

4.4.2. Arquitectura Hardware *95*
4.4.3. Arquitectura Software *96*
4.4.4. Diseño de una librería software genérica *97*
4.4.4.1. Diseño de bajo nivel 98
4.4.4.2. Implementación 101
4.5. DESARROLLO DE UN MÓDULO SOFTWARE PARA LA INTEGRACIÓN DE DISPOSITIVOS HETEROGÉNEOS EN MAEVIF 103
4.5.1. Análisis *105*
4.5.2. Diseño de alto nivel *112*
4.5.3. Diseño detallado *122*
4.5.4. Implementación *143*

5. GESTIÓN DE INTERACCIONES EN UN IVET 146

5.1. INTRODUCCIÓN 146
5.2. AGENTES DE ACCIÓN EN MAEVIF 146
5.3. AGENTE DE ACTUACIÓN 147
5.3.1. Introducción *147*
5.3.1.1. Faceta de planificación 148
5.3.1.2. Faceta de simulación 151
5.3.1.3. Mecanismos de comunicación: Pizarras 152
5.3.2. Fase de análisis *154*
5.3.2.1. Modelo de roles prototipo 154
5.3.2.2. Modelo de interacción 155
5.3.2.3. Modelo de roles completamente elaborado 162
5.3.3. Fase de diseño de alto nivel *165*
5.3.3.1. Modelo de agentes 165
5.3.3.2. Modelo de servicios 166
5.3.3.3. Modelo de conocidos 168
5.3.4. Fase de diseño detallado *168*
5.3.4.1. Definición de mensajes ACL 168
5.3.4.2. Diagrama de clases de diseño 172
5.3.4.3. Definición de los comportamientos JADE 183
5.3.5. Fase de implementación *188*

6. CONCLUSIONES 193

7. TRABAJO FUTURO 195

8. BIBLIOGRAFÍA 199

Índice de figuras

Figura 1: Lentes LCD 23

Figura 2: HMD 24

Figura 3: BOOM 25

Figura 4: CAVE 26

Figura 5: Plataforma de movimiento 28

Figura 6: Guante 28

Figura 7: Arquitectura de la plataforma JADE de agentes FIPA. 39

Figura 8: Plataforma JADE distribuida 40

Figura 9: Ciclo de vida de un agente 40

Figura 10: Modelos en GAIA 43

Figura 11: Representación de protocolos 44

Figura 12: Arquitectura de un ITS 49

Figura 13: Arquitectura ITS extendida para un IVET 51

Figura 14: Arquitectura basada en agentes para IVETs. 53

Figura 15: Flock of Birds 61

Figura 16: Sistema cartesiano de referencia en FOB 62

Figura 17: Ángulos de orientación en FOB 63

Figura 18: Sistemas cartesianos *left-hand* y *right-hand* 64

Figura 19: Arquitectura hardware en FOB 67

Figura 20: Conexión usando una interfaz FBB 68

Figura 21: Conexión usando una interfaz RS232 69

Figura 22: Conexión usando una interfaz RS232 individual por cada Bird 69

Figura 23: Conexión usando una interfaz FBB con ERC/ERT 70

Figura 24: Esquema electrónico de FOB 71

Figura 25: Arquitectura software para el uso de FOB 72

Figura 26: Módulo de acceso a FOB 74

Figura 27: Diagrama de clases de diseño para el módulo de acceso a FOB 75

Figura 28: Anatomía de la mano 86

Figura 29: Sensores en CyberGlove 86

Figura 30: Representación virtual de la mano 87

Figura 31 : Arquitectura software para el uso de CyberGlove 89

Figura 32: Diagrama de clases de diseño para el módulo de acceso a CyberGlove 90

Figura 33: Logitec Force 3D 95

Figura 34: Arquitectura Software para el uso de un Joystick 97

Figura 35: Diagrama de clases de diseño para el módulo de acceso a un joystick 98

Figura 36: Grafo de combinaciones posibles en la selección de dispositivos....104

Figura 37: Diagrama de casos de uso 106

Figura 38: Modelo conceptual....110

Figura 39: Descomposición en paquetes 111

Figura 40: Operación “Inicializar” 117

Figura 41: Operación “Seleccionar Dispositivos” 118

Figura 42: Operación “Sondear Dispositivos” 119

Figura 43: Operación “Procesar Evento” 120

Figura 44: Operación “Repintar”....121

Figura 45: Pantalla de selección de dispositivos 123

Figura 46: Diagrama de clases del prototipo 124

Figura 47: Diagrama de clases refinado 128

Figura 48: Sondear dispositivos 138

Figura 49: Diagrama de actividad para el sondeo de dispositivos de posicionamiento....139

Figura 50: Diagrama de actividad para el sondeo de dispositivos de navegación 140

Figura 51: Seleccionar dispositivos....141

Figura 52: Repintar....141

Figura 53: Evento de teclado....142

Figura 54: Evento de ratón 142

Figura 55: Diagrama de clases de diseño para el *Agente de Actuación*....172

Figura 56: Esquema de comportamientos del Agente de Actuación 184

Figura 57: Protocolo *Informar* 185

Figura 58: Protocolo *Preguntar* 185

Figura 59: Protocolo *Satisfacer Objetivo* 186

Figura 60: Protocolo *Aplicar Acción* 186

Figura 61: Protocolo *Ejecutar Acción* 187

1. Introducción

1.1. Planteamiento inicial

La rápida evolución de las técnicas informáticas junto con las de las comunicaciones se hace patente, cada vez con más fuerza, en todos y cada uno de los ámbitos de la sociedad moderna. Este hecho supone, no sólo un apoyo para el desarrollo de muchas actividades personales, sino, además, una herramienta tremendamente útil para el desarrollo profesional, y de manera especial en el campo del entrenamiento y/o la enseñanza.

Tradicionalmente, los métodos empleados en la enseñanza de actividades profesionales se han basado en una fase de transmisión de conocimiento teórico seguida de un período de asentamiento a través de la realización de trabajos prácticos con los que se pretende emular diversas situaciones que pueden surgir durante el desempeño de las mencionadas actividades.

El problema es que no siempre las circunstancias permiten realizar de manera adecuada esta labor de asentamiento, ya sea por la imposibilidad de recrear las características reales del entorno o por resultar una tarea demasiado arriesgada físicamente, económicamente, o que requiere de una serie de recursos que por distintos motivos no están disponibles. A esto hay que añadir que, a menudo, los trabajos precisan de la coordinación de equipos de personas que colaboren en el desarrollo de los mismos, por lo que se hace necesario un entrenamiento conjunto que muchas veces resulta inviable, por ejemplo, por las distintas localizaciones en las que pueden estar los miembros de dichos equipos.

Es en este contexto en el que las técnicas de simulación por ordenador adquieren mayor relevancia, debido a su capacidad para imitar situaciones o actividades que, por su naturaleza, pueden resultar peligrosas o simplemente costosas. Es por tanto deseable poder reproducir, computacionalmente hablando, ciertos escenarios con el objeto de enseñar o entrenar a las personas sobre la mejor manera de afrontarlos, evitando así los

riesgos físicos que pudieran conllevar estas actividades o los costes de cometer errores que de esta forma se podrían haber minimizado.

Además, para lograr el mayor grado de eficacia en esta tarea de entrenamiento, sería muy interesante poder copiar de la manera más fiel posible el marco en el que se desarrolla la actividad que es objeto de aprendizaje, ofreciendo la posibilidad de llevarla a cabo individualmente o en colaboración con más personas; así como también poder automatizar el proceso de supervisión del aprendizaje por parte de terceras entidades como pueden ser tutores, dando de este modo la opción de adaptar o particularizar la enseñanza a las necesidades de los distintos alumnos.

Una vez que se ha puesto de manifiesto la importancia de conseguir representaciones fieles de la realidad, otra faceta crucial en este propósito es la posibilidad de interacción con estas representaciones. Resulta obvio que el aprendizaje es de mejor calidad si se pueden manipular directamente aquellas herramientas o materiales que intervienen en la tarea en particular. De esta manera la enseñanza alcanza un grado muy alto de interactividad con los alumnos pasando de ser un proceso casi completamente pasivo a totalmente activo o participativo, con los beneficios que ello supone.

Por otro lado, y como se anticipa al comienzo de esta introducción, resulta muy difícil no aprovechar las ventajas que nos ofrece la era de las telecomunicaciones en la que vivimos. Consideremos como ejemplo un contexto en el que un equipo de personas situadas en distintos países, pero dedicadas al mismo oficio, precisan coordinarse para llevar a cabo una determinada labor con toda la eficacia de la que sean capaces. Para ello, lo más adecuado sería que recibiesen una apropiada instrucción previa a la realización de la tarea con el fin de capacitarles no sólo individualmente sino también como equipo. Haciendo uso de las infraestructuras de comunicaciones y en especial de Internet se podría evitar que los miembros del equipo tuvieran que reunirse físicamente, sin perjuicio para el éxito del entrenamiento.

Parece que todas estas necesidades que han sido expuestas apuntan en una misma dirección y es el empleo de técnicas de realidad virtual (RV). La realidad virtual se puede entender como un conjunto de tecnologías de simulación y visualización por computadora que permite recrear escenarios reales tridimensionales e interactuar con ellos sin necesidad de estar presente físicamente. Esta definición se ajusta como un guante al problema planteado si, además, se añade el uso didáctico que naturalmente se infiere de ella.

El desarrollo de un sistema que dé cobertura a las exigencias planteadas puede tener multitud de aplicaciones en ámbitos muy diversos, por lo que es importante que las soluciones propuestas sean flexibles, escalables y que, en la medida de lo posible, definan una arquitectura que sirva de modelo de aplicación para aquellos problemas en los que se pretenda hacer un uso educacional de las técnicas de realidad virtual.

Entre las distintas áreas que podrían beneficiarse de sistemas de este tipo están la medicina y en concreto la cirugía. Este es un ejemplo muy claro de una actividad profesional cuyo éxito depende en gran medida del entrenamiento y de los conocimientos prácticos previamente adquiridos, pero al mismo tiempo, este entrenamiento resulta imposible llevarlo a cabo en el mundo real por razones obvias. La realidad virtual puede entonces ser la solución a esta contingencia.

De la misma forma, son apropiados todos aquellos trabajos en los que se precisa la manipulación de materiales o herramientas peligrosas, como pueden ser los relacionados con la industria química, nuclear, eléctrica, etc. O también aquellos en los que se definen protocolos detallados de actuación que deben seguirse exhaustivamente, como es el caso de la industria aeroespacial, por ejemplo.

Como síntesis del problema expuesto, se puede decir que lo que se busca es un modelo para el desarrollo de sistemas de entrenamiento multiusuario basados en el diseño tridimensional por ordenador y la realidad virtual, con un alto grado de interactividad por parte del usuario y que sea flexible, reutilizable, adaptable y escalable de forma que se pueda cubrir la mayor gama de ámbitos posible.

1.2. Objetivos

Dada la amplitud del problema planteado y la heterogeneidad del mismo, es necesario dividirlo en subproblemas que puedan ser abordados con mayor facilidad. Es por esto que este trabajo únicamente trata de dar solución a uno de los subconjuntos de un sistema global denominado MAEVIF que ha sido desarrollado en el "Laboratorio Decoroso Crespo" de la Facultad de Informática de la Universidad Politécnica de Madrid. Como se verá más adelante, MAEVIF es un sistema que plantea una solución completa para nuestro problema.

El alcance de este trabajo abarca, tan sólo, lo referente a la interacción de los usuarios con el mundo virtual definido en MAEVIF y los efectos que estas interacciones provocan en el mismo.

Entendemos por interacción desde la simple exploración del entorno hasta la manipulación de los objetos que en él se encuentran. Para ello el sistema ha de ser capaz de controlar o manejar ciertos dispositivos que hacen las veces de interfaz con el usuario. A lo largo de este trabajo se presentan aquellos dispositivos que han sido utilizados, sus particularidades y la funcionalidad que ofrecen.

Además, se expone el diseño de un módulo software destinado al manejo de estos dispositivos, planteado siempre desde el punto de vista de su posible ampliación y adaptación a nuevos dispositivos de interacción.

Pero el concepto de interacción no es en absoluto unidireccional, es decir, no se limita únicamente a la observación de lo que sucede en el entorno virtual o a la manipulación de los elementos que lo integran, sino que también es posible desencadenar sucesos en él a través de esta interacción. Para lograr esto, hay que dotar al sistema de capacidad para monitorizar todo aquello que el usuario hace dentro de él y ser capaz de modificar el entorno de acuerdo a ello. Esto, como se explicará más adelante, implica cierta inteligencia para poder tomar iniciativas, establecer planes o efectuar simulaciones en función de las acciones del usuario.

Resumiendo, con este trabajo se pretende establecer un modelo, lo más genérico que sea posible, de comunicación entre los usuarios y el escenario virtual del que sean partícipes, proporcionando, además, la capacidad de transmitir a otras partes del sistema las acciones que el usuario provoca, para que éstas sean procesadas y se decida sobre qué variaciones es necesario aplicar en el entorno tridimensional.

A pesar que, como se ha comentado antes, en este texto se expone tan sólo un subconjunto de un sistema mucho más amplio, a lo largo de él resultará inevitable mencionar otros componentes con los que se relaciona, unas veces más fuertemente y otras de una manera superficial. Por eso, cuando la claridad de la explicación lo requiera se abordarán detalles ajenos al alcance del trabajo intentando siempre no desviar la atención sobre los temas que aquí nos ocupan.

1.3. Estructura del trabajo

Este trabajo está organizado principalmente en cuatro secciones con el siguiente contenido:

La primera sección (capítulos 1 y 2), en la que están incluidas estas líneas, es una sección meramente introductoria donde se exponen el problema que se está tratando de resolver, el contexto en el que se aborda, las tecnologías empleadas para su resolución, la actualidad teórica y práctica de los aspectos relacionados con él, y por último una serie de conceptos y nociones previas que sirven para la correcta comprensión de este trabajo. Se describen, entre otros, los conceptos de realidad virtual, dispositivo de realidad virtual, inmersión, interacción, entorno virtual, agentes inteligentes, etc.

La segunda sección (capítulo 3) es esencialmente descriptiva. Comprende la descripción general del proyecto que engloba a este trabajo, sus antecedentes y sus aplicaciones. Además se estudiará la estructura y/o arquitectura del mismo con el objeto de discernir dentro de ella qué parte es la que se trata en este escrito. En esta sección se describen, sin entrar en detalles, los módulos que componen dicha parte, su relación con el resto del sistema, así como la funcionalidad que aporta al conjunto global.

A partir de la tercera sección (capítulo 4) se empieza a exponer el contenido que ocupa exclusivamente a este trabajo. Aquí se presentan los distintos dispositivos de interacción con entornos virtuales que han sido estudiados e integrados dentro del proyecto. Para cada uno de ellos se describen sus características técnicas en cuanto a hardware y el desarrollo de módulos software que hacen posible su utilización, detallando los resultados de cada una de las fases que forman parte de un proyecto software según una metodología orientada a objetos [Larman, 1999]. Estas fases corresponden a la especificación de requisitos, diseño de alto nivel, diseño de bajo nivel, implementación y pruebas.

Por último, la cuarta sección (capítulo 5) se emplea en el desarrollo de aquellos componentes que interrelacionan el mundo virtual definido en el proyecto con el resto del sistema y que se encargan de hacer llegar a éste las acciones que inicia el usuario para obrar en consecuencia. Ya que esta parte se asienta sobre un estilo de desarrollo software orientado a agentes, se seguirá, por tanto, una metodología apropiada que se describirá a su debido tiempo.

Para finalizar, el trabajo consta de una última sección a modo de epílogo que trata de extraer una serie de conclusiones acerca de todo lo expuesto y también de establecer algunas líneas de trabajo futuro que puedan servir como guía para posteriores investigaciones que den lugar a ampliaciones o revisiones.

2. Estado de la cuestión

2.1. Antecedentes y definiciones previas

Este trabajo versa, fundamentalmente, sobre la manera de construir y aplicar sistemas basados en entornos virtuales inteligentes al entrenamiento. Es por ello que para la correcta comprensión del texto es necesario definir con precisión los conceptos que aquí han de barajarse, así como las tecnologías y metodologías empleadas en el área.

Se puede entender, sin entrar en profundas disertaciones, que "entorno virtual es aquel que ha sido generado mediante el uso de técnicas de realidad virtual" [Aston, 2003]. Sin embargo esta definición nos lleva a otra más amplia que es la de "realidad virtual".

"Realidad virtual" (RV) es un concepto que se aplica cuando "una persona está inmersa en un entorno generado por ordenador que comparte gran cantidad de similitudes con la realidad" [Kepell, 1997] [Aston, 2003].

También se pueden entender los sistemas basados en RV, desde el punto de vista de la interacción hombre-máquina, como aquellos en los que "el usuario interacciona con el sistema mediante la manipulación de objetos como si se tratase del mundo real" [Aston, 2003]. O desde el punto de vista de la tecnología como aquellos en los que se emplean dispositivos tales como sensores de movimiento, cascos o gafas de visión tridimensional, guantes para la manipulación de objetos virtuales, etc.

Cualquiera de estas definiciones es compatible con una característica que se extrae desde una perspectiva psicológica y es que "un sistema de RV es percibido de la misma manera que un entorno real".

Aunque el término de realidad virtual fue acuñado por primera vez en 1989 por el filósofo Jaron Lanier [http://www.well.com/user/jaron/], la evolución de dichos sistemas empezó en la segunda mitad del siglo veinte con investigaciones acerca de la simulación gráfica y la manipulación de la información contenida en un computador.

Actualmente se podrían distinguir tres tipos de sistemas basados en realidad virtual: sistemas de inmersión sensorial, no inmersivos y de inmersión cerebral directa.

Los sistemas de inmersión sensorial se apoyan en dispositivos que van desde cascos de visión y guantes dotados de sensores, hasta trajes completos cuya característica es la de privar al usuario de la percepción real de sus sentidos y reemplazarla por la información generada mediante un ordenador.

En los sistemas no inmersivos no se requiere de aparatos especiales pues no se pretende inhibir las sensaciones reales percibidas, sino más bien complementarlas con las generadas artificialmente. Hablamos aquí de los simuladores, por ejemplo.

Por último, los sistemas de inmersión cerebral directa establecen una vía de comunicación directa entre el cerebro y el computador haciendo que las sensaciones percibidas a través de los sentidos no se distingan de las creadas por el ordenador. Este tipo de sistemas no existe actualmente pero ya han sido contemplados en la literatura y en el cine y quién sabe si algún día, en el futuro, la ciencia permitirá que sean un hecho.

2.2. Motivaciones para el desarrollo de Entornos Virtuales para el Entrenamiento[1]

A continuación se examinan algunas de las razones que llevan al uso de entornos virtuales en lugar de los métodos tradicionales de entrenamiento.

En lo económico, como ya fue apuntado anteriormente, el uso de técnicas de simulación en el aprendizaje puede ofrecer resultados muy satisfactorios en relación al gasto durante su utilización. Sin embargo, pueden resultar un problema los costes previos derivados del desarrollo de este tipo de sistemas y el hardware necesario para su uso, aunque también hay que decir que en gran parte de las ocasiones sólo se precisará de un hardware básico.

[1] Más conocidos por el acrónimo inglés VET: "Virtual Environment for Training"

Otra ventaja de los entornos virtuales es que proporcionan un alto grado de eficiencia en la adquisición de conocimientos por parte de los estudiantes, dado que amplían enormemente el rango de escenarios y situaciones de la vida real que se pueden simular. Actualmente, las investigaciones se encaminan al uso de agentes educativos, como partes activas del sistema, que monitorizan el proceso de aprendizaje. Un ejemplo, ya clásico, es el proyecto desarrollado por el Departamento de Informática de la Universidad del Sur de California denominado STEVE (*Soar Training Expert for Virtual Environments*) [Rickel & Johnson, 1998] [Rickel & Johnson, 1999] [Rickel & Johnson, 2000].

STEVE es un agente diseñado para ayudar en el entrenamiento de las personas en tareas tales como manejar y mantener maquinarias complejas. Además, STEVE es un agente autónomo y animado que comparte espacio con el resto de usuarios en el entorno virtual y que continuamente monitoriza lo que en él sucede, es decir, la ejecución de tareas por el resto de usuarios; puede explicar sus acciones, proporcionar ayuda cuando se necesite y hacer demostraciones. En definitiva, actúa como tutor en el aprendizaje de otras personas.

Anteriormente también se comentó la importancia de poder realizar entrenamientos a distancia. Los sistemas basados en entornos virtuales multiusuario solucionan este problema.

Por último, pero quizás uno de los factores más influyentes a la hora de usar estos sistemas, está la eliminación total del riesgo durante el aprendizaje de procesos peligrosos y complejos, y no sólo en el ámbito industrial, como hasta ahora se ha mencionado, sino también en otras áreas profesionales como la defensa, sanidad, control del tráfico aéreo, entrenamiento de pilotos, e incluso el estudio de las reacciones sociales ante determinadas situaciones, estudio de fobias, etc.

Estas son algunas de las ventajas que se derivan del uso de entornos virtuales, pero esto no quiere decir que no existan inconvenientes. Uno de los problemas que encontramos cuando estamos inmersos en un entorno virtual es que, a menudo, resulta difícil la manipulación de objetos virtuales, sobre todo si para ello se necesita un cierto grado de precisión. El empleo de tecnologías que proporcionen retroalimentación táctil y de fuerza contribuye a paliar esta desventaja, pero el elevado coste y la "aparatosidad" de las mismas hacen que su utilización no sea frecuente. De la misma forma, el equipo necesario para la obtención de datos, como posición y orientación entre otros, puede ser demasiado caro, lo que repercute en la limitación en el número de dispositivos de interacción que son manejados y por consiguiente en la usabilidad de los sistemas diseñados.

Si seguimos hablando de precisión, nos topamos con el hecho de que gran parte de los dispositivos de entrada para RV, tienen importantes carencias en este aspecto, y éstas se ven agravadas por el limitado alcance con el que cuentan.

La solución a estos obstáculos sólo la podemos buscar en la investigación y el desarrollo de la tecnología y la electrónica, pero existen otra clase de dificultades que únicamente se podrán paliar con la generalización en la implantación de sistemas basados en realidad virtual, lo que, a buen seguro, proporcionará el paso del tiempo. Estas contrariedades se derivan del hecho de que hasta la fecha no se ha estandarizado ningún conjunto de interfaces de acceso para entornos virtuales. Estamos acostumbrados a interactuar con las aplicaciones a través de iconos, ventanas, menús, ratones, etc. y al ver estos elementos estamos plenamente familiarizados con ellos; sin embargo no ocurre lo mismo con los mecanismos de acceso a un mundo virtual (sensores, gafas, guantes, etc.) por lo que es más difícil adquirir destreza en su manejo. Para conseguir esta agilidad requerida se tiende a "humanizar" la representación virtual de estos elementos de interacción; por ejemplo, si se dispone de un mecanismo para manipular objetos virtuales sería conveniente darle la representación gráfica de una mano, de forma que su uso dentro del entorno virtual sea muy intuitivo y natural, pues todos sabemos qué movimientos debemos hacer con la mano para tocar, coger, etc.

2.3. Tecnologías relacionadas con entornos virtuales

Son muchas las tecnologías relacionadas con la construcción de entornos virtuales para el entrenamiento (VET), tanto hardware como software. Entre ellas se encuentran las interfaces de interacción hombre-máquina, la visualización y el diseño de espacios tridimensionales, procesamiento de sonido 3D, interfaces táctiles y de retroalimentación de fuerza, navegación por el entorno, posicionamiento y orientación espacial, inteligencia artificial…

Podemos empezar mostrando algunas de las múltiples herramientas que existen para el diseño de entornos virtuales tridimensionales. Las aplicaciones que cuentan con elementos gráficos en tres dimensiones hacen uso de componentes software que, básicamente, "ocultan" la gran cantidad de cálculo necesario para representar figuras en 3D y ofrecen al desarrollador una API con la que trabajar. Además, estos componentes optimizan el acceso al hardware gráfico disponible para lograr un rendimiento en tiempo real satisfactorio.

Como ejemplo de librerías para el desarrollo de gráficos tenemos OpenGL. Ésta es de libre distribución, presenta una API de fácil manejo, posee un alto grado de compatibilidad con otros elementos software que pueda incluir el sistema, ofrece una interfaz de programación de bajo nivel en los aspectos relacionados con el cálculo algebraico de sistemas de referencia, movimientos, transformaciones, giros, etc. de figuras geométricas (lo que puede considerarse una ventaja o una desventaja según las necesidades) y está disponible para su uso bajo distintos sistemas operativos. OpenGL sólo dispone de funcionalidad para manejar gráficos, por lo que la interacción con éstos habrá de hacerse por otros medios.

Como contrapunto a OpenGL, tenemos el conjunto de librerías DirectX, que en este caso son desarrolladas y distribuidas por Microsoft. DirectX proporciona, no sólo el desarrollo de gráficos en tres dimensiones (Direct3D), sino también acceso a dispositivos periféricos (DirectInput), desarrollo de gráficos en dos dimensiones (DirectDraw), sonido (DirectSound), y desarrollo de aplicaciones multiusuario (DirectPlay). Quizá el hecho de ser un conjunto bastante completo y autocontenido de componentes software, unido a que son propiedad de Microsoft, hacen que con

frecuencia sea difícil compatibilizar su uso en aplicaciones con un modelo de programación distinto al impulsado por la propia compañía.

Estas bibliotecas y otras permiten el dibujado de figuras geométricas y su manipulación como un conjunto de vértices, aristas, superficies, polígonos, texturas, luces y colores; pero en un entorno virtual existen normalmente gran multitud de objetos, lo que hace del todo imposible crear cada uno vértice a vértice y arista a arista. Es por eso que existen herramientas de apoyo al diseño gráfico con las que se pueden diseñar escenarios tridimensionales completos y exportar todas las características de todos sus elementos en un formato intermedio, para posteriormente importarlos desde las aplicaciones y que puedan ser visualizados y manipulados.

Ejemplo de estas herramientas de modelado de gráficos es el entorno de trabajo 3DStudio MAX/MAX3, que ofrece las capacidades comentadas y muchas más, para elaborar entornos virtuales con gran calidad visual y un alto grado de realismo.

Desde un punto de vista de más alto nivel que el de la programación y el modelado de gráficos, existen herramientas de trabajo integradas y enfocadas directamente al desarrollo de entornos virtuales. Entre estas herramientas que podemos denominar "de autor" están:

- Superscape VRT: Desarrollada por Superscape VR plc. Es una aplicación compuesta por un conjunto de editores de formas, sonidos y texturas, y un visualizador usado para explorar los entornos creados.
- dVISE: Desarrollada por Division Ltd. Es una plataforma bastante flexible en cuanto a la importación de gráficos modelados con diversas herramientas. Está orientada a la creación de entornos virtuales sin necesidad de tener conocimientos de programación.
- Meme: Desarrollado por Immersive Systems, Inc. Su enfoque se dirige hacia el diseño de mundos virtuales que puedan ser integrados en sistemas distribuidos multiusuario.

- VRCreator: Desarrollado por VREAM Inc. Ofrece soporte para programadores y no programadores, permite importar gráficos desde diversos formatos así como exportarlos a VRML, y dota a las aplicaciones realizadas con VRCreator de facilidades para la interacción con los objetos 3D.
- COVISE/COVER: Desarrollado por High Performance Computing Center. Es una plataforma para la integración de simulaciones distribuidas con un enfoque colaborativo. COVER es una parte de COVISE para dar soporte a aplicaciones tanto técnicas como científicas.
- WorldUp Release 4: Desarrollado por Sense8, proporciona características de tiempo real en la construcción de aplicaciones de realidad virtual.
- MASSIVE-2: Desarrollado en la Universidad de Nottingham por el *Communications Research Group* (CRG), está especialmente diseñado para soportar el trabajo multiusuario así como la conexión entre entornos virtuales heterogéneos, aunque la interacción con los mismos no es su punto fuerte. Proporciona una interfaz de programación de aplicaciones bien definida.

Algunas de estas herramientas ofrecen soporte para la realización de aplicaciones a partir de ellas a través de la definición de una API de programación. Sin embargo, existen sistemas y lenguajes especialmente enfocados en este sentido, como son:

- Dive, desarrollado por SICS: Las siglas DIVE significan *Distributed Interactive Virtual Environment* (Entorno virtual interactivo distribuido). Es un sistema no comercial con código abierto por lo que goza de gran popularidad en la investigación a pesar de trabajar únicamente bajo sistemas UNIX. Entre las posibilidades que ofrece están la de dar soporte a distintos periféricos de entrada/salida como guantes o HMDs, que se verán más adelante.
- dVS, desarrollado por Division Ltd.: Este sistema funciona de forma similar a un sistema operativo de realidad virtual proporcionando servicios a las aplicaciones tales como abstracciones para visualización, sonido, posicionamiento, soporte multiusuario, etc.

- World Toolkit, desarrollado por Sense8: Es una API comercial formada por un conjunto de rutinas de alto nivel escritas en el lenguaje de programación C para el desarrollo de aplicaciones de realidad virtual. Ofrece soporte para gran variedad de hardware específico de realidad virtual y está disponible para diversas plataformas, aunque, preferentemente, está diseñado para máquinas basadas en sistemas UNIX que hacen uso de librerías gráficas como OpenGL o SGI Performer.
- VRML significa *Virtual Reality Modelling Language* (Lenguaje de Modelado de Realidad Virtual) y se trata de un lenguaje estándar para el manejo y la interacción con escenas tridimensionales en el ámbito de Internet. VRML permite modelar formas, colores, movimiento y comportamiento. Además es compatible con la programación de comportamientos más sofisticados en Java o JavaScript.
- Por último, siempre se puede prescindir de todas estas herramientas y desarrollar entornos virtuales directamente mediante el uso de lenguajes de alto nivel como C, C++, Java, etc.

2.4. Interfaces de acceso para realidad virtual

Una vez que sabemos cómo podemos diseñar entornos virtuales cualesquiera es deseable que éstos tengan un comportamiento dinámico, que no sean elementos pasivos y estáticos que únicamente se puedan mostrar. Anteriormente hablábamos de ciertos dispositivos que establecían una interfaz de acceso y de interacción entre el hombre y la máquina. Todos conocemos y sabemos manejar el teclado y el ratón para comunicar a las aplicaciones órdenes mediante mandatos escritos o haciendo "clic" sobre los iconos. De la misma forma, es de sobra conocida y ampliamente extendida la manera en la que visualizamos la información procedente del computador a través de monitores o pantallas. Pues bien, existen otros dispositivos cuyas características son más adecuadas para la inmersión o la manipulación, y por tanto el mejor aprovechamiento de la información, dentro de entornos virtuales.

Empezaremos comentando algunas de las interfaces visuales que existen para este fin, con sus ventajas e inconvenientes. Hay dos factores fundamentales en los que nos fijaremos especialmente a la hora de usar este tipo de interfaces, y son: la resolución de la imagen frente a la velocidad de formación de las figuras, y la visión monoscópica frente a la visión estereoscópica.

En primer lugar disponemos de las lentes de visualización sobre cristal líquido o, comúnmente, gafas de 3D. Éstas tienen una apariencia similar a unas gafas corrientes pero cuentan con un fotosensor que lee una señal procedente del computador indicándole si debe dejar pasar la luz en el lado izquierdo o en el derecho. Cuando la luz pasa a través del lado izquierdo, en la pantalla del ordenador sólo se mostrará la parte de la escena que corresponde a lo que el usuario vería por el ojo izquierdo, y cuando se activa el lado derecho se muestra la misma escena ligeramente desplazada a la derecha. La conmutación entre izquierda y derecha se produce a una frecuencia de 60 Hertz, lo cual causa que el usuario perciba una escena tridimensional continua.

Figura 1: Lentes LCD

Este tipo de gafas tiene la ventaja de ser ligeras y sin cables, pero tienen el inconveniente de que la imagen sólo se muestra proyectada en una pantalla, por lo que el usuario deberá mirar fijamente a ésta pudiendo, de lo contrario, ver el medio ambiente real. Esto no proporciona un efecto de inmersión. En cuanto a la resolución y velocidad de formación de la imagen, será la que permita el hardware instalado en el computador.

Para lograr un mayor efecto de inmersión, existen también cascos de visión que colocan frente a cada ojo una pequeña pantalla y que aíslan al usuario del ambiente exterior. Estos cascos son complementados, normalmente, con sensores que marcan la posición y orientación de la cabeza, por lo que, dependiendo de ésta, variará la imagen presentada en los visores. Además, se suele hacer uso de espejos y lentes ópticas para agrandar la vista, llenar el campo visual y dirigir la escena a los ojos.

Entre los distintos tipos de cascos o HMD (*Head Mounted Display*) más extendidos, podemos hacer una pequeña clasificación atendiendo a los componentes electrónicos que usan:

Figura 2: HMD

Para empezar, tenemos los HMD con tecnología LCD. Las características de esta tecnología hacen que la imagen sea más clara que en la mayoría de los HMD pero, como contrapartida, la resolución y el contraste son bajos. El problema que conlleva la baja resolución es la incapacidad para identificar objetos y localizar su posición exacta; además, existe un pequeño retardo en la polarización del cristal de las lentes que puede causar que el usuario juzgue incorrectamente la posición de los objetos.

Mejores prestaciones ofrece el HMD proyectado, en el que el fósforo de la pantalla es iluminado por la luz transmitida a través de cables de fibra óptica. Proporciona mejor resolución y contraste que el LCD, lo que significa imágenes con mayor detalle, pero resulta muy caro y difícil de fabricar.

El HMD con tecnología CRT funciona de manera muy similar al anterior, pero en este caso cada visor cuenta con un pequeño tubo de rayos catódicos que ilumina la pantalla mediante haces de electrones, como en una televisión convencional. Este tipo de HMD es más pesado y genera gran cantidad de calor, por lo que el usuario puede sentirse incómodo.

En general los HMD tienen el problema de tener un cable conectado a la computadora que hace que su uso sea molesto.

Otro tipo de interfaz visual, que puede sustituir a las lentes LCD o a los cascos, es el llamado BOOM (*Binocular Omni-Orientation Monitor*). Este dispositivo está soportado por un brazo mecánico articulado con sensores de posicionamiento localizados en las articulaciones. Para ver el entorno virtual, el usuario sostiene el monitor y se coloca frente a él. La escena a mostrar se generará de acuerdo a la posición y orientación de las articulaciones del brazo. El BOOM, como muestra la figura, tiene el inconveniente obvio de la limitación de movimientos del usuario, aunque puede resultar más cómodo que el HMD al no tener que ponerse ningún aparato en la cabeza.

Figura 3: BOOM

Por último, en cuanto a visualización de escenarios virtuales, existe un dispositivo novedoso denominado CAVE (*Cave Automatic Virtual Environment*) [http://www.evl.uic.edu/EVL/] que ha sido desarrollado en el *Electronic Visualization Laboratory* en la Universidad de Illinois en Chicago. CAVE es un sistema de realidad virtual basado en la proyección de la imagen sobre cuatro pantallas dispuestas alrededor del observador en forma de habitación. Mediante un sistema de proyectores y espejos, las imágenes son mostradas en las paredes y en el suelo. El observador percibe la sensación tridimensional con la ayuda de unas lentes 3D como las expuestas anteriormente. Además, mediante el uso de sensores de orientación colocados en la

cabeza, se logra que la imagen reflejada sea coherente con los movimientos del usuario. También se pueden añadir dispositivos para interaccionar con el mundo virtual proyectado, tales como guantes o ratones 3D.

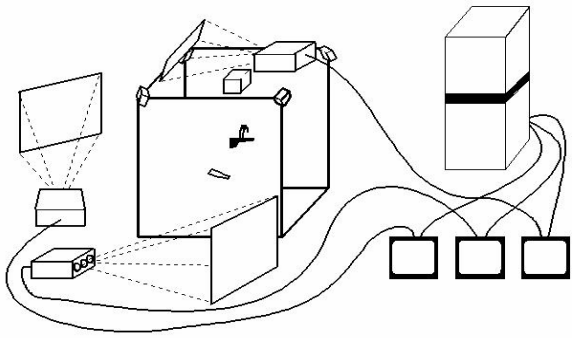


Figura 4: CAVE

La filosofía de los diseñadores de CAVE consiste, entre otras cosas, en:

- Conseguir una alta resolución en las imágenes proyectadas así como mejorar la visión periférica.
- Reducir el número de dispositivos hardware que ha de llevar el usuario.
- Permitir al observador ver simultáneamente la realidad y el entorno virtual sin transiciones que puedan provocar desconcierto.

Junto con la visión, uno de los factores que más contribuyen a conseguir realismo dentro de un entorno virtual, es el sonido. Está demostrado que proporcionar información sonora a los usuarios de un computador, en general, aumenta la capacidad de asimilación, y especialmente si lo que se pretende es recrear un ambiente real con realidad virtual.

Dentro de un entorno virtual en tres dimensiones, el sonido ha de ser percibido también en tres dimensiones, y para ello ha de tenerse en cuenta el comportamiento de las ondas sonoras ante obstáculos, fuentes múltiples de sonidos, ruido de fondo, así como la ausencia de él. El usuario, al igual que en la realidad, ha de poder ubicar la procedencia de los sonidos que escucha, para lo que se requiere una gran potencia de cálculo que, imitando el funcionamiento del oído humano, hace uso de retardos de microsegundos en la recepción para determinar la posición de la fuente del sonido.

El problema más importante a la hora de recrear un ambiente sonoro tridimensional es el de no poder reproducir sonidos previamente grabados, ya que la percepción de los mismos varía en función de la posición del oyente o las condiciones del ambiente, por lo que se necesita calcular estos sonidos en tiempo real. Además, se sabe que cada persona percibe los sonidos de manera distinta, por tanto la computadora deberá personalizar el cálculo en función del usuario y en base a unos parámetros de referencia extraídos sobre la marcha de mediciones tomadas mediante micrófonos desde el propio oído.

Un ambiente sonoro realista puede ser una gran ayuda para invidentes en el aprendizaje de sus tareas cotidianas.

Parece claro que la búsqueda del realismo dentro de simulaciones virtuales pasa por conseguir imitar, lo mejor posible, la capacidad de percepción a través de los sentidos del ser humano. Pues bien, es el turno del tacto; para obtener más credibilidad dentro de un entorno virtual, todas las cosas han de ser igual de "tangibles" que en el mundo real, es decir, los objetos sólidos, puesto que tienen volumen y ocupan un espacio, no se pueden atravesar, y puesto que tienen masa, deben pesar. En estas dos afirmaciones se resume el propósito de dos importantes áreas de investigación en realidad virtual, que son la retroalimentación táctil y la retroalimentación de fuerza o quinestética.

La retroalimentación táctil trata de la manera en que los objetos son percibidos mediante el sentido del tacto, es decir, su temperatura, tamaño, forma, firmeza y textura. Los dispositivos que ofrecen tales características interactúan directamente con los nervios terminales en la piel, estimulándolos de forma que parezca que se aprecian estas sensaciones. Sin duda, la textura de una superficie es la característica más difícil de simular. El sistema *Sandpaper* (papel de lija), desarrollado por el MIT (*Massachusetts Institute of Technology*) y la UNC (*University of North Carolina*) es la mejor aproximación documentada que se conoce para lograrlo, pues es capaz de simular con exactitud varios tipos diferentes de papel de lijar.

Por el contrario, la quinestética se ocupa de la manera en la que el entorno influye físicamente en el usuario, evitando, por ejemplo, que pueda atravesar paredes, o haciendo posible que sea capaz de sentir cuándo se golpea o se tropieza con un objeto. Los dispositivos que dan soporte a esta área interactúan con los músculos y tendones del usuario para hacer sentir la fuerza sobre ellos. Entre ellos podemos contar los siguientes:

- Plataformas de movimiento; originalmente diseñadas para los simuladores de vuelo. Se trata de una plataforma que sostiene al usuario y que, mediante un sistema hidráulico, se mueve de acuerdo a las imágenes mostradas en la interfaz visual, aunque como es lógico, tiene limitaciones en el rango de movimiento.

Figura 5: Plataforma de movimiento

- Guantes: estos dispositivos, de los que se hablará con detalle más adelante, disponen de un número de sensores repartidos por todas las partes de la mano que permiten seguir la posición y flexión de cada dedo y cada articulación de la misma, y adicionalmente cuentan con complementos mecánicos que ejercen presión sobre distintos puntos de la mano para simular la resistencia que ofrecen los objetos al tocarlos o cogerlos.

Figura 6: Guante

- Dermoesqueletos: Son brazos robóticos que se acoplan a las extremidades de la persona y que, al igual que los guantes, simulan la resistencia o el peso de los objetos en un mundo virtual mediante el control de la fuerza trasmitida a la persona. Un ejemplo de estos brazos es el desarrollado en la Universidad de Utah [http://www.es.utah.edu/-jmh/sda-master-tom.gif].

- Mayordomos: Son robots que, básicamente, interponen obstáculos reales ante el usuario inmerso en el mundo virtual para que lo que éste esté percibiendo sea coherente con la realidad física. Este sistema se está investigando en la universidad de Tokyo [Tachi, 1995].

Ya hemos hablado de tres sentidos fundamentales del ser humano, vista, oído y tacto, que le permiten percibir el ambiente exterior tanto en la realidad física como en la virtual. Recientes investigaciones están interesadas también en un cuarto sentido, el olfato, como otro mecanismo más de percepción sobre el entorno. En la actualidad sabemos que han sido desarrollados dispositivos que son capaces de mezclar distintas sustancias químicas de acuerdo a un patrón predefinido para así generar olores [Stetter & Penrose, 2001]. Las utilidades de esto están aún por explorar pero es evidente que, una de las que perseguimos con más entusiasmo, la credibilidad en entornos virtuales, aumenta considerablemente.

Además de los sentidos que usamos directamente para la percepción de todo aquello que nos rodea, existen otros que nos dan conciencia, de manera innata, de nuestra localización y orientación, es decir, gracias a ellos podemos ubicar nuestra posición en un mapa, por ejemplo, y discernir entre lo que es “arriba” y “abajo” o “delante” y “detrás”. Aunque nuestro cerebro, inconscientemente, realiza esta tarea por nosotros, cuando una persona está inmersa en un mundo virtual en donde la percepción del resto de sus sentidos ha sido reemplazada por información generada por ordenador, también la percepción de dónde está y si está mirando hacia arriba o hacia abajo, ha de ser suministrada artificialmente para que, fundamentalmente y entre otras cosas, esa persona sea capaz de mantener el equilibrio y desplazarse por el entorno.

Esto nos lleva a lo que podríamos denominar navegación virtual. Hay muchas formas de moverse por un entorno virtual, la más natural es caminando, pero para eso la computadora ha de conocer en todo momento cuál es nuestra posición y orientación para refrescar aquello que vemos y oímos. El usuario habrá de llevar incorporados, por tanto, lo que se conoce como *trackers* o sensores de posicionamiento.

El propósito de estos dispositivos es determinar las coordenadas tridimensionales (x, y, z) y los ángulos de orientación (elevación, azimut, rotación) del individuo con respecto a un punto fijo de referencia u origen de coordenadas. Algunos de los dispositivos que hemos comentado en párrafos anteriores, como los HMD, guantes, etc. son complementados con estos sensores. Aunque en capítulos posteriores se hablará de ellos, aquí se exponen algunos de los principios de su tecnología que dan lugar a una pequeña clasificación [Perry et al. 2001].

Los *trackers* mecánicos no son más que brazos articulados, similares al BOOM, pero que se pueden acoplar a otras partes de la anatomía, como por ejemplo las manos. Tienen la ventaja de su rapidez, exactitud en los datos y la eliminación de temblores, pero sin embargo son bastante restrictivos en cuanto al rango de movimiento.

Los más extendidos son los *trackers* electromagnéticos, que se basan en la detección de una serie de campos magnéticos, generados a propósito y dispuestos de una determinada manera, para calcular la posición y orientación del sujeto. Este tipo de *trackers* son bastante inexactos, tienen problemas de latencia y presentan distorsión en los datos por la presencia de otros campos magnéticos ajenos. Además el alcance de los mismos tan sólo abarca unos pocos metros.

Existen también *trackers* basados en ultrasonidos, que constan de un sistema de emisores y receptores que intercambian ondas sonoras de alta frecuencia. La posición y la orientación son calculadas midiendo las diferencias de fase de las ondas emitidas por transmisores y receptores o bien calculando el tiempo que las ondas tardan en llegar a los receptores. El empleo de ultrasonidos tiene la ventaja de ser inmune a distorsiones magnéticas pero, sin embargo, sí le afectan la presencia de obstáculos, las variaciones en la temperatura y presión y el nivel de humedad y, como en el caso de los anteriores, el alcance no es muy grande.

De manera muy similar a los ultrasonidos se utilizan las ondas infrarrojas (*trackers* ópticos) para la determinación de posición y orientación, pero, al igual que con los primeros, aquí también es un problema la presencia de obstáculos entre transmisores y receptores. Por el contrario se percibe una mejora en la latencia, aunque cualquier otra fuente de luz puede afectar a la corrección de las medidas.

Por último, existe un tipo de *trackers* que no precisa de cables ni de hardware específico de conexión con el computador, por lo que el radio de trabajo aumenta considerablemente. Se denominan *trackers* inerciales y están basados en el principio físico de conservación del momento angular.

Además de los dispositivos basados en sensores, hay otros cuya misión también es la de permitir navegar por el entorno virtual. Se trata de los ratones 3D (wand), similares a los convencionales pero adaptados al movimiento sobre un espacio tridimensional, y los joysticks o palancas de mando que serán tratados en capítulos posteriores.

2.5. Áreas de investigación sobre entornos virtuales

Llegados a este punto, ha quedado patente la amplitud de la materia que engloba la creación de entornos virtuales. Sin embargo, no es de extrañar, dado el interés por este tipo de investigaciones, que se siga profundizando en nuevas formas de interacción con sistemas de realidad virtual o en aumentar cada vez más el realismo de los mismos.

Actualmente, se está renovando el interés por incluir dentro de los entornos virtuales sistemas de reconocimiento de voz que permitan una comunicación más directa y natural entre el hombre y la computadora.

Además, como cabría esperar, existe una investigación continua en lo que se refiere a mejorar la calidad y el realismo en la percepción visual, sobre todo, y también en la audición tridimensional.

2.5.1. Inteligencia en VETs

Hasta ahora se ha explicado con cierto grado de detalle las características que conforman un entorno virtual así como las herramientas que le dan soporte. Pero no hemos de olvidar que este documento se centra específicamente en un tipo concreto de entornos virtuales especialmente diseñados para el entrenamiento (VET).

En los procesos de enseñanza o entrenamiento se precisa de una supervisión constante por parte de alguna entidad encargada de instruir al alumno en una determinada tarea, corregir sus errores, responder a sus preguntas, adecuar el ritmo de la enseñanza a sus necesidades y en definitiva conseguir que al final del proceso el alumno haya asimilado correctamente los nuevos conocimientos.

En la vida real este papel es llevado a cabo por una persona denominada tutor o profesor.

En la actualidad se investiga la construcción de modelos informáticos que imitan este proceso, los denominados "Sistemas inteligentes de tutoría" (ITS[2]), y su aplicación dentro de los VET.

Estos sistemas, como su nombre indica, han de aparentar comportamientos inteligentes, para lo cual es frecuente que empleen técnicas que entran dentro del área de la inteligencia artificial, como son los algoritmos para la planificación de tareas, cálculo de trayectorias, navegación automática, etc.

Dentro de este campo son de especial interés los sistemas basados en agentes inteligentes y sistemas multi-agente [Huhns, 1998].

El concepto de agente surge dentro del marco de la inteligencia artificial con la pretensión de mejorar unos sistemas demasiado rígidos, poco reutilizables y difíciles de actualizar. En general, los objetivos que se persiguen son:

- Ubicuidad en todas las partes que forman el sistema.

[2] Del ingles "Intelligent Tutoring System".

- Interconexión entre los componentes del sistema.
- Inteligencia en la resolución de problemas.
- Sistemas de computación móviles.
- Delegación de tareas.
- Usabilidad para facilitar la ejecución de tareas periódicas o repetitivas.

En la literatura se pueden encontrar multitud de definiciones que intentan concretar las características que ha de tener un agente, sin embargo, ninguna de ellas es comúnmente aceptada. Algunas de ellas son:

- "Los agentes autónomos son sistemas informáticos que habitan en entornos dinámicos complejos, perciben y actúan de forma autónoma en este entorno y llevan a cabo un conjunto de objetivos o tareas para los que fueron diseñados" [Maes, 1995].
- "Los agentes inteligentes continuamente realizan tres funciones: percepción de las condiciones del entorno; acción sobre las condiciones del entorno; y razonamiento para interpretar las percepciones, resolver problemas, construir inferencias y determinar acciones" [Hayes-Roth, 1995].
- "Una entidad permanente de software dedicada a una tarea específica" [Smith et al. 1994].
- "Programas informáticos que simulan relaciones humanas haciendo algo que otra persona podría hacer por ti" [Selker, 1994].
- "… un sistema informático hardware o (más frecuentemente) software que posee las siguientes propiedades:
  - Autonomía: los agentes operan sin la intervención directa de humanos u otros agentes, y tienen alguna clase de control sobre sus acciones y su estado interno.

- Sociabilidad: los agentes interactúan con otros agentes (y posiblemente humanos) a través de alguna clase de lenguaje de comunicación entre agentes.
- Reactividad: los agentes perciben su entorno (que puede ser el mundo físico, un usuario mediante una interfaz gráfica de usuario, un conjunto de otros agentes, Internet, o quizá una combinación de éstos), y responde de manera oportuna a los cambios que suceden en él.
- Proactividad: los agentes no sólo actúan en respuesta a su entorno, también son capaces de mostrar comportamientos dirigidos por objetivos tomando la iniciativa…" [Wooldridge and Jennings, 1995].

- "Un agente es cualquier cosa de la que pueda decirse que percibe su medio a través de sensores y que actúa sobre ese medio a través de actuadores" [Russel & Norving, 1995].

Atendiendo a todas estas definiciones podemos elaborar una conclusión propia sobre qué es un agente.

Un agente es un sistema concebido para realizar una tarea, más o menos especializada, que además está inmerso en un entorno concreto, es decir, tanto su comportamiento como su estado están ligados a ese entorno, y su devenir, necesariamente, ha de provocar en el agente las respuestas adecuadas que le aproximen a conseguir sus objetivos. Aunque a priori se tienda a pensar en un agente como en un sistema software esto no es siempre así, pues las cualidades que manifiestan los agentes pueden también observarse en toda clase de sistemas desde seres vivos hasta sistemas estrictamente hardware.

La situación de este tipo de sistemas en un entorno determinado nos lleva a considerar una de las características de los agentes, y es su capacidad para interactuar con ese entorno en el que "habitan", de manera continua en el tiempo y a través de la percepción y posterior reacción sobre el mismo.

Pero entonces, ¿cualquier autómata o máquina dotada de mecanismos para percibir el entorno que le rodea y reaccionar en consecuencia, se puede considerar un agente? es más, ¿todo programa software que recibe unas entradas y genera unas salidas podría ser un agente? [Franklin et al. 1996].

Otras características que podemos destacar, y que pueden ayudar a determinar qué sistemas se comportan como agentes y cuáles no, son la autonomía, en cuanto a la capacidad para almacenar su propio estado y exclusividad para manipularlo, y en cuanto a su comportamiento, totalmente independiente de otras entidades externas, para lograr su cometido.

Además un agente debería tener capacidad social, es decir, la capacidad de comunicarse con otros agentes para cooperar por un fin común o competir por recursos necesarios para llevar su tarea a cabo.

Por último, existe una cualidad importantísima de los agentes que se denomina proactividad y consiste en ser capaz de tomar iniciativas propias. Un agente conoce sus metas y es por ello que puede establecer planes o estrategias diversas para conseguirlas de la mejor manera posible.

Aparte de las características expuestas, existen otras con un marcado carácter antropomórfico [Bates et al. 1992] [Bates, 1994] [Wooldridge and Jennings 1995], que podrían estar presentes o no en un agente; éstas son, por ejemplo, la capacidad de tener creencias [Shoham, 1993] sobre el estado más probable del entorno, de tener deseos sobre el estado final, de poseer intenciones, entendidas como corrientes de acción distintas, de aprendizaje, de adaptación, de desarrollar una personalidad concreta, etc.

Podemos decir que todos estos rasgos delimitan, en mayor o menor medida, la condición de ser agente. Pero, ¿qué factores aportan la racionalidad a un agente? ¿qué le confiere inteligencia?

Se dice que un agente es inteligente porque es capaz de cambiar su comportamiento en función de lo que percibe del entorno, y que es racional porque es capaz de hacer "lo correcto"; "lo correcto" es aquello que le permite al agente obtener el mejor resultado,

cualitativamente hablando, en el menor tiempo. La racionalidad de un agente inteligente depende de los siguientes factores [Russel & Norving, 1995]:

1. De la medida con la que se evalúe el éxito logrado.
2. De todo lo que hasta el momento ha percibido el agente.
3. Del conocimiento que posea el agente acerca del medio.
4. De las acciones que el agente pueda emprender.

Por último, y dejando a un lado las reflexiones teóricas, se puede decir que los sistemas multi-agente, en parte, constituyen una evolución de la ingeniería del software orientada a objetos para apuntar hacia arquitecturas distribuidas basadas en componentes autónomos, flexibles, adaptables e inteligentes. Por otro lado, también suponen un avance sobre el concepto de sistemas expertos como entidades con conocimiento especializado capaces de aplicarlo, de una manera cooperativa, en la resolución de problemas.

2.5.2. Metodologías de desarrollo de sistemas multi-agente

A la hora de abordar soluciones basadas en agentes hay que tener presente que éstos, a pesar de poseer ventajas enormemente potentes, no son en absoluto la respuesta para todo tipo de problemas, y su uso incorrecto puede repercutir en la construcción de sistemas ineficientes, debido, por ejemplo, a la poco conveniente explotación de la concurrencia inherente a ellos, o sistemas cuya solución más adecuada no pasa por el desarrollo de agentes, sino por arquitecturas más clásicas. Por esto es por lo que se precisa de metodologías y estándares que ayuden a desarrollar software orientado a agentes desde la perspectiva de la ingeniería del software.

Entre estos estándares, en su mayoría *de facto*, encontramos:

- FIPA (*Foundation for Intelligent Physical Agents*) [http://www.fipa.org]: Estándar para agentes desde 1996. Define las pautas para lenguajes de comunicación de agentes, interacción entre agentes y otro software, gestión de agentes, interacción hombre-agente, movilidad, ontologías, transporte de mensajes y nombrado de agentes. FIPA pretende sentar las especificaciones

necesarias que hacen posible la compatibilidad entre agentes heterogéneos. Todas las especificaciones definidas por FIPA están disponibles para los desarrolladores en Internet, así como implementaciones que cumplen el estándar.

- FIPA ACL (Agent Communication Language) [http://www.fipa.org/specs/fipa00061/]: Define un lenguaje para la comunicación entre agentes pero en este caso cumpliendo las especificaciones FIPA.
- KQML/KIF (Knowledge Query and Manipulation Language/Knowledge Interchange Format) [http://www.cs.umbc.edu/kqml/]: Como el anterior, define un lenguaje y un conjunto de protocolos para el intercambio de información y conocimiento entre agentes. KQML es parte del *ARPA Knowledge Sharing Effort* que tiene como objetivo desarrollar técnicas para construir bases de conocimiento reutilizables y con capacidad para ser compartidas a gran escala. KQML puede ser utilizado además como lenguaje de comunicación con/entre sistemas inteligentes.

Existen, además, una serie de entornos para el desarrollo y la integración de agentes en distintas aplicaciones, los cuáles soportan algunos de los estándares mencionados arriba:

- ZEUS (British Telecom) [http://more.btexact.com/projects/agents/zeus/]: Es un conjunto de componentes Java, con código abierto, que proporcionan librerías para el desarrollo de agentes y herramientas de visualización. Soporta el estándar FIPA desde su versión 1.2.
- JADE (Java Agent DEvelopment framework) [http://sharon.cselt.it/projects/jade/] [Bellifemine et al., 2002]: Es un entorno para el desarrollo de sistemas multi-agente de acuerdo a las especificaciones del estándar FIPA para agentes inteligentes. Incluye dos productos principales: una plataforma para la ejecución de agentes y un conjunto de librerías para su programación. JADE está enteramente desarrollado en Java

con la filosofía de software libre y código abierto y se distribuye bajo los términos de licencia *lgpl*. Algunas de sus características son:

- La plataforma puede ser distribuida en varias máquinas y proporciona soporte para la implementación de los agentes como *threads* agrupados dentro de contenedores.
- Una interfaz gráfica de usuario permite monitorizar agentes y contenedores desde una máquina remota.
- Herramientas para la depuración de agentes.
- Soporte para la movilidad de agentes entre máquinas distintas. Un agente puede congelar su ejecución, viajar a otro lugar físico y retomar su tarea en el punto donde se quedó.
- Soporte para la ejecución de varios comportamientos en concurrencia por cada agente.
- La plataforma incluye un AMS (*Agent Managment System*), un DF (*Directory Facilitator*) y un ACC (*Agent Communication Channel*).
- Soporte para el intercambio de mensajes ACL (Agent Comunication Language) entre agentes de acuerdo con FIPA y de manera transparente al desarrollador.
- Servicios de nombrado e identificación de agentes según FIPA.

En la siguiente figura se muestra un esquema que ilustra la relación entre los distintos componentes de la arquitectura JADE siguiendo el modelo propuesto por FIPA:

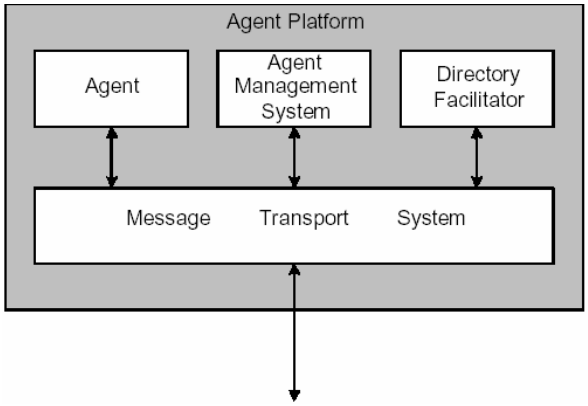


Figura 7: Arquitectura de la plataforma JADE de agentes FIPA.

En el centro del esquema podemos ver un módulo denominado sistema de gestión de agentes o AMS (*Agent Management System*); este módulo realiza la labor de control y supervisión sobre el uso de la plataforma, gestiona el ciclo de vida de los agentes que habitan en ella, mantiene un registro de identificación y estado para todos ellos y proporciona un servicio de "páginas blancas", es decir, un servicio que permite la localización de agentes por medio de su nombre o identificador.

En la plataforma existe, además, otro servicio de localización de agentes llamado DF (*Directory Facilitator*) o servicio de "páginas amarillas". En este caso la localización se realiza a través de la descripción de la especialidad o tarea para la que cada agente ha sido diseñado.

Por último, estos elementos se comunican entre sí, y también con el conjunto de agentes que residen en la plataforma, a través de un canal de comunicación llamado ACC (*Agent Comunication Channel*) que controla el intercambio de mensajes, incluyendo los procedentes de plataformas remotas.

Como se mencionó antes, la plataforma JADE puede distribuirse entre varias máquinas agrupando en contenedores los agentes residentes en cada una. Sólo un contenedor es el principal y en él residen el AMS y el DF. La siguiente figura muestra un ejemplo de distribución lógica:

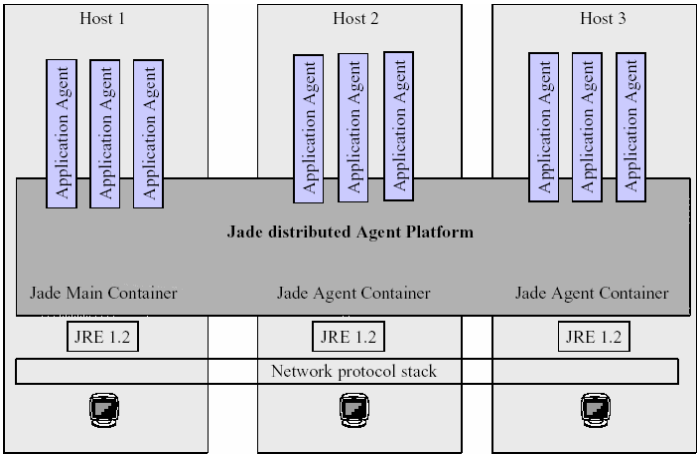


Figura 8: Plataforma JADE distribuida

Por otra parte, JADE define un modelo particular de agente basado en la multitarea, donde una tarea o comportamiento puede ser llevada a cabo concurrentemente con otras.

Un agente JADE puede pasar, desde que es creado hasta que se destruye, por varios estados de acuerdo a lo establecido en FIPA. Es su ciclo de vida:

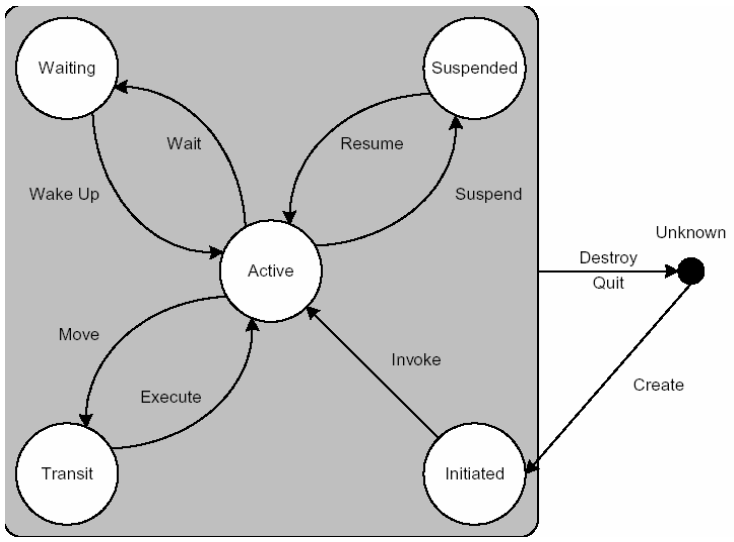


Figura 9: Ciclo de vida de un agente

Inmediatamente después de ser creado, un agente se encuentra en estado **INICIADO,** en este estado permanece hasta que el AMS lo registra y le asigna su identificación; mientras tanto no puede comunicarse con otros agentes. De este estado el agente transita a **ACTIVO,** donde puede llevar a cabo todas sus tareas y establecer las comunicaciones que necesite con el resto. Un agente puede ser **SUSPENDIDO**, deteniendo todas sus actividades o comportamientos, o puede permanecer **ESPERANDO** por algún evento, como por ejemplo la recepción de un mensaje. Finalmente, un agente que está emigrando a otra localización en otra máquina pasa al estado **EN TRÁNSITO** hasta que llega allí.

Actualmente existen algunas metodologías que ayudan al ingeniero a analizar y construir software basado en agentes:

- MAS-COMMON-KADS [http://www.gsi.dit.upm.es]: Es una extensión de COMMON-KADS para el desarrollo de agentes. Se basa en la definición de seis modelos o puntos de vista del sistema:
  - Modelo de organización.
  - Modelo de tareas.
  - Modelo de agentes.
  - Modelo de comunicación.
  - Modelo experto.
  - Modelo de diseño.
- MESSAGE [http://www.eurescom.de/~public-webspace/P900-series/P907/index.htm]: Se basa en MAS-COMMON-KADS, GAIA, AUML, y emplea RUP y UML. Especifica las fases de análisis del problema y diseño del sistema y como antes presenta distintas vistas del producto:
  - Metas/tareas.
  - Agente/rol.

- Interacción.
- Dominio.

- GAIA

  [Wooldridge et al. 2000] [http://citeseer.nj.nec.com/wooldridg00gaia.html]: Es una metodología orientada a agentes para el análisis y diseño de alto nivel; pretende ser general a la vez que intuitiva, manteniéndose siempre neutral con respecto al dominio de la aplicación y a la arquitectura específica de agentes. GAIA parte de un punto de vista de sistemas multi-agente, con una organización computacional formada por varios roles, proporcionando una visión de macro-nivel (nivel de relaciones sociales entre agentes) y de micro-nivel (nivel de agente).

  Los sistemas que mejor se adaptan a esta metodología son aquellos que cuentan con las siguientes características:

  - Agentes complejos, con un uso significativo de recursos computacionales.
  - Sistemas cuyo objetivo es la maximización de alguna medida global.
  - Agentes y plataformas heterogéneas en cuanto a arquitecturas, lenguajes de programación y técnicas.
  - Sistemas con una estructura estática en la que las relaciones entre agentes no cambian en tiempo de ejecución.
  - Las capacidades y servicios que proporcionan los agentes también son estáticos.
  - Sistemas con menos de cien tipos diferentes de agentes.

En general, cualquier metodología empleada en el proceso de desarrollo de software orientado a agentes debería ser lo suficientemente flexible como para no imponer restricciones respecto al ciclo de vida escogido. GAIA se ajusta a este requisito y además, aunque no define pautas para diseño detallado, establece un conjunto de modelos de alto nivel que sirven de punto de partida para el uso de técnicas de diseño detallado más tradicionales.

Los productos resultantes de la aplicación de GAIA a las fases de análisis y diseño de alto nivel de un sistema se muestran en la figura siguiente:

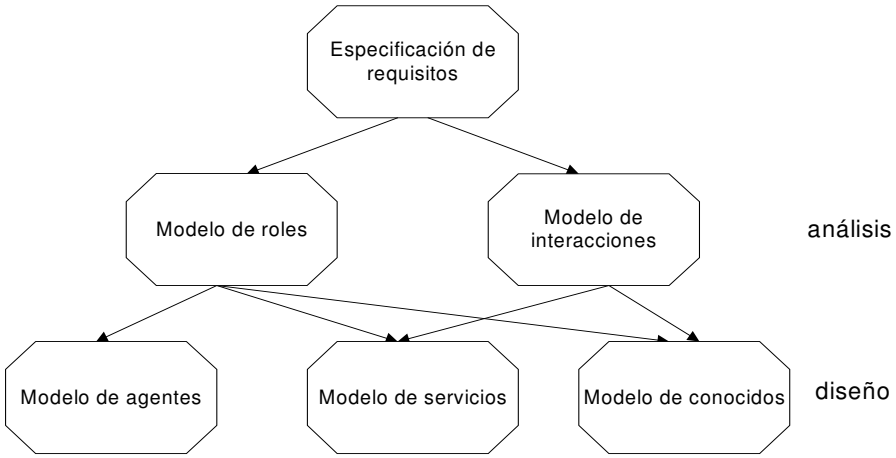


Figura 10: Modelos en GAIA

En la fase de análisis se trata de comprender el sistema y su estructura y extraer un esquema de organización compuesto por una colección de roles que establecen relaciones entre sí. El proceso de análisis se resume en los siguientes pasos:

1. Identificar los roles intervinientes en el sistema: Cada rol puede definirse como la descripción abstracta de la funcionalidad esperada para una entidad. Partiendo de un modelo de roles prototipo general, se llegará, tras varios ciclos de refinamiento, a un modelo de roles completamente elaborado que detalla aspectos como las responsabilidades (funcionalidad) del rol y sus permisos o recursos de los que hace uso.

   Existen dos tipos de responsabilidades asociadas a cada rol; las de vitalidad, expresiones formadas por la combinación de acciones que el rol debe realizar

por sí mismo (actividades) y acciones que requieren la interacción con otros (protocolos), y que describen las tareas que se han de realizar; y las de seguridad, que muestran aquellas condiciones que han de cumplirse para garantizar el éxito de las tareas a ejecutar.

Del mismo modo, existen tres categorías de permisos que establecen el uso que el rol hace de un determinado recurso, a saber, leer, modificar o generar uno nuevo y si ese recurso es proporcionado por una fuente externa o no.

2. Por cada rol, identificar y documentar los protocolos asociados: Cada protocolo define una interacción entre roles, y todos juntos constituyen el modelo de interacción. Un protocolo se define mediante un conjunto de atributos:

   a. Propósito: descripción textual de la interacción.

   b. Iniciador: rol responsable del inicio de la interacción.

   c. Respondedor: rol con el que el iniciador interactúa.

   d. Entradas: información empleada por el iniciador al ejecutar el protocolo.

   e. Salidas: información proporcionada por/al rol respondedor durante la interacción.

   f. Procesamiento: descripción textual de cualquier procesamiento llevado a cabo por el iniciador durante la interacción.

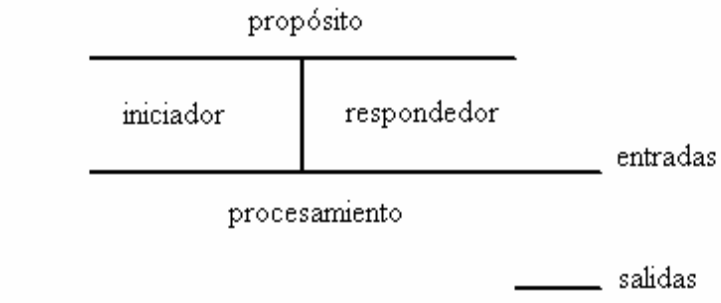


Figura 11: Representación de protocolos

3. Refinar los modelos iterando los pasos 1 y 2.

De los modelos extraídos en el análisis, la fase de diseño del sistema intentará elaborar un conjunto de abstracciones de bajo nivel que definan una sociedad de agentes.

El proceso de diseño se resume en los siguientes pasos:

1. Crear un modelo de agentes asociando los roles con tipos de agente e identificando las instancias de los mismos que se crearán en tiempo de ejecución.
2. Crear el modelo de servicios o funciones de cada rol a partir de las actividades, protocolos y responsabilidades de vitalidad y seguridad de los roles.
3. Crear el modelo de conocidos estableciendo los vínculos de comunicación entre los distintos tipos de agentes a partir de los modelos de interacción y roles.

2.6. Técnicas de integración en la construcción de entornos virtuales

Finalmente, al concluir este capítulo y después de hablar sobre multitud de técnicas y herramientas de lo más variadas y heterogéneas para construir entornos virtuales, surge la pregunta de cómo es posible compaginarlas para aprovechar las ventajas de todas y cada una. Pues bien, dado que lo que se presenta es un sistema distribuido, será necesario hacer uso de modelos o arquitecturas que ofrezcan soporte para la distribución e interconexión de los distintos componentes hardware y software remotos. Al hablar de los sistemas multi-agente, hemos introducido necesariamente un tipo específico de sistema distribuido que, actualmente, está en auge por su potencia y flexibilidad, especialmente cuando los agentes tienen capacidad para viajar a través de una red informática. Supone, pues, un nuevo modelo de programación remota en el que son los "cálculos" los que se mueven hacia los "datos" y no al revés.

Pero, ¿cómo se relaciona la faceta "inteligente" de un VET con su faceta "sensorial"? ¿cómo se transmite la información acerca de lo que se ve, se oye y/o se siente en un entorno virtual?

Una de las arquitecturas más extendidas en el campo de las aplicaciones en red se basa en el modelo cliente/servidor estructurado en tres niveles, donde el tercer nivel está formado por un componente intermediario o *middleware* que permite que la comunicación sea transparente.

Uno de estos *middleware* es CORBA (*Common Object Request Broker Architecture*) [http://www.corba.org/]. CORBA es un estándar que define como se realiza la comunicación entre objetos remotos heterogéneos independientemente del lenguaje de implementación en el que fueron programados, de su localización física, del tipo de máquina en la que residan, del sistema operativo, etc.

CORBA fue ideado en 1990 por el OMG (*Object Management Group*), organización formada por fabricantes de hardware y software, universidades y usuarios, que se dedica a la creación de especificaciones. El OMG definió la especificación OMA (*Object Management Architecture*) de la que CORBA forma parte. OMA es una arquitectura que consta de:

- Modelo de objetos: Definición formal y abstracta de toda la terminología asociada a los objetos distribuidos en entornos heterogéneos.
- Modelo de referencia: Define las interacciones entre objetos.

Algunas de las principales implementaciones del estándar CORBA son:

- Java IDL de Sun Microsystems.
- OrbixWeb 3.X de IONA Technologies.
- Visibroker for Java 3.X.
- ORBacus de IONA Technologies.

3. Descripción general del proyecto

3.1. Descripción del proyecto MAEVIF

MAEVIF, acrónimo que significa "Modelo para la Aplicación de Entornos Virtuales Inteligentes a la Formación" [de Antonio et al. 2005], es un proyecto financiado por el Ministerio de Ciencia y Tecnología español y desarrollado en el Laboratorio Decoroso Crespo de la Facultad de Informática de la Universidad Politécnica de Madrid.

En el capítulo de introducción se anticiparon las ventajas de la aplicación de entornos virtuales tridimensionales en el entrenamiento. Estos entornos permiten a los estudiantes navegar e interactuar con una representación virtual de un entorno real en el que aprenden a realizar ciertas tareas, individual o colectivamente. En MAEVIF se da una vuelta de tuerca más a dichos entornos al combinarlos con sistemas ITS para dar lugar a lo que se denomina IVET[1].

Los IVETs son capaces de supervisar el aprendizaje, y en ellos existen componentes inteligentes de tutoría que a menudo adoptan representaciones virtuales y cohabitan en el entorno junto a la representación virtual[2] de los estudiantes.

Lo que en MAEVIF se propone es la construcción de una arquitectura o modelo para el desarrollo de esta clase de sistemas, basada en un conjunto de agentes software cooperantes. Entre las distintas clases de agentes presentes en esta arquitectura, están aquellos que se encargan de simular el comportamiento real del sistema, los que registran las acciones de los estudiantes, los que supervisan el aprendizaje de los mismos, los que gestionan la interacción con el entorno virtual a través de las diferentes interfaces de usuario que puedan ser usadas, como guantes, HMDs, joysticks, además de monitor, teclado y ratón, y por último los agentes responsables de controlar la comunicación con dichos dispositivos.

[1] Del ingles "Intelligent Virtual Environment for Training".

[2] La representación de una persona en un entorno virtual es comúnmente conocida como avatar.

El área de la informática dedicada a la construcción de entornos virtuales tiene sus orígenes en la década de los 90, por lo que todavía es relativamente pronto como para que la materia haya sido sometida a procesos de estandarización. La mayoría de los sistemas construidos hasta la fecha han sido diseñados a medida, para dar solución a un único problema específico y muy concreto, olvidando en muchos casos los principios y técnicas de la ingeniería del software. El resultado de esto son sistemas poco flexibles, no reutilizables y difíciles de mantener.

Pero actualmente el tamaño y complejidad de los entornos virtuales hacen imprescindible el uso de la ingeniería del software para poder abordarlos en toda su magnitud. Esta es la filosofía con la que MAEVIF plantea su modelo, con la intención de constituir una base para la definición de arquitecturas estandarizadas para IVETs. El ideal es que estas arquitecturas se basen en componentes individuales que puedan ser reutilizados y ampliados y que, fácilmente, sean integrados unos con otros por medio de interfaces bien definidas.

Existen algunos antecedentes importantes en la investigación sobre IVETs, como es el caso de STEVE [Rickel & Johnson, 1999] [Rickel & Johnson, 2000], del que ya hablamos en capítulos anteriores. Otro caso es el experimento, también realizado en la Universidad Politécnica de Madrid [Méndez et ál., 2003], en el que se trata de reutilizar, dentro de un mundo virtual, un agente pedagógico inteligente desarrollado independientemente.

La conclusión de estos experimentos, y algunos más, es que la integración de agentes pedagógicos dentro de entornos virtuales, con las arquitecturas empleadas hasta el momento, es tremendamente difícil, y produce resultados que no son totalmente satisfactorios.

En MAEVIF se busca un paradigma orientado a agentes en contraposición al modelo de objetos clásico. La razón se fundamenta en la convicción de que la creación de entornos altamente interactivos, poblados por entidades autónomas e inteligentes, además de los avatares controlados por los usuarios, requiere de un grado mayor de abstracción. El diseño tradicional orientado a objetos, o incluso orientado a componentes basados en CORBA o COM (*Component Object Model*), no proporciona las características de proactividad y reactividad que son necesarias en entornos con una interactividad alta. Sin embargo los agentes, además de estas cualidades, posibilitan un diseño fácilmente escalable y modificable, al poder cambiar unos agentes por otros que provean la misma funcionalidad, o añadir nuevos dinámicamente, sin necesidad de recompilar el sistema o reiniciarlo.

MAEVIF toma como base la arquitectura del proyecto MAPI (Modelo basado en Agentes cooperativos para sistemas inteligentes de tutoría con Planificación Instructiva) en la que se diseña el componente de tutoría de un ITS mediante un grupo de agentes que cooperan e intercambian información a través de la técnica de pizarra compartida, y que se corresponden con los módulos dibujados en la siguiente figura.

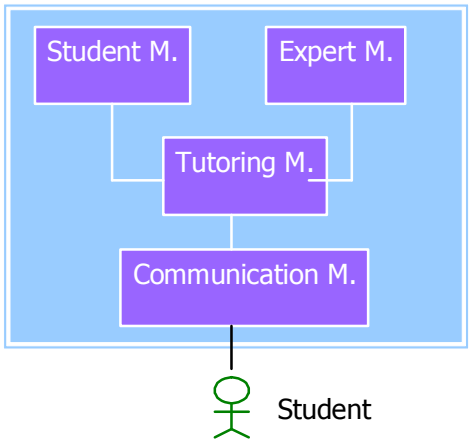


Figura 12: Arquitectura de un ITS

La analogía con un IVET se revela al identificar el componente (agente) inteligente o pedagógico del mismo con el modelo plasmado en la figura. Pero esto no es del todo exacto, ya que la arquitectura clásica de un ITS no se ajusta del todo a las singularidades de los IVETs:

- Los IVETs, a diferencia de los ITS, normalmente están poblados por más de un estudiante, puesto que a menudo son usados para el entrenamiento en equipo. Un IVET no sólo se tiene que adaptar a las necesidades del estudiante, sino también a las del equipo. De la misma manera, se debería modelar el conocimiento colectivo del equipo, además del individual.
- Los estudiantes están inmersos en el ITS, no fuera de él y cada uno debe tener una visión propia del entorno en función de su localización.
- En un IVET, la interfaz de comunicación con el usuario es el propio entorno gráfico, siendo la interacción con éste una parte más del entrenamiento. Sin embargo en un ITS, el módulo de comunicación es simplemente un medio y no un fin en sí mismo. Es necesario dotar al ITS con conocimiento acerca del entorno tridimensional, su estado y las posibilidades de interacción con él.

Para conseguir estas características, la arquitectura del ITS se extiende dando lugar a al diagrama mostrado en la **Figura 13**.

Como se ve en la figura, se ha añadido soporte multiusuario no sólo en el módulo de comunicación, sino con la creación de representaciones individuales del mundo virtual y de las interfaces de interacción, a través de dispositivos, para cada usuario. Además se ha introducido un nuevo módulo para gestionar el conocimiento acerca del entorno tridimensional. Así mismo, el módulo del estudiante contará con información acerca de cada alumno y también del equipo en su conjunto.

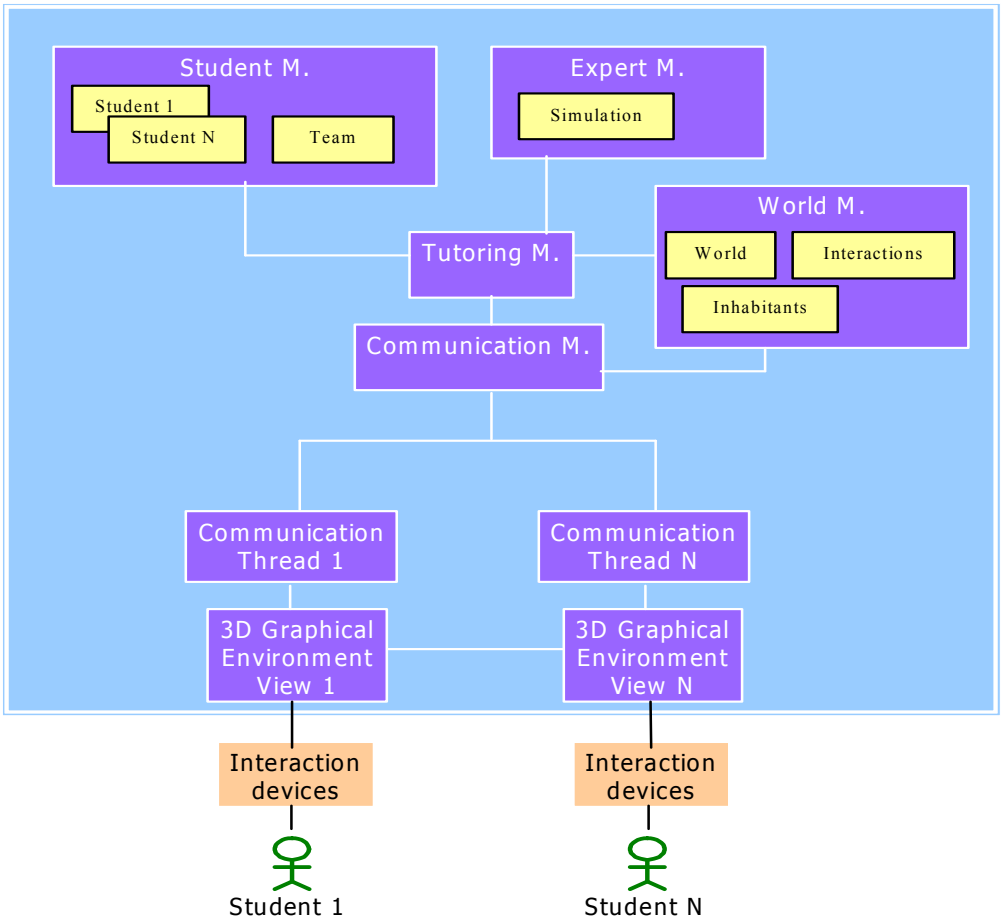


Figura 13: Arquitectura ITS extendida para un IVET

Trasladando este esquema de componentes a la correspondiente arquitectura de agentes tenemos un modelo basado en cinco clases principales de agente:

- Agente de comunicación.
- Agente representante del estudiante.
- Agente mundo.
- Agente experto.
- Agente de tutoría.

A su vez, estos agentes delegan algunas responsabilidades en agentes subordinados, como se explica a continuación.

De esta manera, el agente de comunicación confía algunas de sus tareas a un conjunto de agentes específicos por cada alumno. Estos agentes realizan las labores de gestión de las sesiones y de los dispositivos de interacción elegidos por el usuario.

El agente que representa a cada estudiante también distribuye su trabajo entre varios agentes especializados que se encargan de registrar el historial de interacciones de cada alumno, elaborar su perfil psicológico, útil para personalizar la enseñanza, almacenar el conocimiento adquirido y poder determinar las causas de sus errores.

El agente mundo, por su parte, gestiona información relativa al entorno tridimensional, tanto geométrica como semántica. Es decir, a través de subagentes específicos controla la localización de cada objeto y cada habitante en el entorno, conoce qué interacciones se pueden realizar con cada objeto y cuál serán sus efectos y, además, es capaz de construir caminos o trayectorias en el espacio tridimensional sin colisionar con obstáculos.

De la misma forma que se almacena el conocimiento que posee el estudiante, o el conocimiento sobre el mundo, el sistema también alberga conocimiento experto sobre la materia concreta que se está enseñando. El agente experto gestiona esta información, y como es lógico, es un componente, dentro de un IVET, que varía en función de la actividad desarrollada. Este agente, mediante sus subordinados, usa técnicas de planificación inteligente, como STRIPS (*STanford Research Institute Problem Solver)*, para elaborar planes de actuación, con el objetivo de encontrar la mejor forma de abordar los distintos procedimientos y poder cotejarlos con las acciones llevadas a cabo por el usuario. Además puede simular diferentes caminos de actuación para resolver problemas sobre aquello que se está enseñando.

Por último, el agente de tutoría, encargado de coordinar el proceso de enseñanza, reparte sus responsabilidades entre la gestión de lo que podríamos llamar "plan de estudios" y las distintas estrategias de tutoría a seguir.

La siguiente figura muestra la arquitectura final de un IVET basado en la organización de agentes comentada:

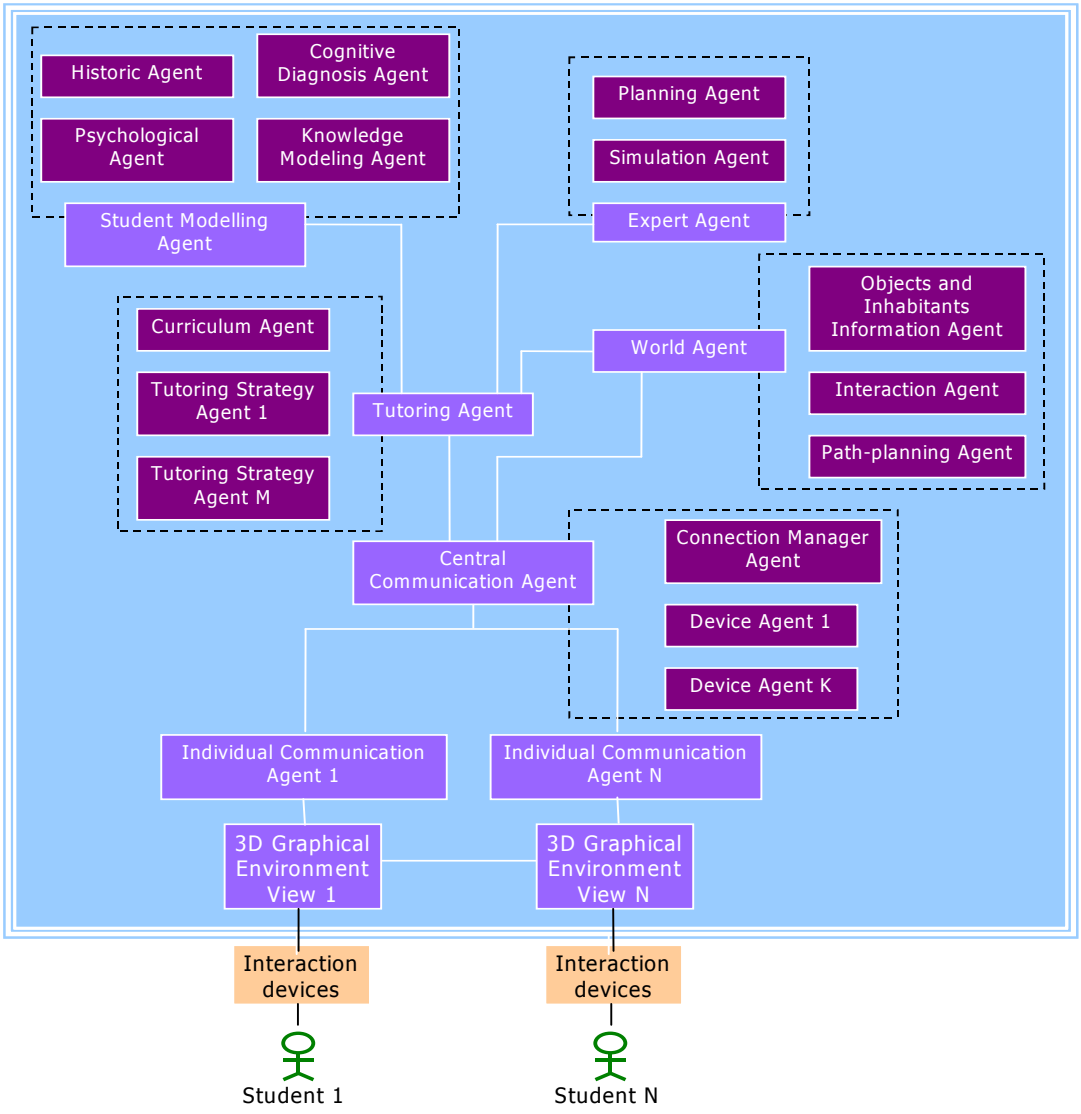


Figura 14: Arquitectura basada en agentes para IVETs.

Una vez que tenemos una visión esquemática de los distintos componentes del modelo podemos comprender su funcionamiento.

Como hemos dicho, el paradigma propuesto en MAEVIF ofrece soporte para el entrenamiento individual y en equipo. Imaginemos pues, que un cierto grupo de profesionales se han de instruir en la realización de cierta actividad conjunta; cada persona deberá asumir el rol asociado a uno de los participantes en la tarea y después conectarse al sistema.

Los participantes en una actividad no tienen por qué localizarse en el mismo lugar físico ni disponer de los mismos dispositivos de interacción con el entorno; el sistema será el encargado de adaptarse a las circunstancias de cada uno y de proporcionarle su vista particular del entorno. En concreto, esto último tiene su reflejo en la arquitectura del sistema, ya que, aunque cada usuario percibe el entorno desde el punto de vista de los ojos de su avatar asociado, el entorno es el mismo para todos y los efectos de las acciones de cada uno serán visibles por los demás. Para conseguir esto, se podría tener una representación común del escenario 3D a partir de la cual se elaboraría cada vista particular que, posteriormente, sería enviada a cada estudiante. De esta forma las modificaciones en la primera provocarían la modificación de las últimas. Esta alternativa requiere de un componente centralizado que gestione la coherencia de todas las vistas, pudiéndose provocar un efecto de "cuello de botella" en el sistema por el volumen de información que habría de procesar este componente. La solución adoptada para evitar este inconveniente es hacer que sea cada vista particular la encargada de comunicar al resto, mediante "broadcast", las modificaciones efectuadas en ella. De esta forma tan sólo se transmite a los demás la información acerca de los cambios producidos y no de todo el entorno, con el consiguiente beneficio en cuanto a eficiencia.

Resumiendo, cuando todos los participantes en la actividad se hayan conectado, cada uno estará inmerso (en mayor o menor grado dependiendo de los dispositivos de interacción que utilice) en el entorno virtual y podrá observar los avatares de sus compañeros, el avatar correspondiente a su tutor virtual y, aunque no se verá a sí mismo, verá y podrá manejar su propia mano. Hay muchas formas posibles de representar a uno mismo en el escenario virtual, pero ésta, que se puede llamar "de primera persona", es la que se aproxima más a la sensación de estar "en el cuerpo del avatar".

El hecho de que el sistema tenga que esperar la conexión de todos los participantes supone que estos tendrán que adaptar mutuamente su velocidad de aprendizaje para participar en la misma actividad. Sin embargo, existe la opción de que el sistema simule el papel de uno o más estudiantes, de manera que no sea necesario que se conecte todo el equipo y se pueda adecuar el ritmo a un estudiante concreto.

Normalmente la actividad a realizar consiste en lograr un objetivo partiendo de una situación inicial. Los participantes, dado que ya están dentro del escenario, podrían empezar a trabajar por su cuenta y riesgo, pero no olvidemos que el sistema supervisa sus acciones y ha de poder guiarles en su aprendizaje. Por lo tanto, una vez planteado el problema, lo primero es que el sistema calcule la solución, así que el agente experto, en colaboración con los agentes de planificación, simulación, planificador de trayectorias, e interacción, elaboran un plan de actuación para resolverlo. Este plan está formado por una secuencia de acciones a seguir para alcanzar el objetivo. El hecho de construir este plan en tiempo real significa que no existen soluciones predefinidas, y por tanto el número de problemas que se le pueden plantear al alumno no está restringido. Además, dado que un problema no tiene por qué tener una única solución, el sistema de tutoría es capaz de decidir dinámicamente si el estudiante podrá alcanzar el objetivo, a pesar de haberse desviado del plan previamente calculado por el agente experto, pues de esto dependen las decisiones de tutoría.

Con la solución calculada, el IVET entra en la fase de supervisión, en la que:

1. Los agentes de comunicación de cada estudiante observan el comportamiento del alumno e informan sobre él al agente de tutoría.
2. El agente de tutoría comparará dicho comportamiento con el esperado, que previamente ha sido calculado por el agente experto, y lo evalúa.
3. El agente historiador registra las acciones realizadas por cada estudiante.
4. Los agentes de diagnóstico y modelado del conocimiento del estudiante actualizan su información.
5. El agente psicológico infiere características psicológicas a partir del comportamiento mostrado.
6. El agente de estrategia de tutoría decidirá qué debe hacer el tutor en función de las acciones del alumno, su conocimiento, su estado psicológico y el plan de estudios. La decisión tomada puede ser esperar por nuevas acciones, dar un consejo o una explicación, felicitar al estudiante, etc.

De este modo funciona el modelo de IVET diseñado en MAEVIF, partiendo de la hipótesis de un entorno virtual predefinido con un plan de actividades fijado sobre el que se impartirán las enseñanzas. La ventaja de MAEVIF es que define una infraestructura que permite diseñar fácilmente nuevos IVETs sin más que seleccionar y configurar el comportamiento del conjunto de agentes deseado (por ejemplo el agente de estrategia de tutoría), proporcionar la información correspondiente al entorno virtual, la materia objeto de la enseñanza, las acciones posibles y sus efectos, y en el peor de los casos crear agentes adicionales e incorporarlos al sistema.

En la práctica, la arquitectura de MAEVIF ha sido implementada con herramientas y técnicas muy heterogéneas.

El sistema multi-agente ha sido desarrollado con la plataforma JADE. Los entornos virtuales y los avatares han sido modelados usando 3D Studio Max e importados dentro de una aplicación C++ que hace uso de las librerías gráficas OpenGL. Para la comunicación entre las distintas vistas de los entornos gráficos de cada usuario se emplea Microsoft DirectPlay. La implementación de las interfaces de acceso mediante dispositivos se comenta en capítulos posteriores, y por último, la comunicación entre el entorno gráfico y el sistema de tutoría se realiza mediante la implementación CORBA de ORBacus.

3.2. Sub-modelo para la gestión de interfaces de acceso e interacciones en MAEVIF.

Dentro de la arquitectura propuesta por MAEVIF, este trabajo se centra únicamente en dos aspectos.

1.- Por una parte, el desarrollo de un módulo que permita la selección del conjunto de dispositivos de realidad virtual que se desean utilizar y el acceso a través de ellos al entorno tridimensional. MAEVIF, potencialmente, da soporte para la incorporación de cualquier tipo de dispositivo; por eso, en el módulo a construir, la flexibilidad y la escalabilidad han de ser la filosofía de diseño a seguir. El capítulo siguiente detalla el conjunto de dispositivos que se han elegido para su integración en MAEVIF.

Este módulo será parte integrante de la aplicación que contiene el entorno gráfico y a través de la cual cada estudiante se conecta al sistema. Con respecto a esto se ha introducido una modificación en la arquitectura de agentes de MAEVIF; en la **Figura 14** se muestra un conjunto de agentes subordinados al agente de comunicación cuya misión es gestionar de forma independiente cada dispositivo de interacción. En la práctica esta organización supone un aumento considerable en la tasa de tráfico de información entre la aplicación gráfica y la plataforma de agentes, con la consiguiente ralentización del sistema, ya que cada dispositivo genera nuevos datos con una frecuencia muy alta y dichos datos habrían de ser transmitidos a los correspondientes agentes. Por eso se ha tomado la decisión de trasladar la funcionalidad de estos agentes al módulo en cuestión.

Otra de las máximas que se intentará seguir en su desarrollo será la de hacer que tanto la representación geométrica del mundo virtual como la interfaz de ventanas de la aplicación sean independientes del módulo de dispositivos en la medida de lo posible. Esta decisión se toma a efectos de modificabilidad del diseño. Ya que este elemento es potencialmente más susceptible a los cambios que los otros, conviene que éstos últimos no dependan en exceso de él.

2.- Por otro lado, en la descripción de MAEVIF se habló de un subcomponente del IVET que mantenía información referente al entorno tridimensional y a las posibilidades de interacción con él. Este elemento se plasmaba en el *Agente Mundo*, y entre los agentes subordinados en los que delegaba su responsabilidad estaba el *Agente de Interacción* o *de Actuación*. Es éste último el que se disecciona en este trabajo.

Como se verá, entre las obligaciones del *Agente de Interacción* está la de gestionar el conocimiento acerca de aquellas acciones básicas que se pueden realizar con cada objeto del entorno. Este conocimiento será útil tanto en la fase de construcción de planes como posteriormente en la de supervisión.

Se explicará también cómo, en el desarrollo de este agente y de aquellos con los que coopera, es necesaria la implementación de un mecanismo para compartir información de manera concurrente, adicionalmente a los mensajes ACL entre agentes.

Aunque aparentemente las dos partes integrantes de este trabajo no guardan una relación directa en el sistema, sin embargo, sí encontramos nexos que ponen de manifiesto la naturaleza común de ambas.

El fin último de los dispositivos de acceso a un entorno virtual es posibilitar la "navegación" por él, e interaccionar con los objetos que en él se encuentran. Estas interacciones se traducen en acciones que el usuario desencadena dentro del entorno y que, como es lógico, tienen efectos que también han de ser simulados. Por esto surge la necesidad de albergar conocimiento acerca de las acciones y reacciones asociadas a la interacción con cada objeto. Este conocimiento será empleado en las simulaciones, en el cálculo de planes de actuación, en la elaboración de explicaciones, respuestas a preguntas, etc.

4. Dispositivos de realidad virtual

4.1. Introducción

La elección de dispositivos de realidad virtual sobre los que se ha investigado en el marco del proyecto MAEVIF, obedece, sin duda, a razones prácticas, teniendo en cuenta las estimaciones de costes, pero sobre todo, a razones funcionales, es decir, analizando la funcionalidad que ofrece el sistema, se han buscado aquellos dispositivos cuyas características favorezcan dicha funcionalidad, aportando, por ejemplo, realismo a los procesos de entrenamiento al permitir la inmersión visual y auditiva en los entornos virtuales, o dotando al usuario de medios para manipular e interaccionar directamente con los objetos virtuales.

La primera y más esencial capacidad que interesa a una persona sumergida en un ambiente virtual, es la de poder moverse por él, recorrerlo, explorarlo. La manera más habitual de realizar esta exploración es por medio del teclado. El usuario, sentado frente a la pantalla del computador, utiliza un conjunto de teclas predefinido a las que se ha asociado un desplazamiento o un giro en una dirección determinada. El resultado es el cambio de posición u orientación del punto de vista propio, lo que da la sensación de movimiento a través del escenario virtual.

Esta sensación de desplazamiento, que se puede conseguir por medio del teclado, mejora progresivamente al introducir nuevos dispositivos que incrementan el grado de inmersión en el entorno.

La evolución del teclado en este sentido podrían ser los dispositivos de tipo joystick. Éstos, a través de una palanca de mando, pueden llegar a ofrecer los mismos grados de libertad que el teclado, con la ventaja de no tener que memorizar teclas. Además, actualmente muchos modelos de joysticks poseen mecanismos de *force-feedback* a través de los cuales se pueden simular efectos de quinestética como por ejemplo inercia, oposición al movimiento, choque con obstáculos, etc.

Tanto teclados como joysticks comparten el inconveniente de no aislar a la persona del entorno real. En MAEVIF, este aislamiento se consigue usando un HMD con tecnología CRT que cuenta, además, con auriculares de sonido estereofónico. Con este casco, el usuario deja de percibir completamente el mundo real a través de la vista y el oído, y "entra" en el mundo virtual encarnándose en la piel de su avatar.

El uso del HMD combinado con sensores que marquen la posición y orientación de la cabeza permite prescindir del joystick y/o el teclado, y pasar a realizar los desplazamientos, dentro del entorno virtual, directamente caminando o girando el cuerpo o la cabeza en el entorno real. Si se hace este uso, conviene que exista una correspondencia entre la disposición de paredes y obstáculos en los dos mundos para evitar que el portador del casco choque con objetos que no existen en el mundo virtual y viceversa.

Por otro lado, aunque sin tanta trascendencia como el desplazamiento, están los dispositivos que permiten la interacción con los objetos virtuales. En MAEVIF se ha utilizado la combinación de guante y sensor de posición para dar soporte a la representación de la mano virtual del usuario. Con esta mano es posible tocar, señalar, seleccionar objetos, etc.

4.2. Sensores de posicionamiento y orientación en el espacio

4.2.1. Introducción

Como ya se vio, existe una gran variedad en cuanto a sistemas de medida de posición y orientación o *Trackers*. Este trabajo centra su investigación en uno en particular, *The Flock of Birds* (en adelante FOB), fabricado por Ascension Techonology Corporation [http://www.ascension-tech.com/].

FOB es un dispositivo de medición con seis grados de libertad capaz de detectar la posición y orientación de hasta treinta sensores situados dentro de un radio de aproximadamente 1.2 metros en torno a un transmisor. Dicho transmisor emite un campo magnético continuo que es medido simultáneamente por todos los sensores para calcular, a partir de él, su posición y orientación y hacerla llegar al computador.

Figura 15: Flock of Birds

FOB ha sido usado en numerosas aplicaciones comerciales que requieren una interfaz de interacción humana con el computador. Algunos de sus usos son:

- Seguimiento del movimiento de la cabeza en simuladores de vuelo.
- Seguimiento de cabeza, manos y cuerpo en juegos de realidad virtual.
- Control de imágenes tridimensionales en tiempo real.
- Mediciones biomecánicas sobre partes anatómicas.
- Control de robots.
- Interacción con imágenes virtuales en tiempo real.

En concreto, la información que proporciona cada sensor está compuesta por las coordenadas cartesianas (x, y, z) de posición con respecto al transmisor, expresadas en pulgadas, metros, o cualquier otra unidad de medida a voluntad del usuario, y la orientación, también tomando como referencia el transmisor, expresada ésta, bien como una matriz de rotación de dimensiones 3x3, bien como los tres ángulos de Euler (*azimuth, elevation, roll*) o como cuatro cuaternios (q_0, q_1, q_2, q_3). Con cualquiera de estos formatos es posible determinar la dirección de los ejes que forman el sistema de referencia espacial de cada sensor en relación al sistema de referencia del transmisor.

Por omisión, el sistema de referencia sobre el que trabaja FOB es el que muestra la siguiente figura:

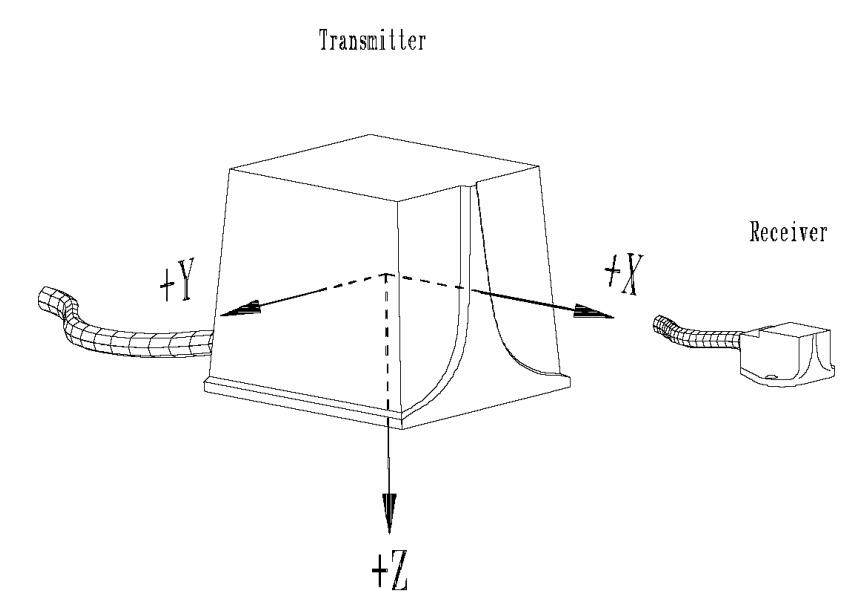


Figura 16: Sistema cartesiano de referencia en FOB

La orientación viene dada por los ángulos que formarían los ejes de coordenadas, tomados dos a dos, de los sistemas de referencia situados respectivamente en el centro del transmisor y en el centro de cada sensor.

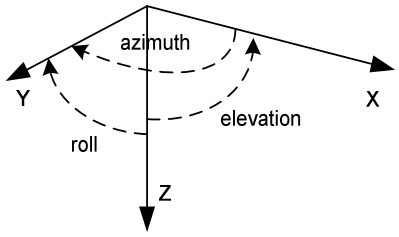


Figura 17: Ángulos de orientación en FOB

La combinación de estos ángulos nos permite determinar la dirección y sentido de los tres ejes del sistema de coordenadas. Esta combinación se representa matemáticamente por medio de una matriz de rotación que resulta de la multiplicación de las correspondientes matrices de rotación sobre cada eje:

$$M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(R) & \sin(R) \\ 0 & -\sin(R) & \cos(R) \end{pmatrix} * \begin{pmatrix} \cos(E) & 0 & -\sin(E) \\ 0 & 1 & 0 \\ \sin(E) & 0 & \cos(E) \end{pmatrix} * \begin{pmatrix} \cos(A) & \sin(A) & 0 \\ -\sin(A) & \cos(A) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Ecuación 1

Donde las variables A, R y E son los valores medidos, por cada sensor, de los ángulos de Euler *azimuth*, *roll* y *elevation* respectivamente.

La matriz queda como sigue:

$$M = \begin{pmatrix} \cos(E) * \cos(A) & \cos(E) * \sin(A) & -\sin(E) \\ -\cos(R) * \sin(A) + \sin(R) * \sin(E) * \cos(A) & \cos(R) * \cos(A) + \sin(R) * \sin(E) * \sin(A) & \sin(R) * \cos(E) \\ \sin(R) * \sin(A) + \cos(R) * \sin(E) * \cos(A) & -\sin(R) * \cos(A) + \cos(R) * \sin(E) * \sin(A) & \cos(R) * \cos(E) \end{pmatrix}$$

Ecuación 2

Los nueve elementos de esta matriz pueden ser determinados directamente por FOB sin necesidad de que las aplicaciones realicen ningún cálculo, a excepción de los que se indican a continuación.

Normalmente las aplicaciones basadas en gráficos tridimensionales utilizan dos clases de sistemas de referencia espacial, dependiendo de la biblioteca de manipulación de gráficos empleada. Estos dos sistemas se denominan respectivamente *right-hand* y *left-hand*[1].

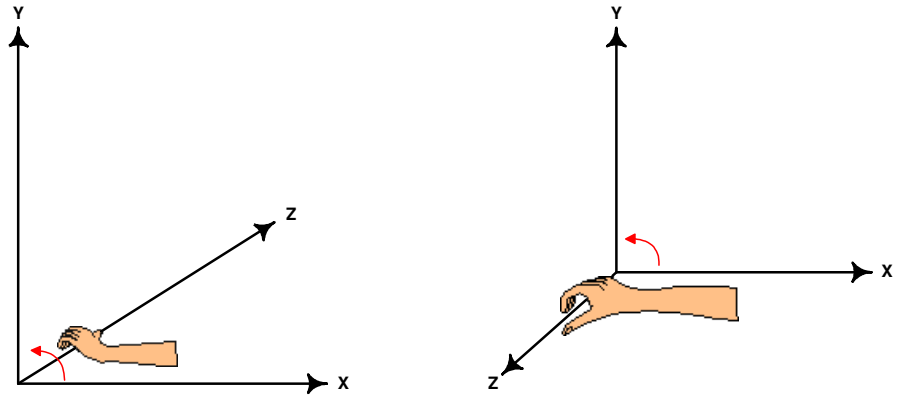


Figura 18: Sistemas cartesianos *left-hand* y *right-hand*

Sin embargo, ninguno de estos sistemas coincide con el definido por FOB, por lo que, para que los datos medidos sean correctamente interpretados por las aplicaciones, es necesario realizar algunas transformaciones a la matriz de la **Ecuación 2**. Adicionalmente, es común que las librerías gráficas usen matrices de dimensiones 4x4, combinando en una sola matriz las coordenadas de rotación y posición de cada punto del espacio.

Para un sistema de coordenadas *left-hand* la matriz M se transforma en:

$$M' = \begin{pmatrix} M(2,2) & -M(2,3) & M(2,1) & 0 \\ -M(3,2) & M(3,3) & -M(3,1) & 0 \\ M(1,2) & -M(1,3) & M(1,1) & 0 \\ y & -z & x & 1 \end{pmatrix} \begin{matrix} eje\,x \\ eje\,y \\ eje\,z \\ Vector\ de\ posición \end{matrix}$$

Ecuación 3

[1] Estos nombres, como muestra la figura 18, se derivan de las respectivas reglas memotécnicas de cada sistema.

Para un sistema de coordenadas *right-hand* la matriz M se transforma en:

$$M' = \begin{pmatrix} M(2,2) & M(2,3) & -M(2,1) & 0 \\ M(3,2) & M(3,3) & -M(3,1) & 0 \\ -M(1,2) & -M(1,3) & M(1,1) & 0 \\ -y & -z & x & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ \\ \end{matrix} \begin{matrix} eje\ x \\ eje\ y \\ eje\ z \\ Vector\ de\ posición \end{matrix}$$

Ecuación 4

En ambos casos las coordenadas **x**, **y**, **z** corresponden a los datos de posición recogidos por el sensor.

Estas matrices establecen las correspondencias entre el sistema de coordenadas definido por FOB y los definidos en los mundos virtuales creados por las aplicaciones. Pero para que estas correspondencias sean efectivas, es preciso alinear el **eje x** del transmisor FOB con el **eje z** del mundo virtual, es decir, en función de la aplicación, será necesario orientar el transmisor de distinta manera.

Dado que los sensores miden su rotación y posición con respecto al sistema de referencia fijado por el transmisor en términos absolutos, si a un objeto cualquiera en el espacio le aplicamos las transformaciones definidas por la matriz calculada para un sensor de acuerdo a la **Ecuación 5**, estaremos **acumulando** en el objeto una rotación y una traslación igual a la que el sensor tiene con respecto al transmisor:

$$\begin{pmatrix} x^* & y^* & z^* & 1 \end{pmatrix} = \begin{pmatrix} x & y & z & 1 \end{pmatrix} * M' \Leftrightarrow \begin{pmatrix} x^* \\ y^* \\ z^* \\ 1 \end{pmatrix} = (M')^T * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Ecuación 5

Esto es relevante porque, si lo que se desea es que un cierto objeto virtual se mueva y oriente siguiendo los movimientos del sensor, no tendremos el resultado esperado si hacemos aplicaciones sucesivas de la **Ecuación 5** sobre las coordenadas del objeto. Por el contrario, lo que deberemos hacer es igualar sucesivamente, la matriz de rotación y posición del objeto con la matriz calculada por el sensor en cada momento.

A modo de ejemplo, podemos ver cómo calcular la nueva orientación del eje x en un sistema *left-hand* tras una cierta rotación:

$$\begin{pmatrix} x^* & y^* & z^* & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} M(2,2) & M(2,3) & -M(2,1) & 0 \\ M(3,2) & M(3,3) & -M(3,1) & 0 \\ -M(1,2) & -M(1,3) & M(1,1) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ecuación 6

Al margen del cálculo con matrices, podremos obtener el valor de cada uno de los ángulos de Euler que determinan la orientación en ambos sistemas de referencia aplicando la siguiente tabla de equivalencias:

Ángulos de Euler	**Ejes de rotación**		
	Flock of Birds	***left-hand***	***right-hand***
Azimuth	Z	-Y	-Y
Elevation	Y	X	-X
Roll	X	Z	Z

De esta forma, siendo $\mathbf{a_1}$, $\mathbf{a_2}$ y $\mathbf{a_3}$ los ángulos de rotación con respecto a los ejes **x, y, z,** respectivamente, de un cierto objeto en el espacio virtual, y ***Azimuth***, ***Elevation*** y ***Roll*** los ángulos de rotación medidos por FOB sobre los ejes **z, y, x,** respectivamente, tal y como muestra la tabla, en un sistema *left-hand* se cumple que:

$$\begin{pmatrix} a_1 & a_2 & a_3 \end{pmatrix} = \begin{pmatrix} Elevation & -Azimuth & Roll \end{pmatrix}$$

Ecuación 7

Y análogamente, en un sistema *right-hand* tenemos que:

$$\begin{pmatrix} a_1 & a_2 & a_3 \end{pmatrix} = \begin{pmatrix} -Elevation & -Azimuth & Roll \end{pmatrix}$$

Ecuación 8

4.2.2. Arquitectura Hardware

A continuación se realiza una ligera descripción de los módulos hardware de *Flock of Birds*, su funcionamiento básico y sus distintas configuraciones. Para descripciones más amplias, se remite al lector a los manuales ofrecidos por el fabricante a través de Internet en su URL http://www.Ascension-tech.com/.

La figura siguiente muestra el diagrama de componentes físicos que forman un sistema FOB.

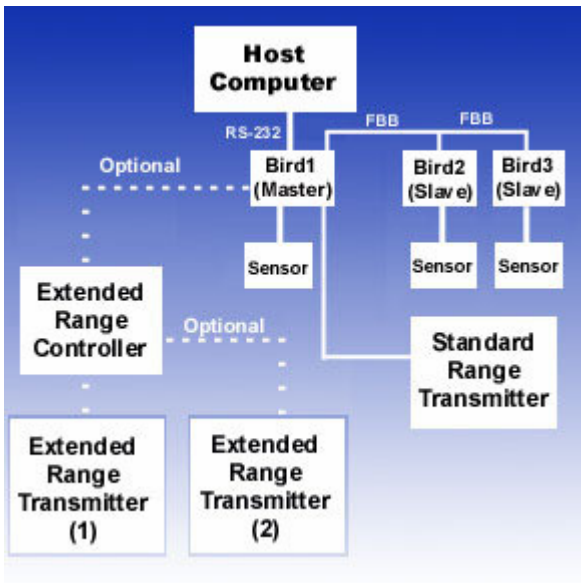


Figura 19: Arquitectura hardware en FOB

En primer lugar, tenemos una serie de unidades electrónicas denominadas *Bird* conectadas con cada sensor y a su vez interconectadas entre sí en una configuración típica maestro-esclavo por medio de una interfaz específica de *Flock of Birds* llamada FBB[2] (*Fast Bird Bus*). En esta disposición únicamente se permite que exista una unidad *Bird* haciendo las funciones de maestro, mientras que el resto funcionarán en modo esclavo.

[2] Interfaz serie del tipo RS-485/422 half-duplex

Cada unidad *Bird* incorpora el hardware necesario para la comunicación con su sensor y el cálculo de posiciones y orientaciones. A su vez, cada sensor es capaz de proporcionar a su *Bird* entre 20 y 144 medidas por segundo.

Finalmente, el *Bird* maestro se conecta, por un lado, con un transmisor cuyas funciones ya fueron comentadas, y por otra parte, con el computador por medio de una interfaz serie full-duplex del tipo RS-232C o bien una interfaz FBB. El computador puede enviar órdenes y recibir datos individualmente desde cualquier unidad *Bird*, ya que cada una dispone de una dirección que la identifica unívocamente.

Existen cuatro disposiciones diferentes de interconexión entre los componentes expuestos:

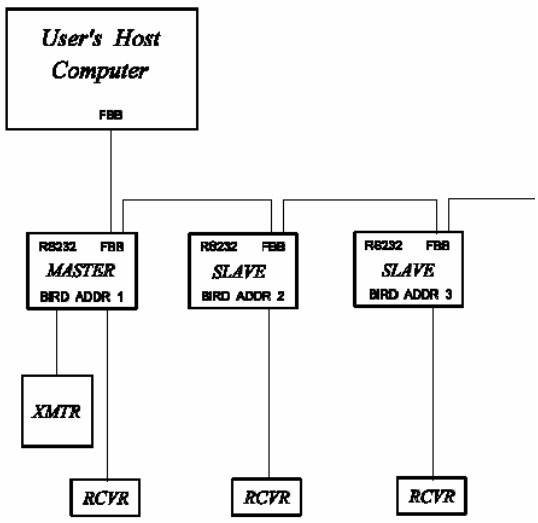


Figura 20: Conexión usando una interfaz FBB

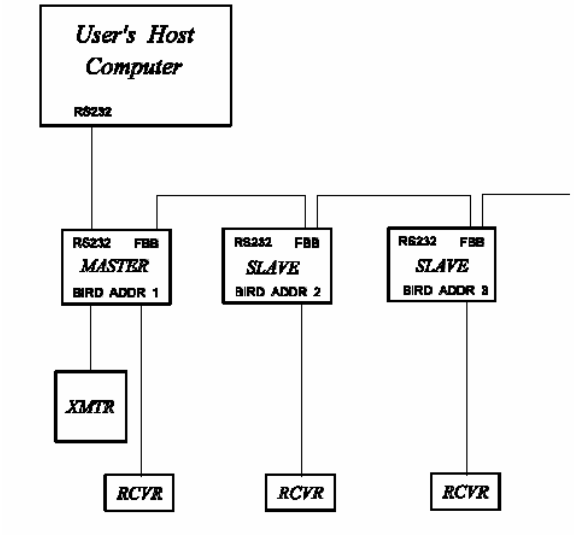


Figura 21: Conexión usando una interfaz RS232

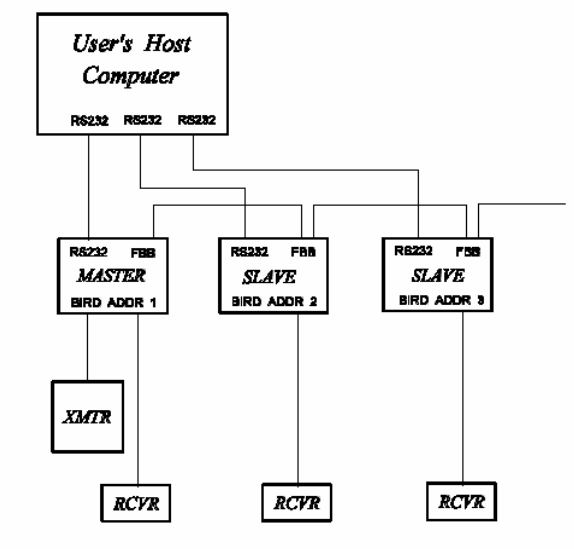


Figura 22: Conexión usando una interfaz RS232 individual por cada Bird

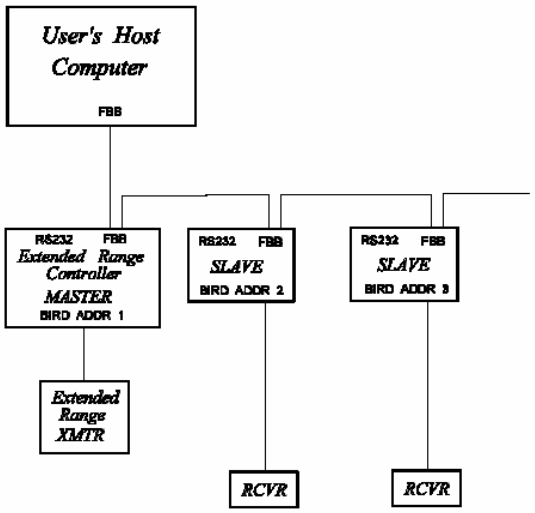


Figura 23: Conexión usando una interfaz FBB con ERC/ERT

Esta última configuración usa dos componentes opcionales que proporcionan un radio de alcance mayor para cada sensor. Se trata del ERC y ERT (*Extended Range Controller* y *Extended Range Transmitter*). Su uso amplía el rango de operación hasta los 3.05 metros y permite la conexión de hasta 125 sensores.

Cualquiera que sea la configuración elegida, será necesario, además, establecer manualmente una serie de parámetros mediante la manipulación de un conjunto de *switches* localizados en el panel trasero de cada unidad *Bird*. Estos parámetros son: velocidad de transmisión, dirección única, y modo de funcionamiento. Existen tres modos de direccionamiento que determinan el intervalo de valores al que puede pertenecer la mencionada dirección única: normal, extendido y superextendido. Según sea el elegido, podremos direccionar hasta un máximo de 125 unidades *Bird*. En cuanto al modo de funcionamiento, este podrá ser *Fly* o *Test* según se desee obtener información de los sensores conectados o bien hacer pruebas de funcionamiento y establecer configuraciones como por ejemplo el modo de direccionamiento deseado.

Un ejemplo de disposición de *switches* es el siguiente:

Conexión de dos unidades *Bird* en modo de direccionamiento normal, velocidad de transmisión de 115200 bps, direcciones 1 para el maestro (conectado al PC) y 2 para el esclavo y modo *Fly* en ambos. Para estos requisitos la configuración de *switches* es "*on on on off off off on off* " para el maestro y "*on on on off off on off off* " para el esclavo,

donde los tres primeros *switches* (de izquierda a derecha) establecen la velocidad, los cuatro siguientes la dirección, y el último el modo de funcionamiento [Fob, 1999]. En el caso de disponer de una única unidad *Bird*, su dirección ha de establecerse obligatoriamente al valor cero.

El esquema electrónico de FOB se resume en el siguiente diagrama de componentes:

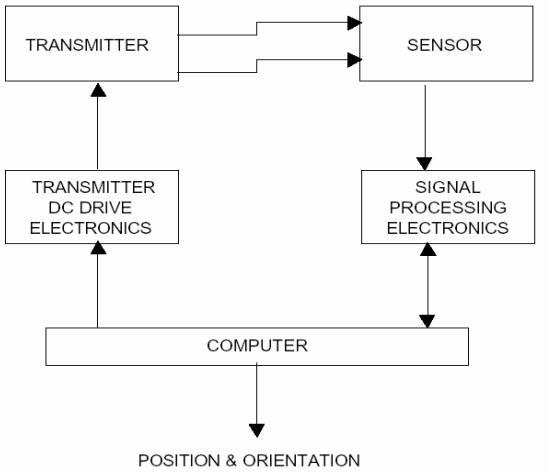


Figura 24: Esquema electrónico de FOB

El transmisor, a través de la electrónica de control, regula los campos magnéticos generados. A su vez, cada sensor capta estos campos y responde con una señal que es transformada por el módulo de procesamiento, utilizada como entrada para el algoritmo de cálculo de posición y orientación y, por último, enviada al computador.

4.2.3. Arquitectura Software

Para la programación de FOB, el fabricante define un completo juego de instrucciones que comprenden desde la configuración del dispositivo hasta la obtención de datos, y que pueden ser transmitidas a través de una interfaz serie de comunicaciones. Sin embargo, no es estrictamente necesario programar el acceso al dispositivo a tan bajo nivel pues, del mismo modo, el fabricante proporciona un módulo intermedio para el acceso desde aplicaciones C/C++ bajo entorno Windows.

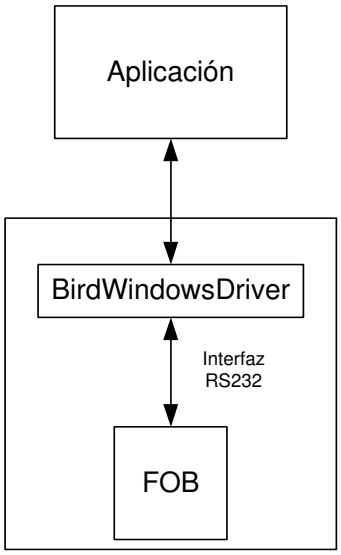


Figura 25: Arquitectura software para el uso de FOB

Este *driver*, disponible de forma gratuita en la página web http://www.ascension-tech.com/support/downloads, ofrece una API a través de la cual se hace transparente al programador el mencionado juego de comandos. De cualquier modo, la manera de usarla se encuentra claramente detallada en un documento proporcionado también por el fabricante y denominado "*Windows Driver User Manual"*.

En estas líneas se relata únicamente algunas de las particularidades y funcionalidades que se consideran más importantes y que por supuesto tienen su ampliación en el manual mencionado.

En concreto, podemos dividir la funcionalidad que ofrece el *driver* en tres partes:

1. Inicialización del *Bird*.
2. Recogida de datos.
3. Finalización.

Durante la fase de inicialización, se configura el dispositivo para la comunicación a través de la correspondiente interfaz, su velocidad de transmisión, el número de unidades *Bird* disponibles y otros parámetros como el formato en el que los datos serán recibidos.

En la fase de recogida de datos, el *driver* ofrece un conjunto de operaciones para obtener las lecturas de los sensores en diversos formatos, es decir, bien como una matriz de orientación, o como un vector de coordenadas de posición, o como un vector de coordenadas de orientación, o bien una combinación de los anteriores formatos. Sea como fuere, el *driver* encapsula las medidas tomadas en una trama que incluye los datos recogidos por todos los sensores.

Por último, tras realizar la lectura de datos, y antes de acabar, es importante finalizar correctamente la comunicación con el dispositivo con el fin de que no haya problemas en posteriores conexiones.

4.2.4. Desarrollo de una librería software genérica para Flock of Birds

Tomando como base el *driver BirdWindowsDriver*, se pretende elaborar un envoltorio software, empleando una metodología de diseño orientada a objetos, que permita a las aplicaciones hacer uso de *Flock of Birds* fácilmente, de modo que la obtención de datos se realice sin la necesidad de preocuparse por aspectos relacionados con la preparación del dispositivo o con la comunicación.

La arquitectura software se amplía, por tanto, de la siguiente manera:

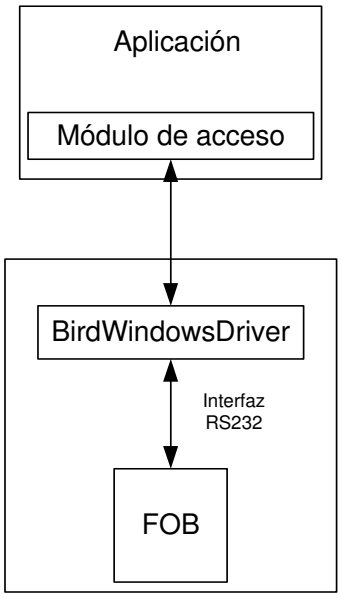


Figura 26: Módulo de acceso a FOB

La responsabilidad de éste módulo, de cara al exterior, será únicamente la de proporcionar, bajo demanda, los datos de cada uno de los sensores conectados. En él, no se asumirán compromisos con el tratamiento que las distintas aplicaciones puedan hacer de estos datos, otorgándole así un carácter flexible y portable entre sistemas distintos.

4.2.4.1. Diseño de bajo nivel

A continuación se muestra el diseño detallado que se ha seguido en el desarrollo del presente módulo.

DIAGRAMA DE CLASES DE DISEÑO:

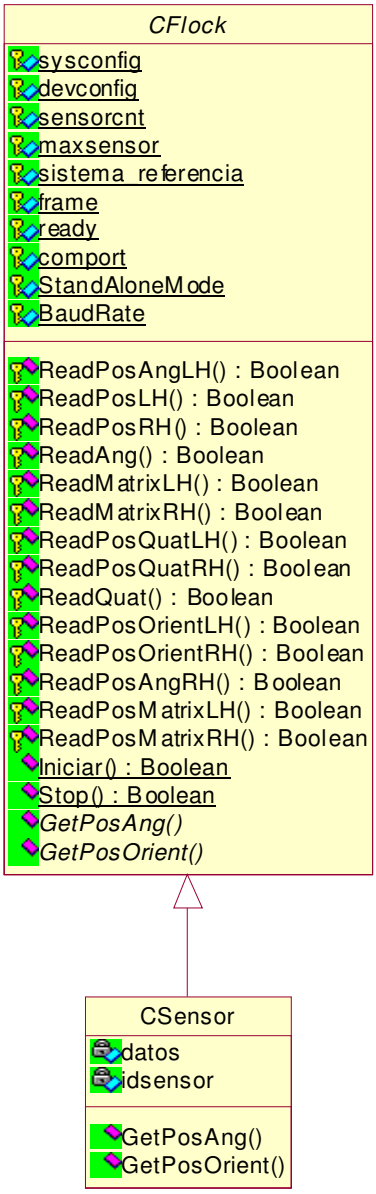


Figura 27: Diagrama de clases de diseño para el módulo de acceso a FOB

DEFINICIÓN DE CLASES, ATRIBUTOS Y MÉTODOS:

Las definiciones que a continuación se presentan son puramente descriptivas y únicamente reflejan las características más relevantes del diagrama de clases.

CLASE	DESCRIPCIÓN
CFlock	Clase responsable de gestionar la configuración del dispositivo *Flock of Birds*, así como de la comunicación con el módulo controlador y la obtención y transformación de datos a sus diferentes formatos de representación.
CSensor	Subclase de CFlock encargada de representar a cada sensor conectado al dispositivo. Define un comportamiento particular para la obtención de datos por parte de las aplicaciones cliente.

CLASE			DESCRIPCIÓN
CFlock	**ATRIBUTOS**	sysconfing	Estructura de datos de información sobre el estado general del dispositivo y de cada una de las unidades *Bird* conectadas.
		devconfig	Configuración de cada unidad *Bird* conectada.
		sensorcnt	Número de sensores en uso en un instante determinado.
		maxsensor	Número de sensores físicamente conectados.
		sistema_referencia	Sistema cartesiano empleado (*Left-Hand o Right-Hand*).

CLASE			DESCRIPCIÓN
		frame	Trama donde se almacenan los datos recibidos de todos los sensores
		ready	Indicador de dispositivo inicializado.
		comport	Identificador del puerto serie de comunicaciones utilizado.
		StandAloneMode	Indicador de que sólo existe una unidad *Bird* conectada en modo aislado[3].
		BaudRate	Velocidad de transmisión de datos del puerto seleccionado.
	MÉTODOS	ReadPosLH (out: Coordenadas, in: sensor) : boolean	Lectura de las coordenadas de posición del sensor indicado en un sistema *Left-Hand*.
		ReadPosRH (out: Coordenadas, in: sensor) : boolean	Lectura de las coordenadas de posición del sensor indicado en un sistema *Right-Hand*.
		ReadPosAngLH (out:Coordenadas, out: Ang, in: sensor) : boolean	Lectura de las coordenadas de posición y de los ángulos de orientación del sensor indicado en un sistema *Left-Hand*.

[3] Consultar "Windows Driver User Manual" [http://www.ascension-tech.com/support/downloads]

CLASE			DESCRIPCIÓN
		ReadPosAngRH (out:Coordenadas,out:Ang, in: sensor) : boolean	Lectura de las coordenadas de posición y de los ángulos de orientación del sensor indicado en un sistema *Right-Hand*.
		ReadPosMatrixLH (out:Coordenadas,out:Matrix, in: sensor) : boolean	Lectura de las coordenadas de posición y de la matriz de orientación del sensor indicado en un sistema *Left-Hand*.
		ReadPosMatrixRH (out:Coordenadas,out:Matrix in: sensor) : boolean	Lectura de las coordenadas de posición y de la matriz de orientación del sensor indicado en un sistema *Right-Hand*.
		ReadPosOrientLH (out:Coordenadas,out:Orient, in: sensor) : boolean	Lectura de las coordenadas de posición y de los vectores de orientación del sensor indicado en un sistema *Left-Hand*.
		ReadPosOrientRH (out:Coordenadas,out:Orient, in: sensor) : boolean	Lectura de las coordenadas de posición y de los vectores de orientación del sensor indicado en un sistema *Right-Hand*.
		ReadPosQuatLH (out:Coordenadas,out:Quat, in: sensor) : boolean	Lectura de las coordenadas de posición y del cuaternio que define la orientación del sensor indicado en un sistema *Left-Hand*.
		ReadPosQuatRH (out:Coordenadas,out:Quat in: sensor) : boolean	Lectura de las coordenadas de posición y del cuaternio que define la orientación del sensor indicado en un sistema *Right-Hand*.

CLASE			DESCRIPCIÓN
		ReadAng(out:Ang, in: sensor) : boolean	Lectura de los ángulos de orientación del sensor indicado según el sistema de referencia de FOB.
		ReadQuat(out:Quat, in: sensor) : boolean	Lectura del cuaternio que define la orientación del sensor indicado según el sistema de referencia de FOB.
		ReadMatrixLH (out:Matrix, in: sensor) : boolean	Lectura de la matriz de orientación del sensor indicado en un sistema *Left-Hand.*
		ReadMatrixRH (out:Matrix, in: sensor) : boolean	Lectura de la matriz de orientación del sensor indicado en un sistema *Right-Hand.*
		Iniciar(in: nsensor, in: ejes, in: comport, in: modo, in: rate) : boolean	Método de clase para la inicialización del dispositivo.
		Stop(): boolean	Método de clase para el correcto apagado del dispositivo.
		GetPosAng(): PosAng	Método abstracto para la obtención de los datos de posición y ángulos de orientación de un sensor.
		GetPosOrient(): PosOrient	Método abstracto para la obtención de los datos de posición y vectores de orientación de un sensor.

CLASE			DESCRIPCIÓN
CSensor	**ATRIBUTOS**	datos	Estructura de datos que almacena la posición y orientación del sensor.
		idsensor	Identificador único del sensor.
	MÉTODOS	GetPosAng(): PosAng	Implementación del método homónimo de la clase CFlock para la obtención de los datos de posición y ángulos de orientación de un sensor.
		GetPosOrient(): PosOrient	Implementación del método homónimo de la clase CFlock para la obtención de los datos de posición y vectores de orientación de un sensor.

OBSERVACIONES:

Clase CFlock:

Ésta es una clase abstracta y por tanto no es posible instanciarla. Como se dijo, es responsable de la configuración y la comunicación directa con el *driver* de *Flock of Birds*. Para ello ha de albergar información acerca de:

- Configuración general del dispositivo.
- Configuración de cada *Bird* conectado.
- Número máximo de sensores que se pueden usar (dependiente del número de unidades *Bird* conectadas).
- Número de sensores usados en cada momento.
- Sistema de coordenadas cartesianas empleado.

- Puerto de comunicaciones serie utilizado.
- Velocidad de transmisión del puerto.
- Estructuras de datos para el almacenamiento de las lecturas.
- Modos de funcionamiento.

Todos estos datos son atributos de clase que serán inicializados a través del método de clase "Iniciar". Algunos de ellos son empleados para lograr las siguientes características:

- Realización de una gestión automática del número de sensores que están siendo usados en cada momento para evitar que se exceda el máximo permitido determinado por el número de unidades *Bird* conectadas.
- Son admitidos dos tipos de sistemas de referencia, *left-hand* y *right-hand*. El módulo se encargará de la adaptación de los datos leídos al sistema correspondiente.
- El *driver* ofrece la posibilidad de realizar una agrupación lógica de unidades *Bird*. El modo de funcionamiento se refiere a si únicamente hay conectado un *Bird* (modo *StandAlone*) o por el contrario hay una agrupación. Estas agrupaciones tienen sentido en configuraciones maestro-esclavo donde los datos de todo el grupo son recogidos a través de la conexión establecida con la unidad que hace las funciones de maestro.

La funcionalidad que ofrece esta clase viene dada por un conjunto de métodos que realizan la lectura de datos en sus diferentes formatos (posición, ángulos de orientación, matriz de orientación, cuaternios), y efectúan las transformaciones necesarias para adaptarlos al sistema de referencia empleado[4]. Estos métodos no deben ser utilizados directamente, sino a través de otros, definidos como métodos abstractos, y que serán implementados en posibles sub-clases de "CFlock".

[4] Véase introducción en páginas 59 y siguientes.

Además de los métodos de instancia, la clase define un método estático que se encarga de inicializar el sistema con los parámetros deseados y que se ha de usar antes de instanciar cualquier objeto de una sub-clase de "CFlock".

```
CFlock::Iniciar(MAX_NUM_SENSOR, LEFT_HAND, COM1, GROUP, 115200);
sensor = new CSensor();
```

Por último, hay que decir que, junto con la definición de la clase, se proporcionan una serie de estructuras de datos que son necesarias para realizar la lectura de datos en todos sus formatos, así como constantes útiles para la inicialización. Todas ellas se detallan en el apartado dedicado a las observaciones sobre la implementación de este diseño.

Clase CSensor:

La clase "CSensor" es una sub-clase de "CFlock" que implementa los métodos abstractos declarados en ella, a través de los cuales los módulos cliente obtendrán las lecturas asociadas a cada sensor en el formato elegido.

Será esta clase la encargada de representar a cada uno de los sensores que se deseen utilizar. A pesar de que el *driver* de acceso proporciona la información de todos los sensores conjuntamente en una única trama, esta clase realiza una clasificación de esa información seleccionando solo los datos referentes a un sensor particular. Esto quiere decir que, por cada sensor conectado, deberá existir una instancia distinta de "CSensor" en la aplicación.

Básicamente, "CSensor" alberga información relativa al número de orden que identifica al sensor, su posición y su orientación.

Es importante comentar que, gracias a que el *driver* ofrece la posibilidad de inicializar cada una de las unidades *Bird* conectadas, en la instanciación de cada objeto de esta clase, y a través de un conjunto de constantes predefinidas también por el *driver* y documentadas en el manual "*Windows Driver User Manual"*, se pueden especificar tanto el formato en el que los datos serán recopilados, como la región del espacio en la que se moverá el sensor con respecto al aparato transmisor. Cada unidad *Bird* utiliza esta información para calcular correctamente las coordenadas de posición y orientación del sensor en función de la región del espacio especificada. Las fronteras de estas regiones alrededor del transmisor están determinadas por el plano horizontal definido por los ejes **y, x**, y los planos verticales definidos por los ejes **y, z** y los ejes **z, x**, repectivamente, según el sistema de referencia de FOB mostrado en la **Figura 16**.

4.2.4.2. Implementación

La implementación del módulo se ha llevado a cabo en el lenguaje C++ por motivos de compatibilidad con el *driver*.

En él, no se soporta la funcionalidad completa que a través del *driver* puede ser ofrecida, sino que se ha limitado a los requisitos impuestos por MAEVIF. Sin embargo, la estructura jerárquica diseñada permite fácilmente la extensibilidad de la funcionalidad a través de implementaciones distintas de la clase "CSensor". De esta forma se ofrece la posibilidad de hacer un tratamiento diferenciado de cada sensor, en cuanto a su comportamiento, en una misma aplicación.

Junto con las clases diseñadas, se ha realizado además la implementación de dos tipos de excepciones para ofrecer a las aplicaciones la posibilidad de efectuar el tratamiento de errores tanto en lo que se refiere a la interfaz de comunicación, como a la instanciación de sensores.

La definición de las estructuras de datos correspondientes a cada uno de los formatos en los que puede ser leída la información proveniente de los sensores, es la siguiente:

```
struct COORDENADAS
{
   double x;
   double y;
   double z;
};
struct ANG
{
   double azimuth;
   double elevation;
   double roll;
};
struct ORIENT
{
   COORDENADAS yaxe;
   COORDENADAS zaxe;
};
```

```
struct POSORIENT
{
   COORDENADAS pos;
   ORIENT orient;
};
struct POSANG
{
   COORDENADAS pos;
   ANG ang;
};
struct QUAT
{
   double q0;
   double q1;
   double q2;
   double q3;
};
typedef  double(*MATRIX)[3];
```

Estas estructuras definen el modo de presentación para las coordenadas tanto de posición como de orientación. En cuanto a éstas últimas, pueden observarse varios formatos equivalentes, como son los ángulos *elevation*, *azimuth* y *roll*, o bien un par de ejes perpendiculares en el espacio, o una matriz de orientación o un cuaternio.

Según las necesidades de la aplicación cliente de este módulo se hará uso de uno u otro formato, pudiendo variar éste entre los distintos sensores utilizados.

4.3. Dispositivo de interacción manual con un entorno virtual

4.3.1. Introducción

El uso combinado de un sensor de FOB y un guante de interacción hace posible la representación gráfica y animada de la mano del usuario en los entornos virtuales. Además de la importancia que esto supone a la hora de interactuar con los objetos presentes en el mundo virtual, ofrece al usuario un punto de referencia visual sobre su propio cuerpo. La decisión sobre la representación en primera persona de cada usuario en el entorno hace que, a diferencia de la realidad física, uno no pueda verse a sí mismo, lo que puede provocar cierta desorientación. El hecho de poder ver y mover una parte propia del cuerpo, como es la mano, ayuda a ubicarse a uno mismo en el espacio.

De entre todos los dispositivos existentes que se pueden emplear para conseguir este efecto, este trabajo se centra en el estudio de uno en particular: CyberGlove, fabricado por Immersion Corporation [http://www.immersion.com].

CyberGlove es un dispositivo basado en un guante al que se le han incorporado cierto número de sensores que miden los ángulos de flexión de cada una de las partes móviles de la mano, logrando así realizar el seguimiento de los movimientos de todos los dedos e incluso de la palma y muñeca.

Si, además, en la base de este guante adherimos un sensor FOB, podremos también transmitir la posición y orientación de la mano del portador.

La información extraída de este conjunto de sensores, ha de servir para posicionar y orientar, en tiempo real, cada uno de los objetos tridimensionales que forman la representación virtual de la mano del usuario.

A pesar de que cada una de las partes de la mano es un objeto independiente, si se establecen entre ellos las oportunas relaciones gobernadas por las leyes de la cinemática, es posible conseguir el efecto de movimiento conjunto deseado.

Además, remitiendonos a las especificaciones dadas por el fabricante, CyberGlove puede ser complementado con otros dispositivos capaces de proporcionar retroalimentación táctil y de fuerza, consiguiendo incrementar notablemente el grado de inmersión dentro de entornos virtuales.

4.3.2. Arquitectura Hardware

Como se ha dicho, CyberGlove es un guante que captura los movimientos de la mano y dedos, a través de los sensores[5] localizados sobre, o cerca de, las articulaciones de la mano y muñeca.

Cada sensor proporciona una medida proporcional a la amplitud, en cada momento, de los arcos de las articulaciones entre huesos adyacentes.

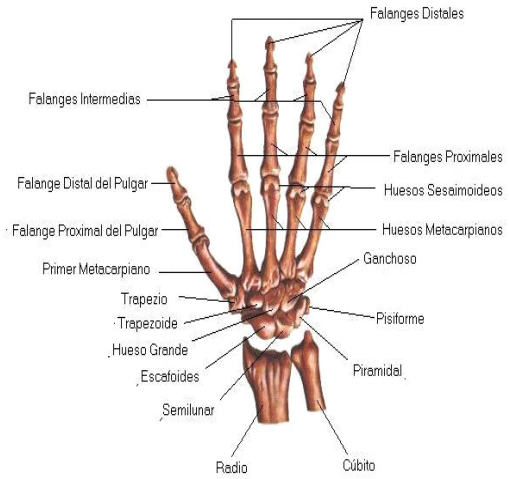


Figura 28: Anatomía de la mano

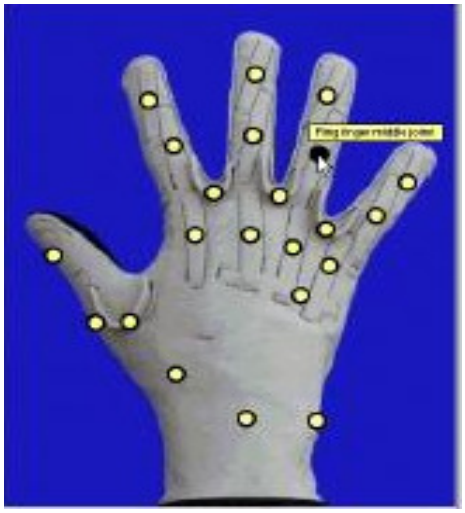


Figura 29: Sensores en CyberGlove

[5] Según el modelo de CyberGlove, el número de sensores es 18 o 22

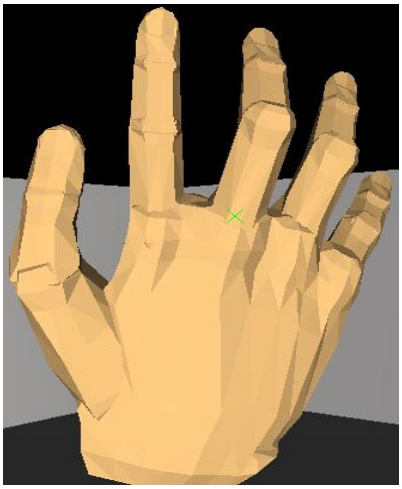


Figura 30: Representación virtual de la mano

Dependiendo del modelo de CyberGlove, puede haber dos o tres sensores en cada uno de los cinco dedos.

En el pulgar, hay dos sensores que miden, respectivamente, la flexión de la articulación que une la falange distal con la proximal, y la que une a ésta con el primer metacarpiano. Para el resto de dedos, existen sensores que miden las articulaciones a ambos extremos de las falanges proximales y, sólo en el caso de contar con un guante de 22 sensores, también las articulaciones de las falanges distales.

Adicionalmente, entre cada par de dedos se localiza un sensor que mide el desplazamiento lateral de los mismos sobre el plano de la palma.

Tanto el dedo pulgar como el meñique disponen, además, de sendos sensores para medir la rotación de estos dedos alrededor de la palma hacia el otro, respectivamente.

Finalmente, hay dos sensores en la muñeca, que miden las rotaciones posibles en esta articulación.

En los modelos con sólo 18 sensores, el valor del ángulo de la falange distal de todos los dedos, excepto en el pulgar, se infiere a partir de la articulación que une las falanges intermedias con las proximales. En la mayoría de los casos este valor resulta ser una buena aproximación.

Como salida, cada sensor produce un voltaje que varía linealmente con la variación del ángulo de curvatura o flexión. Estos valores, tras ser digitalizados en un formato de 8 bits (rango de 0 a 255), son transformados por el software proporcionado por el fabricante para producir una medida equivalente en grados, y para ello es usada una ecuación lineal con dos parámetros ajustables, ganancia (*gain*) y desplazamiento (*offset*). La parametrización de esta ecuación permite particularizar las mediciones a los movimientos físicos de cada mano particular:

$$y = gain * x + offset$$

Ecuación 9

Esta conversión da lugar a una resolución aproximada de 0.5 grados/unidad para una articulación típica cuyo rango de movimiento sea de 90 grados.

CyberGlove consta, básicamente, de dos componentes hardware; por un lado el guante de datos y, por otro, un componente denominado CGIU (*CyberGlove Inteface Unit*) al que se conectará el guante y que será el encargado de amplificar y digitalizar la señal producida por los sensores.

El CGIU, a su vez, se comunica con el computador utilizando una interfaz serie RS232C. A través de esta interfaz de comunicación, el computador envía órdenes y recibe respuestas. A tal efecto, el fabricante especifica un juego de instrucciones que permiten desde la recogida de datos hasta la realización de pruebas de funcionamiento, pasando por la calibración y configuración del hardware.

Por último, CyberGlove dispone de un *switch* situado en la base de la muñeca, cuyo estado de activación puede ser controlado por software, y que puede ser útil para supervisar determinadas condiciones dependientes de la aplicación.

4.3.3. Arquitectura Software

Junto con el hardware descrito, el fabricante proporciona un conjunto de aplicaciones y librerías de programación que pueden ser utilizadas como *driver* de acceso al dispositivo. Este *driver* propone una arquitectura cliente-servidor donde el papel del cliente es desempeñado por la aplicación de usuario, y el de servidor es realizado por un programa, conocido como *Device Manager*, cuya misión es centralizar el acceso al dispositivo y gestionar la comunicación.

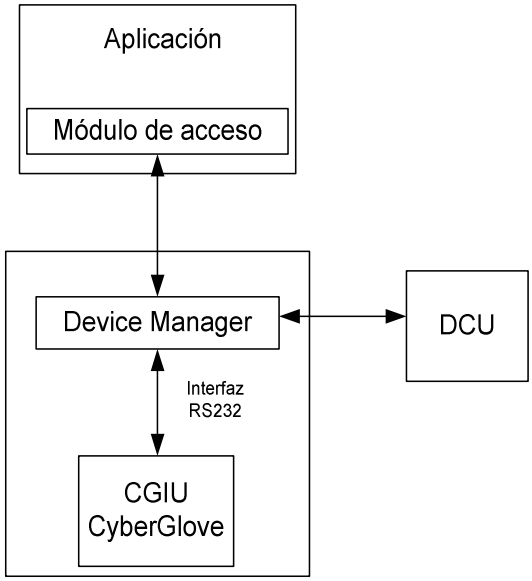


Figura 31 : Arquitectura software para el uso de CyberGlove

A través del *Device Manager* las aplicaciones disponen de un punto de acceso para la interacción con el hardware del dispositivo, pero para ello es necesario realizar algunas tareas de configuración.

Para poder comunicarse con las aplicaciones, el *Device Manager* ha de registrar información sobre los guantes conectados, la conexión por medio de la interfaz serie, la calibración de los sensores, etc. El fabricante permite al usuario establecer estos parámetros y realizar pruebas de funcionamiento por medio de una aplicación llamada DCU (*Device Configuration Utility*). Una vez registrados estos parámetros, las aplicaciones de usuario podrán solicitar al *Device Manager* que establezca la comunicación con los dispositivos conectados y que recopile los datos proporcionados por los mismos.

4.3.4. Desarrollo de una librería genérica para Immersion CyberGlove

El análisis y diseño de esta librería sigue la filosofía adoptada en el desarrollo hecho anteriormente para FOB, es decir, abstraer a las aplicaciones cliente de los detalles particulares de la conexión, ofreciendo una "caja negra" que proporciona los datos de los sensores correspondientes a cada una de las partes de la mano.

4.3.4.1. Diseño de bajo nivel

DIAGRAMA DE CLASES DE DISEÑO:

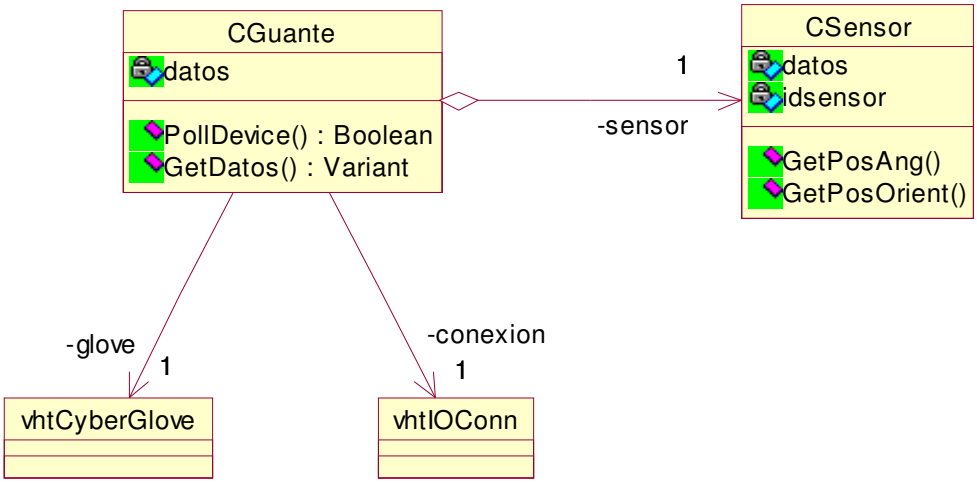


Figura 32: Diagrama de clases de diseño para el módulo de acceso a CyberGlove

DEFINICIÓN DE CLASES, ATRIBUTOS Y MÉTODOS:

Las definiciones que a continuación se presentan son puramente descriptivas y únicamente reflejan las características más relevantes del diagrama de clases.

CLASE	DESCRIPCIÓN
CGuante	Clase que representa al dispositivo CyberGlove en la aplicación y a través de la que se obtienen los datos de sus sensores.
vhtCyberGlove	Clase proporcionada por el fabricante como parte del módulo controlador de acceso al dispositivo. Contiene los métodos necesarios para realizar las lecturas de cada sensor.
vhtIOConn	Clase proporcionada por el fabricante como parte del módulo controlador de acceso al dispositivo. Representa la configuración de la conexión con el mismo.
CSensor	Esta clase, perteneciente al módulo de acceso para FOB descrito anteriormente, representa al sensor adherido a la muñeca que complementa la funcionalidad del guante.

CLASE			DESCRIPCIÓN
CGuante	**ATRIBUTOS**	datos	Estructura de datos que almacena las lecturas recogidas de cada uno de los sensores.
	MÉTODOS	PollDevice() : boolean	Método que realiza el sondeo del dispositivo actualizando el atributo "datos" con nuevas medidas.
		GetDatos() : datos	Método para la obtención de los datos medidos.

OBSERVACIONES:

El módulo está constituido por una clase principal "CGuante". Esta clase hace uso de las librerías proporcionadas por el fabricante de CyberGlove y la librería propia diseñada para FOB, para recopilar y encapsular la información recogida por todos los sensores distribuidos por mano y muñeca.

La recopilación de datos se realizará mediante una solicitud de sondeo invocando a la operación "PollDevice", mientras que las aplicaciones cliente podrán obtener la estructura de datos que contiene la lectura de todos los sensores por medio de la operación "GetDatos", para cuyo éxito es indispensable haber realizado previamente el sondeo del dispositivo. Aunque este proceso de sondeo y recogida de datos podría haberse fusionado en una sola operación, se ha optado por dividirlo en dos en previsión de futuras modificaciones. Por ejemplo, podría darse el caso en el que se quisieran hacer sondeos selectivos de un grupo de sensores en función de ciertas condiciones determinadas por la aplicación, en tal caso existirían distintos métodos de sondeo y sería la aplicación cliente la que decidiría cual aplicar antes de realizar la recogida de datos.

4.3.4.2. Implementación

Por imperativo del *driver* proporcionado por el fabricante, la implementación de este módulo ha sido realizada en el lenguaje C++.

Una de las decisiones tomadas durante la fase de implementación se refiere a las responsabilidades que este módulo asume con respecto al sensor FOB que, necesariamente, se le asocia para hacer el oportuno seguimiento de la posición de la mano en el espacio.

Se entiende que el sensor adherido a la muñeca es un dispositivo independiente de CyberGlove, que forma parte del conjunto de dispositivos que las aplicaciones pueden manejar, y como tal, la clase "CGuante" no es responsable de su creación. De esta manera, se deja a la aplicación que hace uso de este módulo, la responsabilidad de seleccionar el sensor FOB que permanecerá ligado al guante durante la ejecución.

Por otra parte, en esta implementación, se ha elegido el siguiente formato para la encapsulación de los datos recogidos de cada sensor. Este formato ha de ser conocido por las entidades cliente del módulo para su correcta interpretación, por eso, como trabajo futuro, se propone la transformación de esta estructura de datos en una clase con métodos capaces de interpretarlos y de ofrecerlos a los clientes liberándolos así de este conocimiento.

```
struct DATOS
{
   double * dedos[20];
   double * led;
   void * muñeca;
};
```

Como se puede ver, en el vector "dedos" se almacenan las mediciones correspondientes a un total de veinte de los veintidós sensores de CyberGlove, ignorando los sensores de la muñeca, cuya posición y orientación se tomará del campo "muñeca" que, en este caso, será rellenado con las medidas tomadas por un sensor FOB. Además, el campo "led" de esta estructura recoge la información sobre el estado del *switch* disponible en CyberGlove.

4.4. Dispositivos de navegación: Joysticks

4.4.1. Introducción

Este tipo de dispositivos es el más conocido por su difusión dentro del mercado de videojuegos. Precisamente, el avance en el desarrollo de juegos para ordenador ha hecho que la tendencia se dirija hacia el empleo de técnicas de realidad virtual, por lo que, de manera recíproca, también en el área de investigación sobre realidad virtual han sido adoptados algunos usos provenientes del desarrollo de videojuegos, entre ellos el de las palancas de mando o joysticks.

Debido también en parte al éxito de la industria de los juegos, existen multitud de variedades de joysticks, desde volantes para simulación de la conducción hasta palancas con más o menos botones, palancas deslizadoras, con control del punto de vista, con posibilidad de movimiento en dos ejes o más, etc.

Para el propósito del proyecto MAEVIF, las características básicas que se buscan en un joystick son:

- Control del punto de vista, necesario para orientar la dirección de la mirada del avatar simulando así la rotación de la cabeza.
- Desplazamiento de la palanca al menos en dos ejes para permitir el avance y retroceso del avatar y el desplazamiento lateral
- Control de aceleración, para regular la velocidad del movimiento.
- Rotación de la palanca para orientar el avatar en la dirección deseada.
- Al menos tres botones que nos permitan incrementar los grados de libertad del movimiento, por ejemplo, en la orientación de la cabeza, y para proporcionar al usuario alguna funcionalidad adicional.

Además de estas características, opcionalmente, la tecnología actual de algunos tipos de joysticks permite simular efectos de retroalimentación de fuerza que pueden ser personalizados según las necesidades de la aplicación. Por ejemplo, a través de pequeñas vibraciones rítmicas en la palanca, se puede lograr el efecto del movimiento al andar, se pueden conseguir también efectos de choque contra paredes u objetos, efectos de inercia, etc.

Dentro de las limitaciones inherentes a la morfología del dispositivo en cuanto a capacidad de inmersión en un entorno virtual, el joystick tiene ventajas muy significativas que son notablemente apreciadas. Por ejemplo, la fluidez y estabilidad en los movimientos no se puede comparar con la lograda a través de sensores; el uso del joystick elimina temblores e interferencias y su manejo es tan intuitivo como pueda ser el de un ratón.

En el caso concreto del proyecto MAEVIF, se ha elegido el modelo de joystick *Logitec Force 3D* aunque esto no es relevante, puesto que, como se verá, la arquitectura software empleada para el control de este tipo de dispositivos permite la abstracción total de la aplicación sobre el modelo específico. Tan sólo con que el joystick empleado cumpla con los requisitos funcionales impuestos por MAEVIF, bastará para que pueda ser usado sin necesidad de adaptaciones especiales en la implementación.

4.4.2. Arquitectura Hardware

Típicamente, la configuración hardware de esta clase de dispositivos consta de una palanca de control conectada al ordenador a través de una interfaz serie, ya sea a través del puerto de comunicaciones serie, un puerto USB o un puerto especial para dispositivos de juego. Adicionalmente, puede que se precise conectar el joystick a una fuente de corriente para aquellos que dispongan de la característica de retroalimentación de fuerza o *force-feedback*.

Otra propiedad del hardware que de alguna manera clasifica la multitud de tipos de joystick, es la de medir la cantidad de movimiento aplicado sobre la palanca, es decir, proporcionar la información de manera analógica como un valor numérico en un rango determinado, o de forma digital informando, únicamente, de si se ha desplazado a izquierda, derecha, arriba o abajo. Con un joystick analógico siempre será posible simular mediante software el funcionamiento digital, pero no al revés.

El modelo *Logitec Force 3D* dispone de dos ejes de movimiento, un control del punto de vista, una palanca deslizadora para el control de aceleración, seis botones y rotación a izquierda y derecha de la palanca de mando. Cuenta, además, con un motor de *force-feedback* y utiliza una interfaz USB para la conexión con el ordenador.

Figura 33: Logitec Force 3D

4.4.3. Arquitectura Software

El software proporcionado por el fabricante para este dispositivo consta de un *driver* que es instalado en el sistema operativo como en cualquier otro periférico de uso común. A diferencia de los otros dispositivos expuestos, en esta ocasión el fabricante no proporciona una API específica de acceso al *driver*, puesto que existen interfaces de programación genéricas de libre distribución que son perfectamente compatibles. De esta forma podemos programar el acceso al dispositivo a través de la biblioteca de clases MFC de Microsoft, por medio de DirectInput, también de Microsoft, o utilizando otras librerías con el mismo propósito.

En este caso se ha utilizado la API DirectInput, que paso a describir.

DirectInput es el componente de la interfaz de programación DirectX para el procesado de información proveniente de dispositivos de entrada como el teclado, ratón, joystick u otros dispositivos de juegos, así como dispositivos con *force-feedback*.

DirectInput proporciona servicio para dispositivos no soportados directamente por la API Win32 de Microsoft; además ofrece un acceso más rápido mediante la comunicación directa con los controladores hardware, en contraposición al mecanismo de intercambio de mensajes con el sistema operativo Windows típico del modelo de aplicación con MFC.

Al igual que en el resto de la librería DirectX, el modelo de programación está basado en componentes COM (*Component Object Model*). COM es una especificación binaria para objetos de forma que puedan ser utilizados desde distintos lenguajes.

Los componentes COM son cajas negras implementadas generalmente como DLLs que pueden ser usadas por las aplicaciones para realizar una o más tareas.

El funcionamiento del paradigma COM se basa en la publicación de métodos, por parte de los objetos, que son agrupados en interfaces a través de las cuales las aplicaciones invocan sus servicios.

El esquema de la arquitectura usada en MAEVIF para el control del joystick es el siguiente:

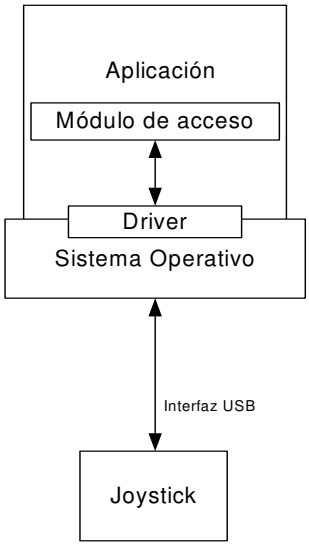


Figura 34: Arquitectura Software para el uso de un Joystick

4.4.4. Diseño de una librería software genérica

El propósito de esta librería es hacer transparente el uso de DirectInput para la comunicación con el dispositivo. La configuración, el sondeo del dispositivo y la recogida de datos serán sus responsabilidades, ofreciendo al exterior una interfaz sencilla ajena a estos procesos.

4.4.4.1. Diseño de bajo nivel

DIAGRAMA DE CLASES DE DISEÑO:

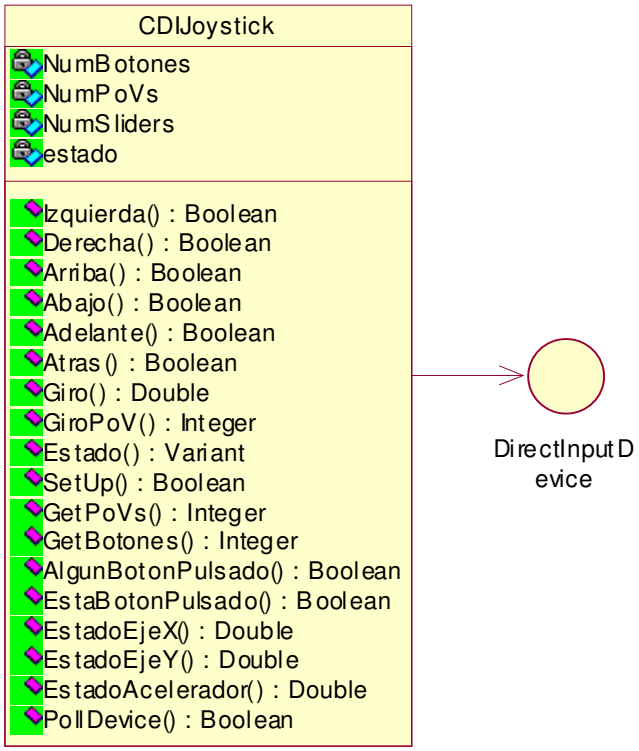


Figura 35: Diagrama de clases de diseño para el módulo de acceso a un joystick

DEFINICIÓN DE CLASES, ATRIBUTOS Y MÉTODOS:

Las definiciones que a continuación se presentan son puramente descriptivas y únicamente reflejan las características más relevantes del diagrama de clases.

CLASE	DESCRIPCIÓN
CDIJoystick	Clase que encapsula el acceso al *driver* del joystick a través de la librería "DirectInput".
DirectInpuDevice	Interfaz de acceso al componente COM a través del que se establece la comunicación con el dispositivo.

CLASE			DESCRIPCIÓN
CDIJoystick	**ATRIBUTOS**	NumBotones	Número de botones disponibles en el joystick.
		NumPoVs	Número de controles del punto de vista disponibles en el joystick.
		NumSliders	Número de palancas deslizantes presentes en el joystick.
		Estado	Estructura de datos que almacena información sobre el estado de todos los componentes del joystick.
	MÉTODOS	Izquierda() : boolean	Informa sobre si el joystick está desplazado hacia la izquierda.
		Derecha() : bolean	Informa sobre si el joystick está desplazado hacia la derecha.
		Arriba() : bolean	Informa sobre si el joystick está desplazado hacia arriba.
		Abajo() : bolean	Informa sobre si el joystick está desplazado hacia abajo.
		Adelante() : bolean	Informa sobre si el joystick está desplazado hacia delante.
		Atras() : bolean	Informa sobre si el joystick está desplazado hacia atrás.
		GiroPoV() : Integer	Informa sobre la situación del control del punto de vista (norte, sur, este u oeste).

CLASE			DESCRIPCIÓN
		GetBotones(): Integer	Informa del número de botones
		GetPoVs(): Integer	Informa del número de controles del punto de vista.
		AlgunBotonPulsado(): boolean	Informa sobre si ha sido pulsado cualquier botón.
		EstaBotonPulsado(in: boton) : boolean	Informa sobre si ha sido pulsado un boton determinado
		Estado(): datos	Proporciona una estructura de datos con el estado completo del dispositivo.
		EstadoEjeX(): double	Informa sobre la cantidad de desplazamiento lateral del joystick.
		EstadoEjeY(): double	Informa sobre la cantidad de desplazamiento vertical del joystick.
		EstadoAcelearador(): double	Informa sobre la posición del control de aceleración.
		Giro(): double	Informa sobre la cantidad de giro aplicado sobre la palanca.
		PollDevice(): boolean	Realiza un sondeo del dispositivo para actualizar el atributo estado.
		SetUp(): boolean	Inicializa el dispositivo.

OBSERVACIONES:

El módulo diseñado para la interacción con el joystick consta de una única clase "CDIJoystick" que encapsula y hace transparente el uso de DirectInput, ofreciendo una interfaz pública de métodos para informar sobre el estado completo del dispositivo.

Esta clase será responsable de crear e inicializar los componentes DirectInput, configurar todas las características del joystick en particular y ofrecer la información acerca de su estado, siempre bajo demanda.

El diseño contempla la posibilidad de utilizar el joystick en modo digital o analógico, si es que esto es posible, y deja abierta la posibilidad de incorporar a la implementación efectos de *force-feedback* sin más que añadir la implementación de sus características, puesto que la configuración realizada por este módulo deja al dispositivo preparado para ofrecerlos.

4.4.4.2. Implementación

Como el resto de dispositivos, y en este caso casi obligado por la decisión de utilizar DirectInput, el módulo ha sido implementado en el lenguaje C++.

Como notas a tener en cuenta, cabe decir que, en la implementación del comportamiento en modo analógico, se barajan cuatro conceptos:

1. La velocidad máxima (A) de los desplazamientos y giros, es decir el número de unidades máximo, en la escala en la que esté diseñado el entorno virtual, que se desplazará o girará el avatar cada vez que se actualice la situación del escenario. Este valor ha sido fijado para oscilar entre 0 y 20 en función de la posición de la palanca de control de aceleración, es decir, el desplazamiento de dicha palanca incrementará o decrementará dicho valor.

2. La cantidad de movimiento o giro (P), es decir, dependiendo de si la palanca es llevada al límite o no de su recorrido, tanto en desplazamiento como en rotación, el valor obtenido a través de la API de DirectInput varía entre -10000 y 10000, y es transformado por el módulo de acceso a un valor entre -1 y 1 para ser utilizado como factor de ponderación.

3. La velocidad lineal en los desplazamientos. Es la velocidad efectiva que se aplicará al movimiento tanto frontal como lateral. Se calcula ponderando la velocidad máxima con la cantidad de movimiento:

$$V = A * P$$

Ecuación 10

Donde:

A es la velocidad máxima.

P es la cantidad de movimiento.

V es la velocidad efectiva medida en las unidades de la escala del entorno.

4. La velocidad angular en los giros, es decir, el ángulo que recorrerá el avatar en su giro cada vez que se actualiza la situación del entorno virtual. Este valor está parametrizado por el radio con centro en la posición del avatar, sobre la que se realiza el giro, de manera que resulta inversamente proporcional al mismo:

$$W = \frac{V}{r} = \frac{A * P}{r}(rad) \Leftrightarrow \frac{A * P * 180}{r * \pi}(grados)$$

Ecuación 11

Donde:

W es la velocidad angular.

r es el radio de giro.

En la implementación se ha fijado el radio de giro a un valor experimental de 300 unidades, con el que se ha apreciado una velocidad de giro equivalente a la velocidad de desplazamiento.

Ilustrando estas ecuaciones con un ejemplo vemos que, si la palanca de control de aceleración está al 75% de su recorrido y la palanca de mando se encuentra desplazada hacia arriba al 50%, el resultado es que el avatar se desplazará hacia delante una distancia igual a:

$$(20*0.75)*0.50=7.5 \quad unidades$$

Del mismo modo, si la cantidad de giro aplicado sobre la palanca es del 50% en el sentido de las agujas del reloj, el avatar girará sobre sí mismo un número de grados igual a:

$$\frac{(20*0.75)*0.50*180}{300*\pi}=1.43 \quad grados$$

4.5. Desarrollo de un módulo software para la integración de dispositivos heterogéneos en MAEVIF

Para el desarrollo de este módulo se va a seguir el método de desarrollo orientado a objetos propuesto por Craig Larman [Larman, 1999] [Ferré, 2002].

MAEVIF, para cada avatar que representa a un usuario dentro de un entorno virtual, considera tres variables: la posición y orientación de su cuerpo, en función de la que se determinará la dirección de los movimientos; la orientación de su punto de vista y la posición y orientación de su mano derecha. Cada una de estas variables es controlada por una combinación de cuatro de los dispositivos mencionados anteriormente, es decir, teclado, joystick, *tracker* y guante.

La misión de este módulo es conseguir integrar, en una interfaz común, la diversidad en el control de estos dispositivos. De esta forma, su posterior incorporación dentro de MAEVIF se podrá realizar de manera sencilla y transparente.

Como es lógico, no todas las combinaciones de esos cuatro dispositivos son útiles. Algunas de ellas carecen de sentido; por ejemplo, si un usuario lleva puesto un HMD y su posición y orientación se obtienen a través del sensor o *tracker* adherido a su cabeza, es imposible que pueda manejar un teclado, ya que, en primer lugar no puede verlo y en segundo, el usuario normalmente estará moviéndose a lo largo de un recinto, por lo que no puede llevar consigo el teclado. Como ésta, existen más limitaciones que restringen ciertos usos.

En concreto las combinaciones posibles son:

1. Si el usuario lleva HMD con sensor de posicionamiento, la posición y la orientación del punto de vista se obtendrán de él, mientras que el control de la mano virtual únicamente se hará a través del guante. La razón de esta combinación es la expuesta en el ejemplo anterior.

2. Si el usuario controla el desplazamiento de su avatar a través del teclado, su punto de vista también podrá ser controlado por el teclado, o bien mediante el uso del HMD y sensor. Igualmente se podrá elegir entre manejar la mano con el teclado o con el guante. Se asume que al hacer uso del teclado, el usuario deberá estar sentado frente a él, por lo que el efecto del sensor que pueda incorporar el HMD, si éste es el caso, se restringe a la orientación de su punto de vista.

3. Si el usuario controla el desplazamiento de su avatar a través del joystick, estamos ante el mismo caso que el anterior, reemplazando el teclado por el joystick tanto en el control del movimiento como en el del punto de vista si es el caso.

El siguiente grafo ilustra cómo la elección de un dispositivo para una tarea limita las opciones en las demás:

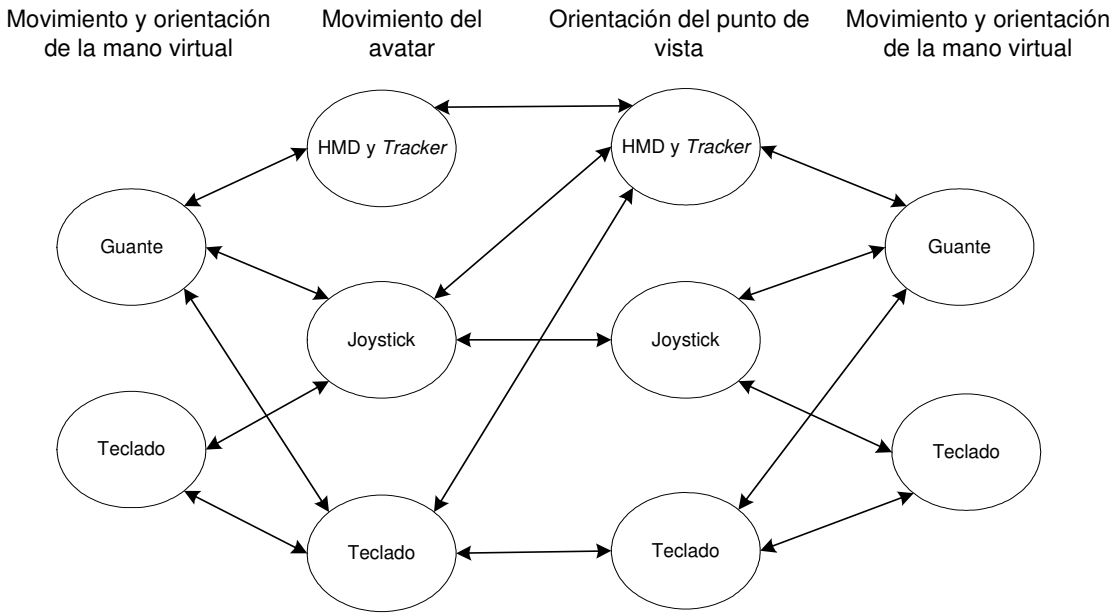


Figura 36: Grafo de combinaciones posibles en la selección de dispositivos

Hay que destacar que, al margen de la combinación de dispositivos elegida, el sistema ha de mantener la coherencia entre los movimientos del cuerpo y cabeza del avatar, por un lado, y los de la mano por otro, consiguiendo que la mano virtual esté correctamente posicionada con respecto al cuerpo en cada momento, incluyendo giros y cambios de altura.

4.5.1. Análisis

De la especificación dada en el anterior apartado de introducción así como en los capítulos previos podemos extraer los siguientes requisitos:

REQ[1]: El usuario podrá hacer uso, a su elección, de tres tipos de dispositivos para desplazarse y orientarse en el entorno virtual. Estos dispositivos son: teclado, *tracker* y joystick.

REQ[2]: El usuario podrá hacer uso, a su elección, de dos tipos de dispositivos para controlar la representación de su mano en el entorno virtual. Estos dispositivos son: teclado y guante de datos.

REQ[3]: La combinación de dispositivos elegida por el usuario estará sujeta a las restricciones reflejadas en la **Figura 36**.

REQ[4]: El sistema será responsable de consultar periódicamente el estado de cada dispositivo seleccionado o, en su caso, de "escuchar" los eventos procedentes de éstos y/o del sistema multi-agente.

REQ[5]: La interacción mediante dispositivos sólo afectará al control del avatar que representa al usuario o a la representación de su mano en el entorno.

REQ[6]: Independientemente de los dispositivos escogidos, el sistema mantendrá la coherencia entre la posición del avatar que representa al usuario, de su punto de vista y la de su mano virtual.

REQ[7]: El usuario podrá interrogar al sistema sobre los objetos que ve en el entorno virtual sin más que señalarlos con la mano.

REQ[8]: El sistema debe reflejar la situación del entorno virtual de acuerdo a las acciones efectuadas por el usuario con los dispositivos.

La información contenida en esta colección de requisitos nos permite definir el conjunto de casos de uso que describen la funcionalidad requerida. Además, dado que este módulo ha de ser una parte integrante de MAEVIF, su análisis funcional se hará no sólo desde el punto de vista de la interacción con usuarios externos, sino también con otros subsistemas pertenecientes a MAEVIF.

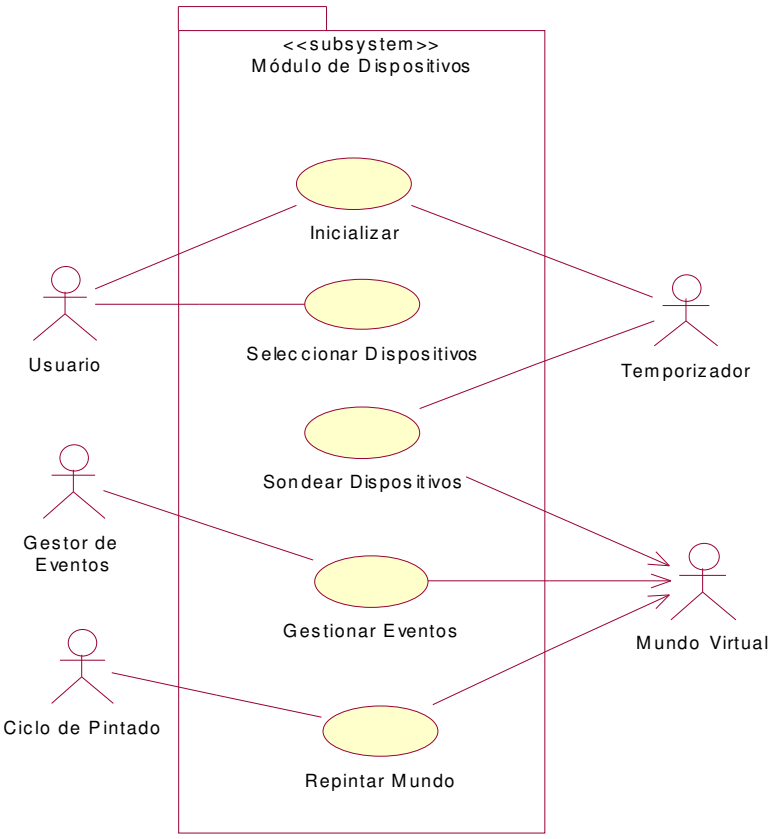


Figura 37: Diagrama de casos de uso

El diagrama de casos de uso representa, por medio de actores, aquellas entidades que quedan fuera de los límites de este módulo.

Como muestra la siguiente tabla, existen cinco actores distintos que mantienen una relación de interacción con el módulo de dispositivos; cuatro de ellos representan a otros subsistemas o procesos dentro de MAEVIF, y el último representa al usuario que se conecta al sistema.

Actor	Descripción
Usuario	Representa a la persona que se conecta al sistema e interacciona con él por medio de un conjunto de dispositivos de realidad virtual.
Temporizador	Representa a un proceso dentro del sistema que se activa periódicamente desencadenando la realización de ciertas tareas, entre las que está el sondeo de dispositivos.
Gestor de Eventos	Representa a la capa de software encargada de capturar eventos externos procedentes de dispositivos, como el teclado o el ratón, cuyo funcionamiento está orientado a eventos.
Ciclo de Pintado	Representa al proceso encargado de solicitar periódicamente el repintado de los elementos de la escena gráfica. Su creación tiene lugar durante la fase de conexión e inicio de sesión de un usuario en el sistema.
Mundo Virtual	Es la representación del módulo que contiene la información geométrica de los elementos contenidos en el entorno virtual, y será afectado por algunos de los procesos que se ejecutan en el módulo de dispositivos.

La descripción esencial y de alto nivel de estos casos de uso nos proporciona una primera aproximación en el análisis del problema:

Caso de uso: *Inicializar*
Actores: *Usuario (iniciador), Temporizador*
Tipo: *primario y esencial*
Descripción: *Cuando un usuario inicia una nueva sesión en el sistema, éste realiza la inicialización del proceso temporizador que periódicamente ordenará el sondeo de dispositivos.*

Caso de uso: *Seleccionar Dispostivos*
Actores: *Usuario (iniciador)*
Tipo: *primario y esencial*
Descripción: *Un usuario que ha iniciado una sesión en el sistema selecciona la combinación de dispositivos que va a utilizar para moverse por el entorno virtual, controlar su punto de vista y mover su mano.*

Caso de uso: *Sondear Dispositivos*
Actores: *Temporizador (iniciador), Mundo Virtual*
Tipo: *primario y esencial*
Descripción: *El actor Temporizador solicita que se realice el sondeo de los dispositivos seleccionados previamente. Como consecuencia de ello se produce la actualización de las posiciones y orientaciones de algunos de los elementos contenidos en el Mundo Virtual, como el avatar que representa al usuario o su mano virtual.*

Caso de uso: *Gestionar Eventos*
Actores: *Gestor de Eventos (iniciador), Mundo Virtual*
Tipo: *primario y esencial*
Descripción: *El Gestor de Eventos, tras capturar un evento procedente de algún dispositivo externo o del sistema multi-agente, solicita que sea procesado provocando cambios en el Mundo Virtual.*

Caso de uso: *Repintar Mundo*
Actores: *Ciclo de Pintado (iniciador), Mundo Virtual*
Tipo: *primario y esencial*
Descripción: *El actor Ciclo de Pintado solicita periódicamente que se realice un repintado de la escena gráfica, con lo que se deberá notificar al Mundo Virtual para que repinte cada uno de sus objetos virtuales.*

Además de describir en más detalle la funcionalidad, los casos de uso sacan a relucir algunos conceptos que pueden ayudar a entender el problema. Con ellos se puede construir un modelo conceptual que en absoluto pretende reflejar la solución adoptada, sino tan solo sintetizar y desmenuzar la complejidad del problema.

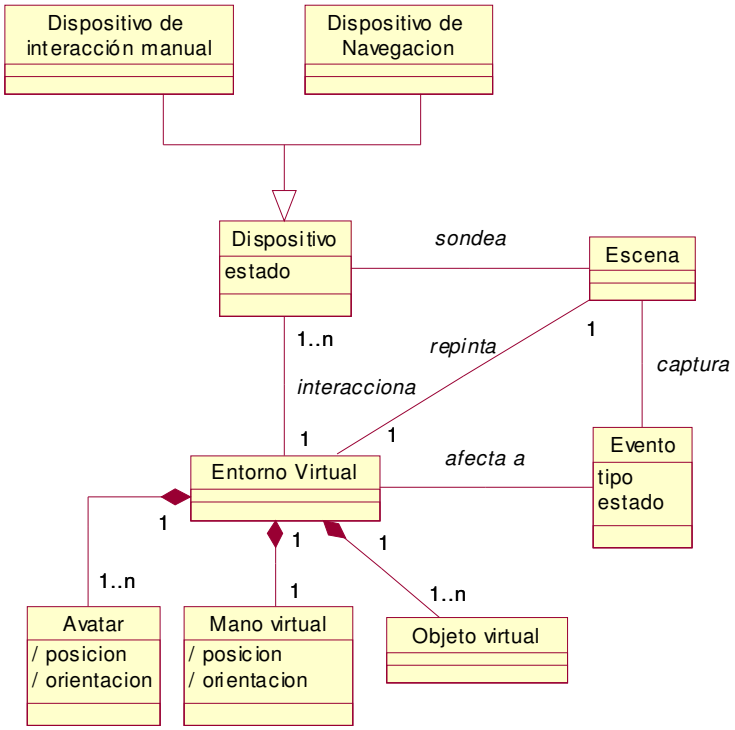


Figura 38: Modelo conceptual

Este modelo manifiesta la relación que existe entre los distintos tipos de dispositivos y el entorno virtual. Un entorno virtual puede estar controlado por varios dispositivos distintos (de navegación y de interacción manual) pero todos ellos interaccionan con un mismo entorno. A su vez, dicho entorno está formado por los objetos presentes en él, por los avatares que representan a los usuarios y, adicionalmente, por la representación virtual de la mano del usuario.

La entidad Escena representa el marco de visualización del entorno virtual, así como una ventana de entrada desde el exterior del sistema. Es la encargada de realizar el sondeo de los dispositivos y también de capturar los eventos del exterior que afectan al entorno virtual.

Se puede ver cómo a través de las relaciones “interacciona” y “afecta a”, el entorno virtual es capaz de obtener el valor de los atributos posición y orientación de sus componentes a partir del estado de cada dispositivo y del tipo y estado de los eventos capturados.

Como último paso de este análisis se presenta un diagrama que puede ayudar a ubicar el módulo a desarrollar en su contexto adecuado dentro de la arquitectura de MAEVIF:

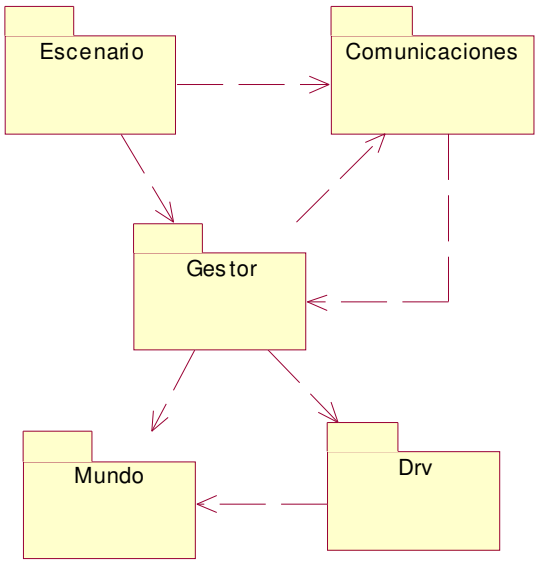


Figura 39: Descomposición en paquetes

En realidad, el módulo que aquí se discute estaría formado únicamente por el paquete *Drv* y en parte por los paquetes *Gestor* y *Escenario*. Sin embargo, para poder entenderlo en su totalidad, es necesario situarlo en el conjunto del sistema completo, o por lo menos de la parte incluida dentro del entorno gráfico.

El punto de entrada para las interacciones que el usuario invoca sobre el sistema es el paquete *Escenario*. Este podría describirse como la vista e interfaz del entorno virtual con el exterior. Es responsable de ordenar su visualización y de la captura de eventos procedentes tanto del usuario como del sistema multi-agente.

Todos los eventos que llegan a *Escenario* (pulsaciones de teclas, eventos de ratón, interrupciones de temporización, eventos producidos por agentes, etc.) son automáticamente transmitidos al paquete *Gestor*. Este paquete centraliza y unifica el acceso a los componentes del mundo virtual (paquete *Mundo*) desde otras partes del sistema. Además procesa los eventos e inicia la actualización del estado del mundo mediante el sondeo de los dispositivos gestionados en el paquete *Drv*.

El paquete *Drv* es el que más nos interesa. Es responsable del control de los distintos dispositivos, de su inicialización y sondeo y, además, ha de ofrecer al *Gestor* una interfaz transparente e independiente de cada dispositivo.

Por último, comentamos brevemente el paquete de *Comunicaciones*. Su responsabilidad es transmitir al sistema multi-agente los cambios producidos en el estado del entorno virtual. Además, este paquete se encarga del aspecto multiusuario, coordinando las distintas vistas con que los usuarios perciben el entorno virtual según sus posiciones dentro de él.

4.5.2. Diseño de alto nivel

El conocimiento sobre el problema, adquirido al definir sus límites y su modelo conceptual, permite desarrollar los casos de uso en su formato expandido, lo que no significa aún ningún compromiso con la solución:

<table>
<tr><td colspan="2">Caso de uso: Inicializar</td></tr>
<tr><td colspan="2">Actores: Usuario (iniciador), Temporizador</td></tr>
<tr><td colspan="2">Propósito: Preparar el sistema para realizar el sondeo periódico de los dispositivos conectados.</td></tr>
<tr><td colspan="2">Visión general: Cuando un usuario inicia una nueva sesión en el sistema, éste realiza la inicialización del proceso temporizador que periódicamente ordenará el sondeo de dispositivos.</td></tr>
<tr><td colspan="2">Tipo: primario y esencial.</td></tr>
<tr><td colspan="2">Referencias: REQ[4]</td></tr>
<tr><td colspan="2">Curso típico de eventos:</td></tr>
<tr><td>Acción del usuario:</td><td>Respuesta del sistema:</td></tr>
<tr><td>1. Este caso de uso empieza</td><td>2. El sistema crea un proceso Temporizador que</td></tr>
</table>

cuando el usuario ha iniciado una sesión en el sistema y tras haber seleccionado la combinación de dispositivos que desea utilizar.	*periódicamente iniciará el sondeo de cada dispositivo seleccionado. Este proceso, además, llevará a cabo otras tareas como gestionar la sincronización de las vistas del entorno virtual de cada usuario conectado, así como la detección de colisiones entre objetos del mismo.*

Caso de uso: *Seleccionar Dispositivos*	
Actores: *Usuario (iniciador)*	
Propósito: *Seleccionar una combinación de dispositivos de interacción con el entorno virtual.*	
Visión general: *Un usuario que ha iniciado una sesión en el sistema selecciona la combinación de dispositivos que va a utilizar para moverse por el entorno virtual, controlar su punto de vista y mover su mano.*	
Tipo: *primario y esencial.*	
Referencias: *REQ[1], REQ[2], REQ[3]*	
Curso típico de eventos:	
Acción del usuario:	**Respuesta del sistema**:
1. *Este caso de uso empieza cuando el usuario inicia una sesión en el sistema.*	2. *Presenta un menú de selección de dispositivos.*
3. *Selecciona una combinación de dispositivos para controlar la navegación, el punto de vista y la mano virtual.*	4. *Solicita confirmación.*

5. *Confirma la selección.*	6. *Guarda la selección.*
Cursos alternativos: - Línea 5: *cancela la selección y realiza una nueva. Ir a 4.*	

Caso de uso: *Sondear Dispositivos*	
Actores: *Temporizador (iniciador), Mundo Virtual*	
Propósito: *Actualizar el estado de todos los dispositivos seleccionados*	
Visión general: *El actor Temporizador solicita que se realice el sondeo de los dispositivos seleccionados previamente. Como consecuencia de ello se produce la actualización de las posiciones y orientaciones de algunos de los elementos contenidos en el Mundo Virtual, como el avatar que representa al usuario o su mano virtual.*	
Tipo: *primario y esencial.*	
Referencias*: REQ[4], REQ[5], REQ[6], REQ[7]*	
Curso típico de eventos:	
Acción del iniciador:	**Respuesta del sistema**:
1. *Este caso de uso empieza cuando el actor Temporizador solicita que se consulte el estado de todos los dispositivos seleccionados.*	2. *Realiza un muestreo de los dispositivos y actualiza el estado de cada uno.* 3. *Solicita al actor Mundo Virtual la actualización del estado de los elementos virtuales controlados*

	por cada dispositivo, es decir, el avatar del usuario y su mano virtual, de acuerdo con el estado previamente muestreado.

Cursos alternativos:

- Línea 4: *el sistema detecta que ha sido activado o desactivado el mecanismo de señalización de objetos en el dispositivo correspondiente y se lo comunica al Mundo Virtual para que sea tenido en cuenta en el momento de pintar la escena.*

Caso de uso: *Gestionar Eventos*	
Actores: *Gestor de Eventos (iniciador), Mundo Virtual*	
Propósito: *Procesar los eventos procedentes de dispositivos cuyo funcionamiento está dirigido por ellos, y los que proceden del sistema multi-agente.*	
Visión general: *El Gestor de Eventos, tras capturar un evento procedente de algún dispositivo externo o del sistema multi-agente, solicita que sea procesado, lo que provocará cambios en el estado del Mundo Virtual.*	
Tipo: *primario y esencial.*	
Referencias: *REQ[4]*	
Curso típico de eventos:	
Acción del iniciador:	**Respuesta del sistema**:
1. *Este caso de uso empieza cuando el actor Gestor de Eventos captura un evento procedente de un dispositivo o del sistema multi-agente y solicita que sea procesado.*	2. *Procesa el evento provocando la actualización del estado de los elementos del Mundo Virtual afectados por el evento.*

Caso de uso: *Repintar Mundo*	
Actores: *Ciclo de Pintado (iniciador), Mundo Virtual*	
Propósito: *Solicitar el redibujado de la escena gráfica para reflejar posibles cambios en ella.*	
Visión general: *El actor Ciclo de Pintado solicita periódicamente que se realice un repintado de la escena gráfica, con lo que se deberá notificar al Mundo Virtual para que repinte cada uno de sus objetos virtuales.*	
Tipo: *primario y esencial.*	
Referencias: *REQ[8]*	
Curso típico de eventos:	
Acción del iniciador:	**Respuesta del sistema**:
1. *Este caso de uso empieza cuando el actor Ciclo de pintado solicita el refresco de la escena gráfica.*	2. *Comunica al Mundo Virtual la orden de que debe pintar cada uno de los objetos virtuales reflejando su estado actual de posición y orientación.*

Cada uno de los eventos que llegan al sistema se traduce en una operación invocada por alguno de los actores que interactúan directamente con el módulo de dispositivos. A su vez, cada operación lleva asociado un contrato que define el comportamiento del sistema ante estos eventos:

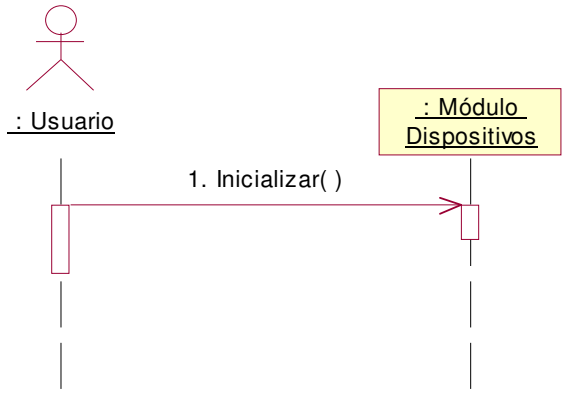


Figura 40: Operación "Inicializar"

Nombre:	**Inicializar()**
Responsabilidades:	*Crear el proceso o tarea periódica responsable de solicitar el sondeo de dispositivos.*
Referencias Cruzadas:	*Caso de uso: Inicializar* *Requisitos: REQ[4]*
Notas:	
Excepciones:	
Salida:	*Creación del Temporizador.*
Pre-condiciones:	*Una combinación de dispositivos ha sido seleccionada y existe una instancia por cada uno de ellos.*
Post-condiciones:	*El sistema ha quedado preparado para empezar a muestrear el estado de cada dispositivo periódicamente.*

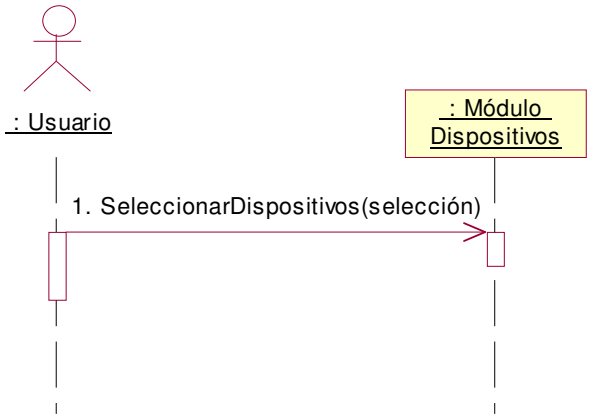


Figura 41: Operación "Seleccionar Dispositivos"

Nombre:	**SeleccionarDispositivos(selección)**
Responsabilidades:	*Crear e inicializar cada uno de los dispositivos seleccionados por el usuario y comprobar que todos son accesibles.*
Referencias Cruzadas:	*Caso de uso: Seleccionar dispositivos* *Requisitos: REQ[1], REQ[2], REQ[3]*
Notas:	
Excepciones:	*Si alguno de los dispositivos seleccionados no está disponible, indicar el error.*
Salida:	
Pre-condiciones:	*Existe una instancia de Escena.*
Post-condiciones:	*Por cada tipo de dispositivo indicado en el parámetro selección, se ha creado una instancia del mismo y se ha inicializado.*

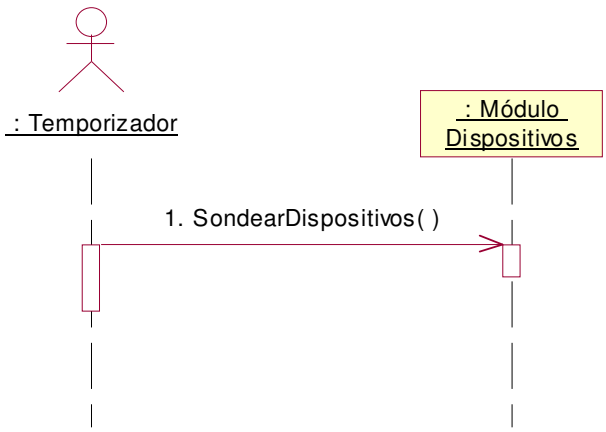


Figura 42: Operación "Sondear Dispositivos"

Nombre:	**SondearDispositivos()**
Responsabilidades:	*Actualizar el estado actual de todos los dispositivos seleccionados y hacer que se modifique, en función de él, el estado de los elementos del Entorno Virtual asociados a cada uno de ellos.*
Referencias Cruzadas:	*Caso de uso: Sondear dispositivos* *Requisitos: REQ[4], REQ[5], REQ[6], REQ[7]*
Notas:	
Excepciones:	
Salida:	*Solicitud al Entorno Virtual de modificación de la posición y orientación de la instancia del avatar que representa al usuario, de su punto de vista y de su mano virtual.*
Pre-condiciones:	*Una combinación de dispositivos ha sido seleccionada y existe una instancia por cada uno de ellos.* *Una instancia del Entorno Virtual ha sido creada previamente.*

Post-condiciones:	*Se ha modificado el estado interno de cada dispositivo.*

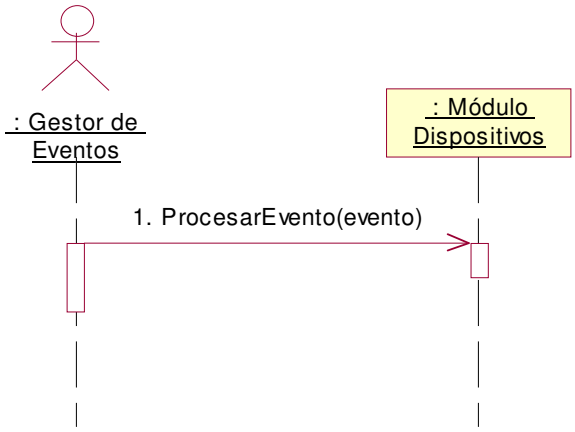


Figura 43: Operación "Procesar Evento"

Nombre:	**ProcesarEvento (evento)**
Responsabilidades:	*Hacer que se actualice el estado de los elementos del Entorno Virtual que son afectados por el tipo de evento en cuestión.*
Referencias Cruzadas:	*Caso de uso: Gestionar Eventos* *Requisitos: REQ[4]*
Notas:	
Excepciones:	
Salida:	*Solicitud al Entorno Virtual de modificación del estado de los objetos virtuales afectados por el evento.*
Pre-condiciones:	*Existe una instancia de Evento para el evento capturado.* *Una instancia del Entorno Virtual ha sido creada previamente.*
Post-condiciones:	

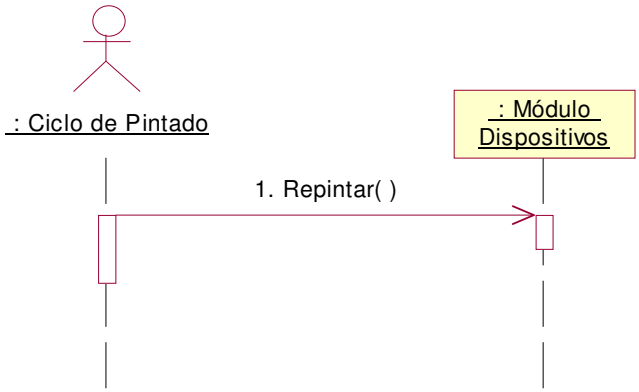


Figura 44: Operación "Repintar"

Nombre:	**Repintar ()**
Responsabilidades:	*Hacer que se redibujen todos los elementos del Entorno Virtual que forman la escena a visualizar.*
Referencias Cruzadas:	*Caso de uso: Repintar Mundo* *Requisitos: REQ[8]*
Notas:	
Excepciones:	
Salida:	*Solicitud al Entorno Virtual del pintado de los objetos virtuales de acuerdo a su estado actual.*
Pre-condiciones:	*Una instancia del Entorno Virtual ha sido creada previamente.*
Post-condiciones:	

4.5.3. Diseño detallado

El diseño de bajo nivel de este módulo se apoya en las librerías creadas para cada dispositivo involucrado que han sido descritas previamente, aunque, como se verá, podría utilizarse cualquier otro dispositivo sin variar el modelo.

Para empezar, uno de los casos de uso esenciales definidos en el diseño de alto nivel puede refinarse a un nivel de abstracción más bajo, haciendo referencia a las interfaces de usuario concretas o a los dispositivos específicos. Este es el caso de uso real:

<table>
<tr><td colspan="2">Caso de uso: Seleccionar Dispositivos</td></tr>
<tr><td colspan="2">Actores: Usuario (iniciador)</td></tr>
<tr><td colspan="2">Propósito: Seleccionar una combinación de dispositivos de interacción con el entorno virtual.</td></tr>
<tr><td colspan="2">Visión general: Un usuario que ha iniciado una sesión en el sistema selecciona la combinación de dispositivos que va a utilizar para moverse por el entorno virtual, controlar su punto de vista y mover su mano.</td></tr>
<tr><td colspan="2">Tipo: primario y real.</td></tr>
<tr><td colspan="2">Referencias: REQ[1], REQ[2], REQ[3]</td></tr>
<tr><td colspan="2">Curso típico de eventos:</td></tr>
<tr><td>Acción del usuario:</td><td>Respuesta del sistema:</td></tr>
<tr><td>1. Este caso de uso empieza cuando el usuario inicia una sesión en el sistema.</td><td>2. Presenta una pantalla de selección de dispositivos, organizada en tres categorías, que permite seleccionar una combinación de teclado, joystick, guante y sensores de posición para controlar el movimiento del avatar, su punto de vista y el movimiento de su mano.</td></tr>
<tr><td>3. Selecciona</td><td>4. El sistema restringe las posibilidades de selección en</td></tr>
</table>

sucesivamente los dispositivos de cualquiera de las categorías presentadas.	*las restantes categorías de acuerdo con la **Figura 36**.*
5. *Cuando ha terminado confirma la selección.*	*6. Guarda la selección.*

Este caso de uso se asocia con la siguiente interfaz de usuario:

Figura 45: Pantalla de selección de dispositivos

El siguiente paso, en el diseño de bajo nivel, es definir los diagramas de estructura estática y los diagramas de interacción para las operaciones descritas.

Primeramente se ha planteado un posible diseño como prototipo para la integración de las distintas librerías de dispositivos desarrolladas dentro del mismo módulo. Posteriormente este diseño se ha refinado para darle características de flexibilidad y escalabilidad.

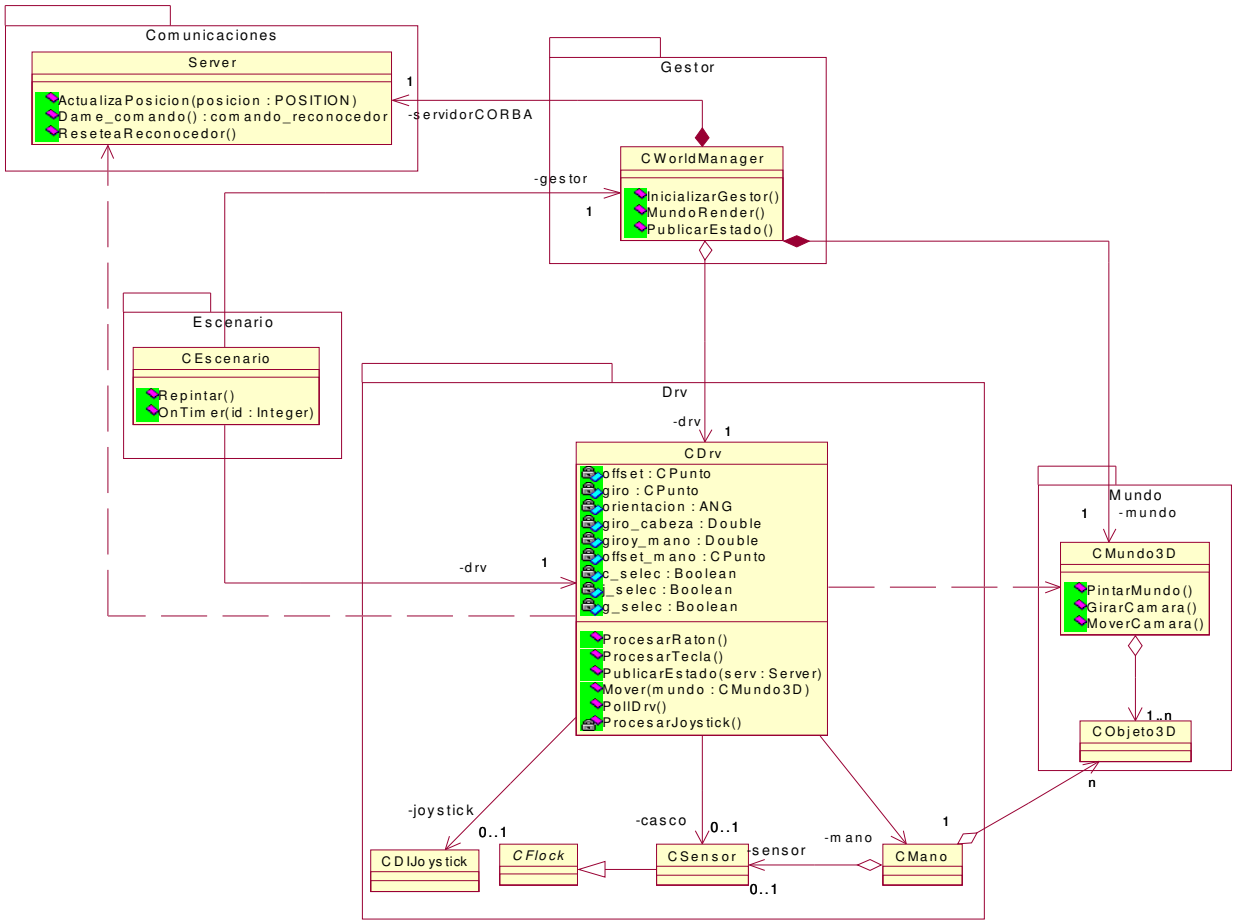


Figura 46: Diagrama de clases del prototipo

En un primer vistazo, pueden identificarse las clases incluidas en cada uno de los paquetes expuestos antes.

- Paquete *Escenario*:
  - Clase "CEscenario": Responsable de solicitar periódicamente la actualización del estado del entorno virtual a través de la operación "OnTimer()" y después pedir su visualización gráfica mediante "Repintar()". Además, también se encarga de realizar la captura de ciertos eventos externos, como los procedentes del teclado o del ratón, y transmitirlos al paquete *Drv*.
- Paquete *Gestor*:
  - Clase "CWorldManager": Mantiene relaciones de agregación con las clases "CMundo3D" (paquete *Mundo*), "CDrv" (paquete *Drv*), y "Server" (paquete *Comunicaciones*), ya que la existencia de objetos de éstas está ligada a la clase encargada de su gestión, en este sentido se pueden considerar como partes de ella. Es responsable de solicitar al paquete *Drv* el sondeo de dispositivos y la actualización del estado del mundo. Además, pedirá a las clases del paquete de *Comunicaciones* que transmitan los cambios realizados al resto del sistema y a otros usuarios.
- Paquete *Drv*:
  - Clase "CDrv": Esta clase es el núcleo del módulo de gestión de dispositivos. En ella se mantiene información acerca de la posición y orientación del avatar que representa al usuario, así como de su mano virtual. Bajo demanda, está información será actualizada por medio del método *PollDrv()*, que sondea cada uno de los dispositivos con los que tiene relaciones de asociación. Posteriormente, el paquete *Gestor* pedirá que se hagan efectivos esos cambios en el mundo invocando al método *Mover(mundo)* que, haciendo uso de los datos previamente muestreados, invocará las actualizaciones en el estado de los objetos

del mundo virtual. La clase también es responsable de procesar otros eventos procedentes del *Escenario* y de comunicar los cambios en el estado al paquete de *Comunicaciones*. La clase "CDrv" es el único punto de acceso a las clases que forman las librerías de dispositivos, pues su misión es integrarlas a todas ellas y ocultar sus detalles al exterior.

- Paquete *Mundo*:
  - Clases "CMundo3D" y "CObjeto3D": En estas clases reside la información sobre la geometría de todos los objetos del entorno virtual.
- Paquete *Comunicaciones*:
  - Clase "Server": aunque este paquete no es objeto de análisis en este escrito, tan sólo comentar que, a través de un conjunto de clases de las que su mayor representante es ésta, se coordina la comunicación entre usuarios dentro del sistema multi-usuario, y también con el sistema de agentes vía una interfaz CORBA.

Se pueden encontrar similitudes en este diseño con un patrón Modelo-Vista-Controlador, donde el modelo sería el mundo, la vista el escenario y el controlador la clase "CWorldManager". Las razones de esta forma de proceder no son en absoluto arbitrarias, sino las de lograr la mayor independencia posible de la interfaz gráfica de usuario sobre la representación geométrica del mundo. Las ventajas son conseguir que posibles cambios en la definición del mundo o incluso el cambio completo de representación del entorno no tengan efecto sobre el resto del modelo.

A pesar de todo, este diseño de clases ha sido rechazado en un ciclo posterior de desarrollo, aunque su filosofía se mantiene. La razón es la poca flexibilidad y escalabilidad que presenta en el tratamiento de dispositivos.

La gestión de todos los dispositivos está centralizada en la clase “CDrv”, lo que supone que la incorporación de nuevos dispositivos o los cambios en éstos provocarán una reestructuración completa de esta clase. Es cierto que, de cara al exterior, se aísla de las particularidades de diseño de cada uno de los dispositivos pero, sin embargo, este problema no ha desaparecido sino tan sólo ha sido camuflado.

El siguiente diseño de clases, fruto de un ciclo de refinamiento, intenta paliar las desventajas expuestas.

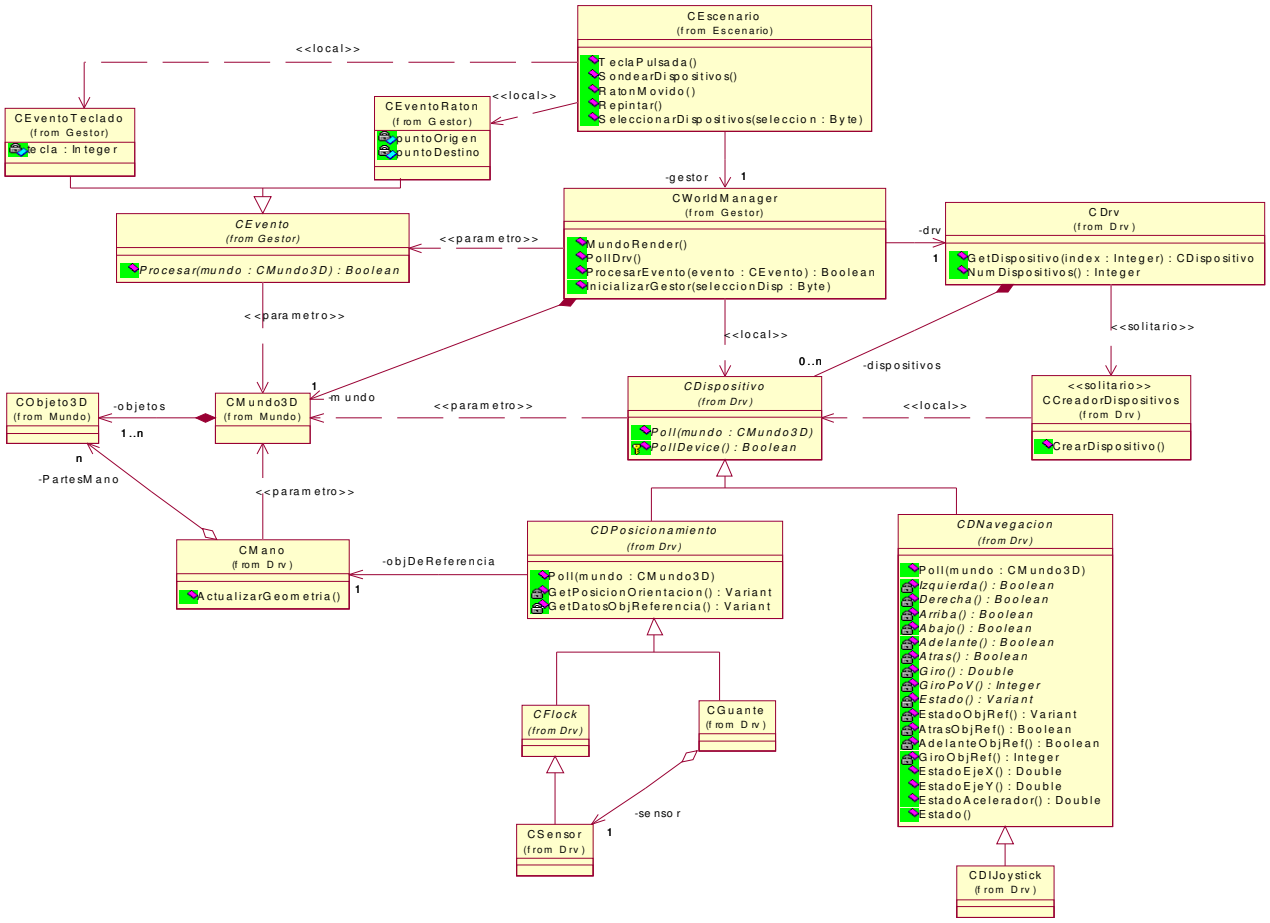


Figura 47: Diagrama de clases refinado

DEFINICIÓN DE CLASES, ATRIBUTOS Y MÉTODOS:

Las definiciones que a continuación se presentan son puramente descriptivas y únicamente reflejan las características más relevantes del diagrama de clases.

CLASE	DESCRIPCIÓN
CEscenario	Clase responsable de gestionar la vista del escenario virtual, captura los eventos externos y ordena la actualización del entorno.
CWorldManager	Clase encargada de la gestión y mantenimiento del mundo virtual, sondear dispositivos y procesar eventos.
CDrv	Colección de dispositivos presentes en el sistema.
CCreadorDispositivos	Clase responsable de la creación de los diferentes dispositivos.
CDispositivo	Clase abstracta que representa un dispositivo genérico.
CDPosicionamiento	Clase abstracta que representa aquellos dispositivos cuya tecnología está basada en sensores.
CDNavegacion	Clase abstracta que representa a dispositivos basados en su accionamiento directo, tipicamente dirigidos a la navegación por el entorno virtual.
CMano	Clase que encapsula la geometría de cada una de las partes de la mano virtual y es responsable de su actualización.
CEvento	Clase abstracta que representa un evento externo genérico cuyos efectos se deben reflejar en el mundo virtual.
CEventoTeclado	Evento asociado a las pulsaciones de teclado.

CLASE	DESCRIPCIÓN
CEventoRaton	Evento asociado a los movimientos de ratón.

CLASE			DESCRIPCIÓN
CEscenario	**ATRIBUTOS**	gestor	Objeto de la clase CWorldManager a través del que se realiza la comunicación con el mundo virtual.
	MÉTODOS	SeleccionarDispositivos (in:selección)	Método que solicita la creación de los objetos correspondientes a los dispositivos seleccionados.
		SondearDispositivos()	Método que solicita el sondeo de nuevos datos desde cada uno de los dispositivos creados.
		TeclaPulsada(in: tecla)	Método que crea un evento como respuesta a la pulsación de una tecla y solicita que sea procesado.
		RatonMovido (in: origen, in: destino)	Método que crea un evento como respuesta al movimiento del ratón y solicita que sea procesado.
		Repintar()	Método que solicita que se refresque la vista del escenario virtual.
CWorldManager	**ATRIBUTOS**	drv	Colección de dispositivos creados.

CLASE			DESCRIPCIÓN
		mundo	Objeto de la clase CMundo3D que mantiene la representación geométrica del entorno virtual
	MÉTODOS	InicializarGestor (in: seleccionDisp)	Método encargado de la creación de la colección de dispositivos según la selección hecha.
		MundoRender()	Método que envía un mensaje al mundo virtual para que se muestre gráficamente.
		ProcesarEvento (in: evento)	Método que procesa un evento externo.
		PollDrv()	Método que realiza el sondeo de cada uno de los dispositivos de la colección.
CDrv	**ATRIBUTOS**	dispositivos	Colección indexada de los dispositivos seleccionados.
	MÉTODOS	GetDispositivo(in: index)	Método que proporciona un dispositivo a través de un índice.
		NumDispositivos(): Integer	Método que proporciona el número de dispositivos que se han seleccionado.
CCreadorDispositivos	**ATRIBUTOS**		

CLASE			DESCRIPCIÓN
	MÉTODOS	CrearDispositivo (in: tipo, in: modo)	Método encargado de crear un dispositivo en función de su tipo y modo de funcionamiento.
CDispositivo	**ATRIBUTOS**	tipo	Tipo de dispositivo (HMD, DATAGLOVE, JOYSTICK)
		modo	Modo de funcionamiento para HMD y JOYSTICK según si se desea controlar la orientación del punto de vista, el movimiento o ambos.
	MÉTODOS	Poll(in: mundo)	Método abstracto cuya misión es obtener los datos del dispositivo y actualizar el estado del mundo en consecuencia.
		PollDevice(): boolean	Método privado y abstracto que debe ser invocado al comienzo del método Poll y que insta al dispositivo físico a actualizar su estado. Su implementación será proporcionada por la librería desarrollada para cada dispositivo particular.
CDPosicionamiento	**ATRIBUTOS**	objDeReferencia	Objeto que se asocia al movimiento de la cámara y que en este caso se corresponde con la mano virtual.

CLASE			DESCRIPCIÓN
	MÉTODOS	Poll(in: mundo)	Implementación del método homónimo heredado de CDispositivo.
		GetPosicionOrientacion(): datos	Método virtual que será sobrescrito por la clase del dispositivo concreto para proporcionar los datos de posición y orientación.
		GetDatosObjReferencia(): datos	Método virtual que será sobrescrito por la clase del dispositivo concreto para proporcionar los datos de posición y orientación del objeto de referencia.
CDNavegación	**ATRIBUTOS**		
	MÉTODOS	Poll(in: mundo)	Implementación del método homónimo heredado de CDispositivo.
		Izquierda(): boolean	Métodos abstractos que han de ser implementados por la clase del dispositivo concreto y que indican si ha sido accionado en la dirección del mismo nombre.
		Derecha(): boolean	
		Arriba(): boolean	
		Abajo(): bolean	
		Adelante(): boolean	
		Atrás(): boolean	

CLASE			**DESCRIPCIÓN**
		Giro(): double	Método abstracto que ha de ser implementado por la clase del dispositivo concreto y que proporciona la cantidad de giro aplicado sobre él
		GiroPoV(): integer	Método abstracto que ha de ser implementado por la clase del dispositivo concreto y que proporciona la dirección en la que se deberá orientar el punto de vista.
		EstadoObjRef(): datos	Métodos virtuales que informan sobre el estado del objeto de referencia que irá ligado a la cámara y que en este caso se corresponde con la mano virtual.
		AtrasObjRef(): boolean	
		AdelanteObjRef(): boolean	
		GiroObjRef(): integer	
		EstadoEjeX(): double	Método abstracto que informa sobre la cantidad de movimiento lateral aplicado sobre el dispositivo.
		EstadoEjeY(): double	Método abstracto que informa sobre la cantidad de movimiento vertical aplicado sobre el dispositivo.
		EstadoAcelerador(): double	Método abstracto que informa sobre el factor de aceleración que debe ser aplicado a cada movimiento.

CLASE			DESCRIPCIÓN
		Estado(): datos	Método virtual que informa sobre el estado general del dispositivo
CMano	**ATRIBUTOS**	PartesMano	Colección de objetos virtuales que articulan las distintas partes de la mano.
	MÉTODOS	ActualizarGeometria (in: datos)	Método que actualiza la posición y orientación de cada una de las partes de la mano según los datos de entrada.
CEvento	**ATRIBUTOS**		
	MÉTODOS	Procesar(in: mundo) : boolean	Método abstracto cuya misión es hacer que las subclases de CEvento alteren el estado del mundo según sus características. Devueve cierto si el evento se ha procesado con éxito.
CEventoTeclado	**ATRIBUTOS**	tecla	Código de la tecla que provoca el evento al ser pulsada.
	MÉTODOS	Procesar(in: evento) : boolean	Implementación del método heredado de CEvento que provoca un cambio en el mundo según sea la tecla pulsada.
CEventoRaton	**ATRIBUTOS**	PuntoOrigen	Coordenadas de la situación origen del ratón.

CLASE			DESCRIPCIÓN
		PuntoDestino	Coordenadas de la situación final del ratón.
	MÉTODOS	Procesar(in: evento) : boolean	Implementación del método heredado de CEvento que provoca un cambio en el mundo según el desplazamiento del ratón.

OBSERVACIONES:

En el diagrama se observa que la estructura general entre escenario, gestor y dispositivos permanece, pero se ha optado por retocar la organización de éstos últimos. Los cambios efectuados se resumen en los siguientes puntos:

1. La clase "CDrv" ha sido liberada de sus responsabilidades para pasar a ser una simple colección de dispositivos que no realiza ningún proceso de lógica de negocio.
2. Se ha introducido una jerarquía de dispositivos cuya raíz es la clase abstracta "CDispositivo". Esta clase permite al gestor sondear los distintos tipos de dispositivos de manera general, es decir, haciendo uso de las ventajas del polimorfismo. El gestor ya no habrá de preocuparse por qué clase de dispositivo está sondeando y de qué manera ha de hacerlo.
3. La creación de los dispositivos concretos que van a estar presentes en el sistema pasa a ser, siguiendo un patrón de diseño creacionista, responsabilidad de una clase especializada en ello. Es la clase "CCreadorDispositivos" y su razón de ser es aislar a "CDrv" de los distintos procesos de creación para cada tipo de dispositivo y evitar dependencias que repercuten en la modificabilidad del diseño, proporcionando a la propia clase "CDrv", y por consiguiente a "CWorldManager", objetos genéricos de la clase "CDispositivo".

4. La jerarquía de dispositivos incluye todos aquellos dispositivos cuya obtención de datos es bajo demanda. De éstos, hace la siguiente clasificación: Por un lado, los dispositivos de navegación, clase "CDNavegacion", y por otro los de posicionamiento, clase "CDPosicionamiento". En realidad esta clasificación atiende a aquellos dispositivos que no están basados en sensores de ubicación y los que sí, respectivamente.

5. Las clases "CDNavegación" y "CDPosicionamiento" albergan la lógica necesaria para actualizar la geometría del mundo con los datos obtenidos de cada dispositivo. Por tanto, las clases situadas en los extremos de la jerarquía, que se corresponden con las librerías de acceso a los dispositivos previamente diseñadas, serán ligeramente ampliadas para implementar un conjunto de operaciones abstractas, heredadas de "CDNavegación" o de "CDPosicionamiento", según el caso, y también de "CDispositivo", como la operación "PollDevice", en las que se basan los algoritmos de la lógica mencionada.

6. Se introduce otra jerarquía para la gestión de eventos. Aquellos dispositivos cuyo funcionamiento esté basado en eventos, como pueden ser el teclado o el ratón, u otras partes del sistema que generan eventos que, a su vez, provocan efectos en el mundo virtual, tendrán su correspondiente clase representante en esta jerarquía, que realizará el procesamiento particular de su evento asociado. Cada evento producido en el sistema será propagado hacia el gestor y éste será el encargado de ordenar su procesamiento.

Todas estas ampliaciones en el diseño introducen características de escalabilidad, al poder ampliar el rango de dispositivos manejados sin más que implementar su clase de acceso e introducirla en la jerarquía; de modificabilidad, al hacer uso de las mencionadas técnicas de polimorfismo; y de portabilidad, pues la jerarquía de dispositivos es en gran parte independiente del mundo, del escenario o del sistema de agentes.

Por otra parte, estas mejoras implican un modelo más complejo que hacen de él un sistema de más difícil entendimiento.

Para ilustrar los procesos más complejos que en él se llevan a cabo, de acuerdo con el modelo de casos de uso, se presentan únicamente algunos diagramas de interacción:

- **Sondeo de dispositivos:**

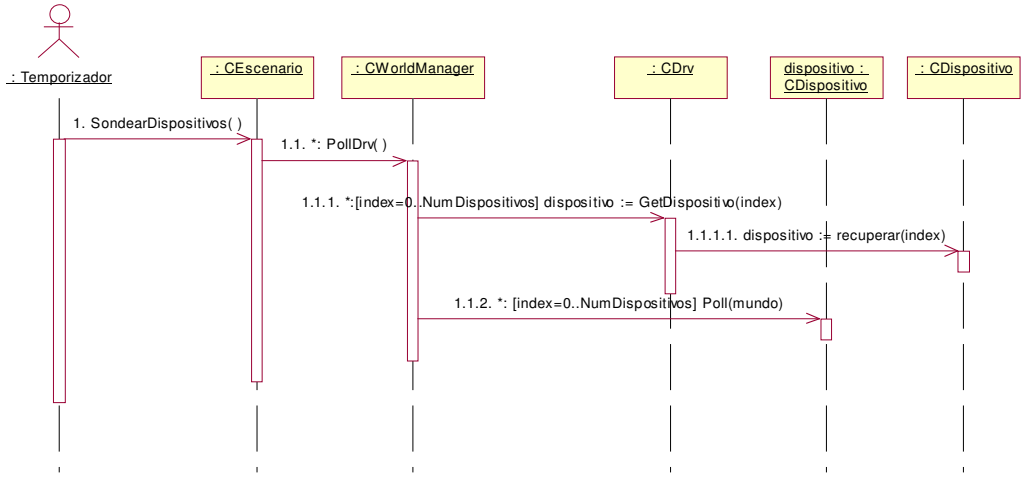


Figura 48: Sondear dispositivos

La operación de sondeo del dispositivo (método *Poll(mundo)* de la clase CDispositivo) realiza distintas operaciones para actualizar el estado del mundo virtual dependiendo del tipo de dispositivo en cuestión, ya que, según sea éste, los datos recogidos pueden referirse a coordenadas de posición y orientación en dispositivos de posicionamiento, o a cantidad de movimiento en dispositivos de navegación. Además, los dispositivos de navegación están diseñados para el desplazamiento dentro del entorno virtual, por lo que el tratamiento que se realiza en ellos con respecto al objeto de referencia asociado a la cámara es más simple que en los dispositivos de posicionamiento, limitándose a movimientos sencillos de giro y desplazamiento.

Estas operaciones se ilustran con los siguientes diagramas de actividad para los dispositivos que están basados en sensores y los que no, respectivamente:

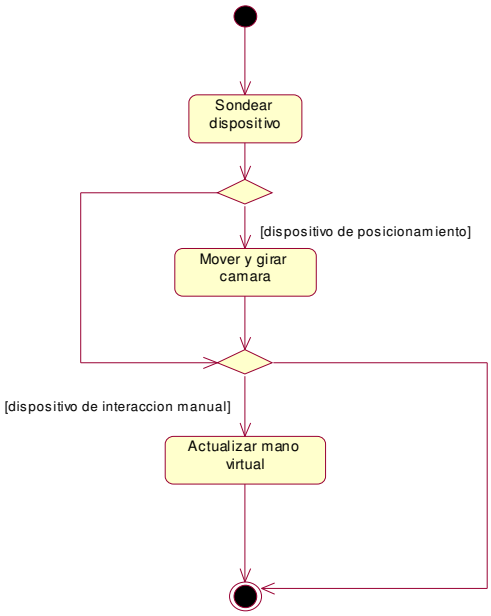


Figura 49: Diagrama de actividad para el sondeo de dispositivos de posicionamiento

Muestra de pseudocódigo:

```
void CDPosicionamiento::Poll(CMundo3D mundo)
{
   PollDevice();
   DATOS sondeo;
   sondeo = GetPosicionOrientacion();
   if(sondeo == exito) {
   //se trata de un dispositivo de posicionamiento
      mundo.GirarYMoverCamara(sondeo);
   }
   sondeo = GetDatosObjReferencia();
   if(sondeo == exito) {
   //se trata de un dispositivo de interacción manual
      if(objeto_asociado == NULL) {
         objeto_asociado = new CMano(mundo);
      }
      objeto_asociado->ActualizarGeometria(sondeo);
   }
}
```

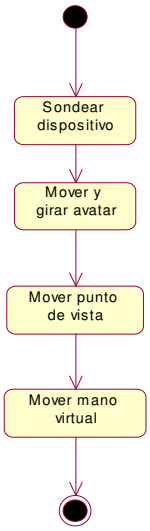


Figura 50: Diagrama de actividad para el sondeo de dispositivos de navegación

Muestra de pseudocódigo:

```
void CDNavegacion::Poll(CMundo3D mundo)
{
   PollDevice();
   ESTADO_DISPOSITIVO sondeo;
   sondeo = Estado();

   //Movimiento del avatar del usuario
   mundo.MoverYGirarAvatar(sondeo);
   mundo.MoverPuntoDeVista(sondeo);

   sondeo = EstadoObjRef();
   //Movimiento del objeto asociado a la camara
   mundo.MoverYGirarObjRef(sondeo);
}
```

- **Selección de dispositivos:**

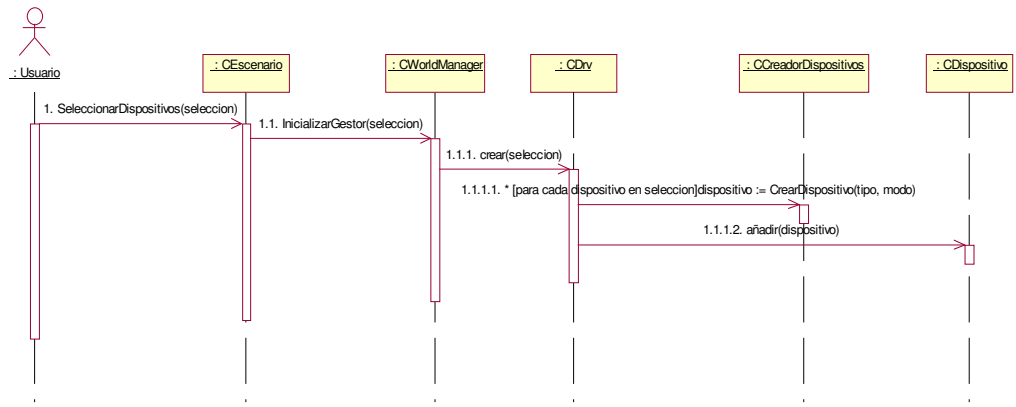


Figura 51: Seleccionar dispositivos

- **Repintar:**

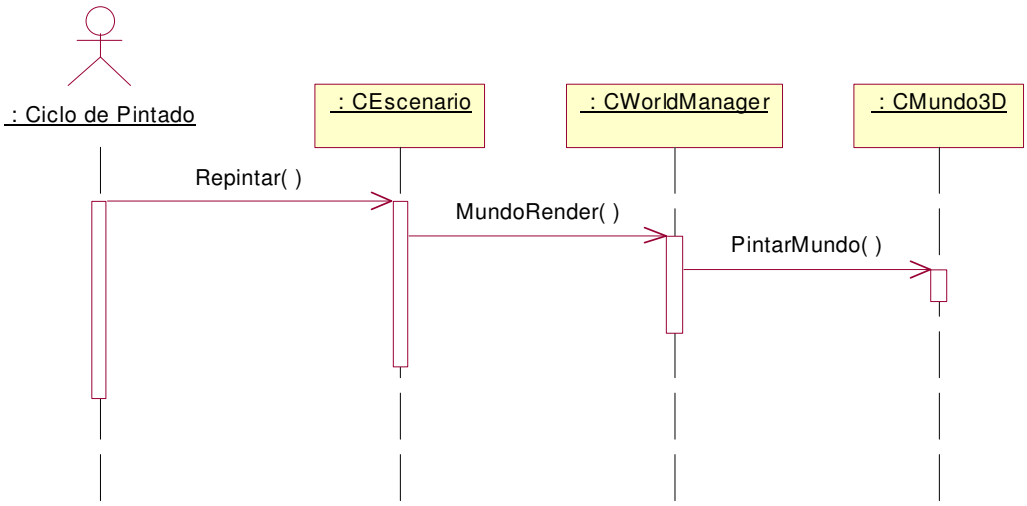


Figura 52: Repintar

- **Procesar Eventos:**

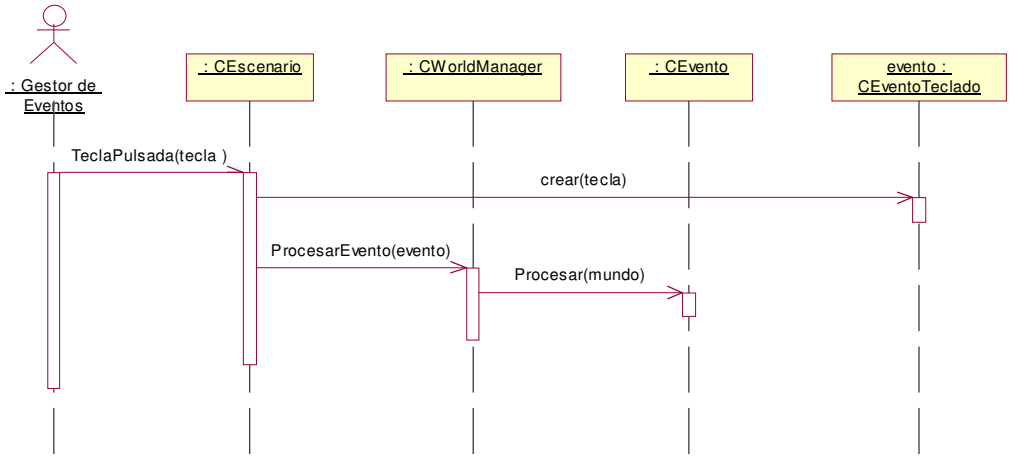


Figura 53: Evento de teclado

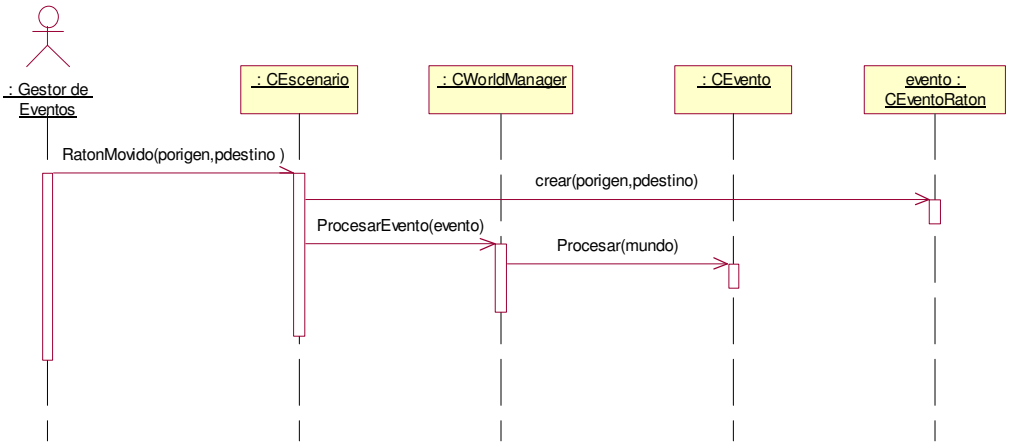


Figura 54: Evento de ratón

El proceso realizado por la operación *Procesar(mundo)* de la clase "CEvento" es simplemente el de solicitar al mundo virtual que actualice la posición y orientación del avatar del usuario, de su mano o de su punto de vista, en función de la tecla pulsada o del movimiento realizado con el ratón.

```
Procesar(CMundo3D mundo)
{
   nueva_situacion = CalcularSituacion(estado_evento);
   mundo.Actualizar(nueva_situacion);
}
```

4.5.4. Implementación

La implementación de este módulo, debido a las restricciones impuestas por las APIs ofrecidas por los distintos fabricantes de los dispositivos utilizados, y las características derivadas del modelo de aplicación empleado por el paquete *Escenario,* que hace uso de la jerarquía de clases MFC[1], cuyos beneficios no son objeto de discusión aquí, se ha realizado en el lenguaje C++, con el entorno de desarrollo proporcionado por Microsoft Visual C++.

La mezcla de la tecnología de Microsoft y su modelo de programación, la de cada uno de los dispositivos usados y la de las librerías gráficas para la representación geométrica del mundo virtual, son las razones del especial hincapié que se ha puesto en los aspectos de diseño, modificabilidad, flexibilidad, escalabilidad, etc. La implementación de cada uno de los subsistemas con su correspondiente tecnología no debe suponer, y de hecho así es, ningún inconveniente a la hora de integrarlos en el mismo sistema.

Como ya ha sido mencionado, el módulo implementado hace uso de las librerías de dispositivos previamente desarrolladas. Para conseguir la integración de las mismas ha sido necesario efectuar algunas ligeras modificaciones en ellas con el fin de implementar aquellos métodos abstractos impuestos por la jerarquía de dispositivos desarrollada.

[1] *Microsoft Foundation Classes*

Además, se ha intentado definir los tipos de datos manejados por los dispositivos de la manera más genérica posible, de forma que la jerarquía tenga una cierta tolerancia a los cambios efectuados en los mismos.

Si ponemos la atención en lo que hemos venido llamando "objeto de referencia", vemos cómo, en la jerarquía de dispositivos, tanto los dispositivos de posicionamiento como los de navegación definen los métodos de lectura de datos asociados a dicho objeto procurando siempre que el tipo de los mismos sea lo más genérico posible. En el futuro sería conveniente hacer a este objeto más abstracto haciendo por ejemplo que los dispositivos de posicionamiento y navegación hagan uso de una interfaz de acceso genérica y común que pueda ser implementada de diversas formas.

Por último, se presentan algunos datos de gestión de configuración, como los nombres de los ficheros de código fuente desarrollados y su correspondencia con las clases implementadas en ellos:

Clase	**Ficheros de código fuente**
CEscenario	Escenario.cpp, Escenario.h
CWorldManager	WorldManager.cpp, WorldManager.h
CDrv	Drv.cpp, Drv.h
CCreadorDispositivos	CreadorDispositivos.cpp, CreadorDispositivos.h
CDispositivo	Dispositivo.cpp, Dispositivo.h
CDNavegacion	DNavegacion.cpp, DNavegacion.h
CDPosicionamiento	DPosicionamiento.cpp, DPosicionamiento.h

Clase	Ficheros de código fuente
CFlock	Flock.cpp, Flock.h
CSensor	Sensor.cpp, Sensor.h
CGuante	Guante.cpp, Guante.h
CDIJoystick	DIJoystick.cpp, DIJoystick.h
CEvento	Evento.cpp, Evento.h
CEventoTeclado	EventoTeclado.cpp, EventoTeclado.h
CEventoRaton	EventoRaton.cpp, EventoRaton.h
CMano	Mano.cpp, Mano.h

En total son 28 ficheros, entre archivos de declaración y de definición, para implementar 15 clases que representan aproximadamente unas 5000 líneas de código.

5. Gestión de interacciones en un IVET

5.1. Introducción

Las interacciones que existen entre un usuario y el entorno virtual en el que está inmerso generan una cierta cantidad de información que debe ser gestionada para poder realizar correctamente la supervisión del entrenamiento.

Cuando un usuario se mueve, manipula objetos, hace preguntas, etc., el sistema ha de reaccionar para poder corregir sus movimientos, proporcionarle información acerca de los objetos, darle explicaciones... Además, todas estas interacciones han de quedar registradas con el propósito de construir perfiles de aprendizaje o simplemente históricos de formación.

Como se vio en el capítulo sobre la descripción de MAEVIF, la información sobre las interacciones y movimientos que tienen lugar en el entorno virtual llega al sistema de tutoría a través de los agentes de comunicación. Éstos, a su vez, la distribuyen entre los agentes de tutoría encargados de cotejar estas interacciones y desplazamientos con los planes previamente construidos y con las trayectorias óptimas a seguir en cada desplazamiento, y entre un conjunto de agentes involucrados en analizar si es posible realizar físicamente cada acción y en cuyo caso ordenar las simulaciones que son consecuencia de las mismas. Este último grupo de agentes está formado por el *Agente Mundo*, el *Agente Experto*, y algunos de sus agentes subordinados, incluidos en los que se ha dado en llamar *Agentes de acción*.

5.2. Agentes de acción en MAEVIF

Existe un grupo de agentes dentro de la arquitectura presentada en la **Figura 14** que, por su conocimiento especializado, intervienen directamente en los procesos de planificación, y algunos de ellos, posteriormente, en la fase de simulación.

Estos agentes son el *Agente Planificador de Trayectorias, Agente de Simulación* y *Agente de Actuación*.

Cada uno de estos agentes, durante el proceso de construcción de planes, conoce qué acciones pueden hacer cumplir ciertos objetivos.

El *Agente Planificador de Trayectorias* es capaz de determinar las trayectorias que ha de seguir el alumno para realizar la secuencia de acciones planificada. Se encarga de satisfacer objetivos del tipo *"EstarEn(posición)"* mediante acciones del tipo *"MoverA(posición)"*.

El *Agente de Actuación* sabe cómo satisfacer objetivos simples mediante acciones básicas sobre el entorno virtual como "pulsar un botón", "coger un objeto", etc.

Por último, el *Agente de Simulación* sabe cómo satisfacer objetivos relacionados con el estado de la simulación subyacente mediante tareas complejas formadas por acciones simples, como "elevar la temperatura del reactor".

Una vez terminada la planificación, estos dos últimos agentes también participan en la simulación del comportamiento del entorno frente a las interacciones. En las páginas sucesivas nos centramos en el *Agente de Actuación*, su misión y su relación con el resto de agentes.

5.3. Agente de Actuación

5.3.1. Introducción

Esta entidad, que posteriormente veremos si se concreta en uno o varios agentes, surge para vertebrar el procesamiento de las interacciones que inicia el usuario con el entorno virtual y las posteriores reacciones de éste.

Las responsabilidades que asumirá la entidad dentro del sistema multi-agente se derivan del conocimiento que gestiona. Toda la información relativa al entorno virtual sobre aquellos objetos con los que se puede interaccionar y la manera de hacerlo, es almacenada por el *Agente de Actuación*, y es a él a quien se dirigirán otros agentes solicitando la verificación de acciones emprendidas por el usuario o respuestas acerca de las posibilidades de actuación sobre el entorno.

Por decirlo de alguna manera, esta entidad dicta las reglas de interacción en un entorno virtual, consiguiendo que la simulación del comportamiento de éste sea coherente y se ajuste a la realidad. Es capaz, por ejemplo, de abortar simulaciones de acciones tales como coger objetos, si resulta físicamente imposible hacerlo por no estar al alcance, o proporcionar a quien lo solicite las consecuencias de las acciones realizadas por el usuario para que puedan ser simuladas y, de esta forma, dotar de dinamismo al entorno.

Por su misión vertebradora, este agente o agentes mantendrán una estrecha colaboración con diversas entidades dedicadas tanto a la supervisión y seguimiento de los alumnos como a la comunicación con el entorno y simulación del mismo.

Fundamentalmente, la consecución de los objetivos del *Agente de Actuación* se distribuye entre las dos fases importantes por las que pasa el sistema, dando lugar a las dos siguientes facetas.

5.3.1.1. Faceta de planificación

Esta faceta es la que, principalmente, da al *Agente de Actuación* su estatus de *Agente de acción*.

La elaboración de planes de actuación para encontrar soluciones a los problemas propuestos es una tarea compleja en la que han de participar las distintas entidades que gozan de conocimiento especializado sobre el entorno.

En este caso, el proceso de planificación está inspirado en el algoritmo STRIPS [Fikes & Nilsson, 1971] al que se le han introducido ciertas variaciones. STRIPS se fundamenta en el paradigma de resolución de problemas denominado GPS[1].

En el análisis de cualquier problema encontramos tres conceptos básicos:

- Enunciados (o estados). Son los hechos que se pueden constatar como ciertos en cada momento.

[1] *General Problem Solver*

- Operadores. Acciones que posibilitan la transición hacia nuevos estados del problema.
- Meta. Estado final o solución del problema.

GPS y STRIPS tratan de averiguar las diferencias que existen entre el estado inicial del problema y la meta deseada, y transformar, mediante la aplicación sucesiva de operadores, dicho estado en otros que reduzcan esas diferencias hasta alcanzar la solución.

En este proceso se precisa, necesariamente, definir fórmulas lógicas para describir tanto los estados por los que pasa el problema, como los operadores que dirigen las transiciones entre ellos.

Un estado está formado por un conjunto de predicados lógicos o hechos del tipo *"nombre_predicado(x_1, x_2,...x_n)"* que tienen asociado un valor de verdad.

Por otro lado, los operadores son acciones del tipo *"nombre_acción(x_1, x_2,...x_n)"* que tienen la siguiente información asociada:

- Precondiciones: Conjunto de predicados o hechos que han de cumplirse en el estado actual para poder aplicar el operador.
- Postcondiciones a "añadir": Conjunto de predicados o hechos que se cumplen después de aplicar el operador y que modifican el estado actual.
- Postcondiciones a "borrar": Conjunto de predicados o hechos que han dejado de ser ciertos tras aplicar el operador y que por tanto habrán de ser eliminados del estado actual.

Concretamente, STRIPS es un algoritmo que realiza una búsqueda del plan hacia atrás y en profundidad. Su funcionamiento, en términos generales, es el siguiente:

1. Partiendo de la descripción del estado meta, se consideran las diferencias con respecto al estado inicial. Cada diferencia es una meta u objetivo a satisfacer.

2. Se construye un árbol de búsqueda, cuya raíz será el conjunto de diferencias y sus hijos las distintas ordenaciones de los predicados de dicho conjunto organizados en una estructura de pila FIFO también llamada pila de objetivos activos.
3. Se recorre el árbol en profundidad:
    a. En cada nodo sucesor se verifica si la meta que está en la cima de la pila es cierta en el estado actual. Cuando la pila está vacía el plan ha sido encontrado.
        i. Si la meta es cierta, se elimina de la pila y se continúa el estudio de la siguiente meta volviendo al punto 3.a.
        ii. Si la meta no es cierta, se busca un operador capaz de añadir dicha meta al estado actual. A continuación se expande el árbol, a partir del nodo actual, con tantos nodos como ordenaciones distintas haya de las precondiciones del operador, y en cada uno se apila el operador y seguidamente una de esas ordenaciones.
4. Si en la cima de la pila lo que hay no es una meta sino que es un operador, significa que han sido verificadas todas las condiciones que permiten su aplicación, pudiendo, entonces, añadirlo al plan y modificar el estado para poder seguir con el recorrido del árbol.

El resultado de este algoritmo, en caso de éxito, es una sucesión de operadores que constituyen un plan de actuación. Sin embargo, el plan calculado puede no ser único ni óptimo, es por ello que, sobre este algoritmo se han realizado una serie de alteraciones para obtener no cualquier plan, sino el mejor. Estas modificaciones consisten, por ejemplo, en hacer el recorrido del árbol en amplitud y no en profundidad, con el propósito de obtener el plan que tiene menor número de operadores.

El *Agente de Actuación* juega un papel primordial en la ejecución de este algoritmo, pues es él quien almacena la definición genérica de los operadores, sus precondiciones y sus postcondiciones, que se pueden aplicar sobre cada objeto manipulable presente en el entorno virtual. Por eso, cuando el *Agente Planificador*, encargado de controlar la ejecución del algoritmo, necesita aplicar un operador concreto para modificar el estado actual, solicitará al *Agente de Actuación* que lo instancie a partir de los argumentos proporcionados, y que actualice dicho estado con las postcondiciones correspondientes. Además, el *Agente de Actuación* es responsable de buscar e instanciar aquellos operadores que satisfacen determinados objetivos y hacerlos llegar al *Agente Planificador* para que éste pueda expandir el árbol de búsqueda.

5.3.1.2. Faceta de simulación

Una vez que el sistema ha concluido la fase de planificación y dispone de una solución óptima para el problema que han de resolver los estudiantes, comienza la fase de supervisión de las acciones que éstos realizan dentro del entorno.

Cuando un usuario intenta iniciar una acción, ésta ha de ser verificada antes de poder ser simulada. Por cada acción se comprobará si en el entorno se dan las condiciones necesarias para que pueda ejecutarse y si es así, se ordenará la simulación de un conjunto de consecuencias asociado a la acción.

Este proceso de validación se consigue ampliando la definición de los operadores que intervienen en la fase de planificación, añadiéndoles una descripción de las consecuencias derivadas de su ejecución.

En el sistema se consideran tres tipos de consecuencias:

- Consecuencias de "inventario". Dentro del sistema, a cada alumno se le asocia un inventario de objetos que lleva consigo. Este inventario será actualizado cada vez que se realice una acción de coger o de dejar algún objeto.

- Consecuencias de "animación". El entorno virtual es algo dinámico que reacciona con la ejecución de la mayoría de las acciones. Para aparentar este dinamismo, el sistema ha de animar los objetos y avatares implicados en las acciones, consiguiendo respuestas con credibilidad. Por ejemplo, si un usuario empuja una puerta, el sistema ejecutará la animación correspondiente a mover dicha puerta.
- Consecuencias de simulación. Además de las animaciones visuales que son reacción inmediata a las interacciones del usuario, existen otros efectos que, a pesar de no percibirse, alteran el estado del entorno y que son también consecuencia directa de realizar ciertas acciones. Un ejemplo podría ser el hecho de apretar un cierto botón de un panel de control; esta acción puede tener el efecto de modificar variables del entorno tales como temperatura, presión, etc.

En resumen, durante la simulación de las actividades, el *Agente de Actuación* será responsable de informar sobre las condiciones de ejecución y las consecuencias de las acciones a quien tenga la misión de validar y ejecutar las mismas.

Además, en el transcurso del aprendizaje, el alumno puede necesitar conocer la descripción de los objetos que está viendo, o puede querer solicitar información sobre las posibilidades de interacción con los mismos, es decir, el conjunto de sus operadores asociados. Como es lógico, el *Agente de Actuación* ha de dar respuesta a estos requerimientos.

5.3.1.3. Mecanismos de comunicación: Pizarras

Existen diversas formas de establecer un diálogo entre agentes, una de ellas es el intercambio de mensajes en un lenguaje común que pueda ser interpretado correctamente por ambas partes, como el lenguaje ACL. Sin embargo, este tipo de comunicación tiene restricciones en cuanto que es necesario tener identificado tanto al emisor como al receptor de los mensajes. Además, si el volumen de la información intercambiada en los mensajes es muy alto la latencia en la transmisión puede repercutir negativamente en la eficiencia del sistema.

Hay situaciones en las que un agente que desea compartir cierta información no tiene por qué conocer quién o quiénes son los beneficiarios de dicha información, y viceversa, los agentes receptores pueden no estar interesados en la identidad del emisor. Para lograr este tipo de comunicación se crean zonas comunes de almacenamiento de datos denominadas pizarras.

Una pizarra es un lugar en el que un agente puede publicar sus conocimientos para colaborar con otros en la consecución de sus objetivos.

Este es el mecanismo que utilizan los *Agentes de acción*, tanto en la fase de planificación como posteriormente en la simulación. En concreto, el *Agente de Actuación* lo utiliza para hacer públicas las condiciones de ejecución de acciones y también sus consecuencias, y por otro lado para consultar si las acciones no han sido validadas por otros agentes por no cumplirse los requisitos necesarios para su ejecución y, en tal caso, abortar la simulación de sus efectos.

En un futuro se pretende también que el diálogo entre el *Agente de Planificación* y el *Agente de Actuación* sea a través de una pizarra compartida, liberando al primero de la necesidad de conocer quién es capaz de satisfacer determinados objetivos durante la ejecución de STRIPS.

Un factor importante a la hora de trabajar con pizarras compartidas es la sincronización en su acceso. La pizarra es un recurso común del que diversos agentes leerán y/o escribirán información concurrentemente. Para garantizar la integridad y validez de los datos contenidos en la pizarra es preciso establecer políticas de sincronización en el acceso.

Tres son las operaciones de acceso utilizadas por los agentes que se comunican con el *Agente de Actuación* a través de una pizarra, y cada una lleva asociada una política:

- Anotar datos: Sólo el *Agente de Actuación* es responsable de añadir información a la pizarra, y hasta que éste no notifique a los demás el hecho de que hay nuevos datos en la pizarra, ningún otro tendrá la necesidad de acceder a la misma. Por tanto esta operación no requiere sincronización.

- Leer datos: Cuando el *Agente de Actuación* emite la notificación de que el contenido de la pizarra ha cambiado, cualquier agente podrá acceder a ella para consultarlo. Por tanto, mientras haya agentes leyendo el contenido de la pizarra, ninguna entidad podrá eliminar datos o realizar accesos que afecten a la integridad de los datos anotados.
- Eliminar datos: Varios agentes pueden eliminar datos distintos de la pizarra de forma simultánea. En el transcurso de esta operación ningún otro agente podrá acceder a la pizarra para leer su contenido.

En adelante, se aborda el desarrollo de la parte del sistema multi-agente expuesta en esta introducción, siguiendo para ello, y hasta donde sea posible, la metodología GAIA[2].

5.3.2. Fase de análisis

En esta fase del proceso de desarrollo, en la que únicamente se está desentrañando el problema sin prestar atención a su solución, se pretende identificar los roles o papeles implicados en la consecución de las tareas propuestas para esta parte del sistema multi-agente.

5.3.2.1. Modelo de roles prototipo

Dado que, por lo que se ha comentado antes, el agente o los agentes que se desean desarrollar han de disponer y gestionar un conocimiento muy específico acerca de las acciones básicas que se pueden llevar a cabo dentro del entorno virtual, los roles que han de desempeñar giran alrededor de este conocimiento. En concreto se podrían diferenciar dos roles:

[2] Ver páginas 42 y siguientes.

- ***Informador de acciones***: Su misión será responder a las solicitudes por parte del entorno virtual acerca de las acciones que se pueden realizar sobre un determinado objeto. Además, también deberá contestar a preguntas del estilo de *¿cuál es el efecto de la acción que se acaba de hacer?*, o *¿qué pasa si realizo esta acción?*, realizadas directamente por el usuario, y comunicar su respuesta a los roles de comunicación.

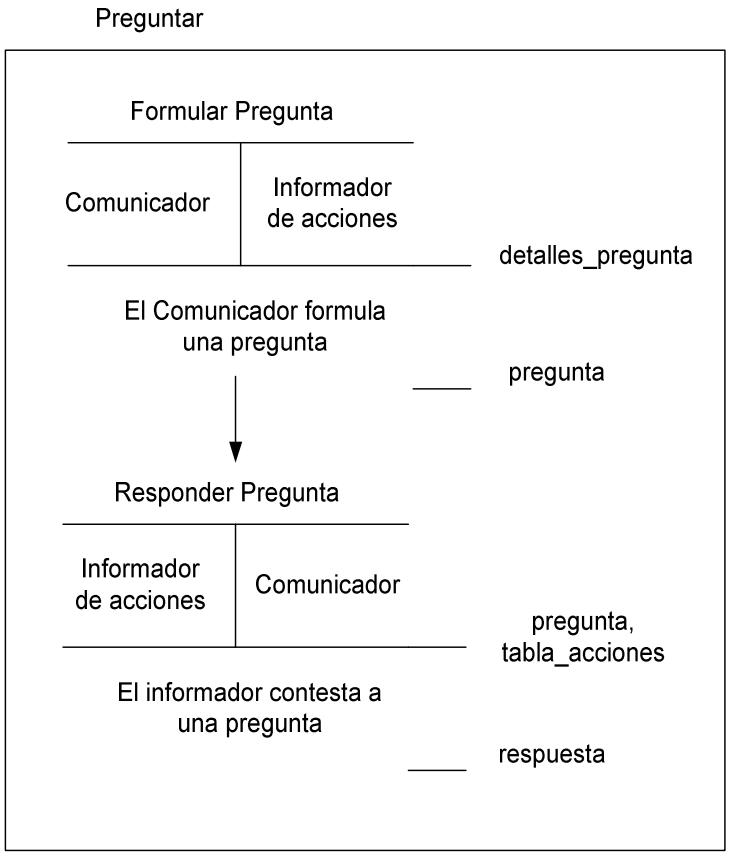
- ***Facilitador de acciones***: Se encarga de proporcionar a los roles que se lo soliciten las acciones pedidas, junto con toda la información asociada a ellas, como pueden ser las condiciones necesarias para que puedan llevarse a cabo y sus consecuencias derivadas. Este rol va a intervenir tanto en los procesos de planificación como en los de simulación, dando origen a las dos facetas mencionadas anteriormente. En concreto, durante la fase de construcción de planes de actuación basados en STRIPS, acepta solicitudes de un rol planificador para proporcionar acciones capaces de satisfacer un determinado objetivo, o para aplicar acciones que modifican el estado actual del problema añadiéndolas al plan. Por otro lado, durante la fase de simulación, en donde se intentan llevar a cabo los planes previamente construidos, aceptará solicitudes, provenientes de roles de comunicación, para que se pongan en conocimiento de aquellos roles responsables de las simulaciones, las precondiciones y las consecuencias de realizar cada acción.

5.3.2.2. Modelo de interacción

Además de los dos roles identificados, en el sistema intervienen algunos otros con los que los primeros mantienen relación, como son el *Comunicador*, el *Planificador*, el *Tutor* y el *Simulador*. Con este modelo se definen los protocolos de comunicación que se ejecutan en estas relaciones.

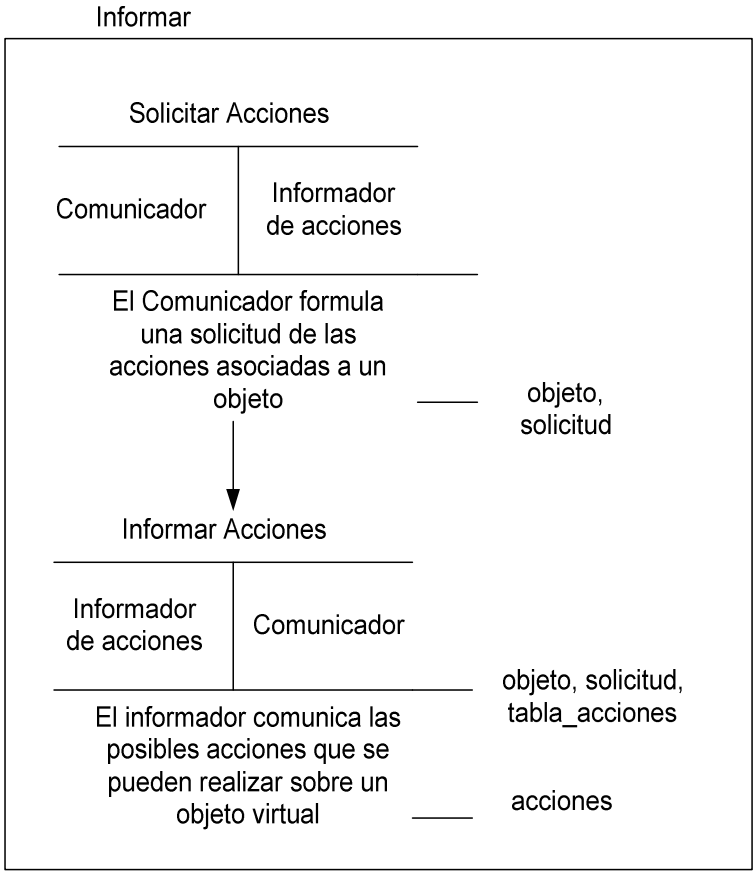
A continuación se muestran los protocolos en los que participa el *Informador de acciones*, a pesar de que ninguno de ellos es iniciado directamente por éste.

1. Preguntar:



Este es un protocolo complejo iniciado por el *Comunicador* cuando el usuario, desde el entorno virtual, lo solicita. Con los detalles de la solicitud, el *Comunicador* elabora una pregunta que es enviada al rol *Informador de acciones*; éste, a su vez, genera una respuesta haciendo uso de su conocimiento acerca de las acciones posibles sobre los objetos del entorno, y la transmite, a través del rol *Comunicador*, al usuario.

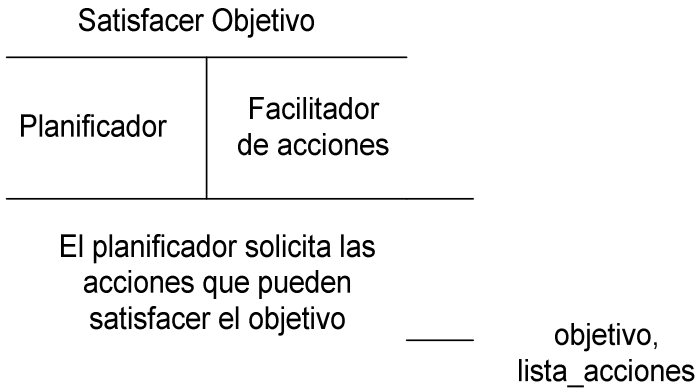
2. Informar:



Este también es un protocolo complejo iniciado por el rol *Comunicador* cuando el usuario, desde el entorno virtual, pide información acerca de lo que se puede hacer con un objeto. El *Comunicador* elabora una solicitud que es enviada al rol *Informador de acciones*, que a su vez, tras consultar su tabla de acciones posibles asociadas a cada objeto, genera una lista y la transmite, vía el rol *Comunicador*, al usuario.

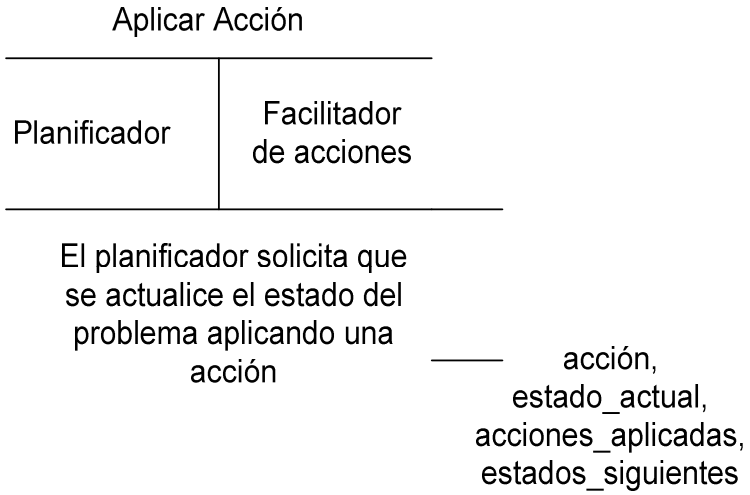
Seguidamente se muestran los protocolos en los que participa el *Facilitador de acciones*, aunque, como antes, ninguno de ellos es iniciado directamente por éste. Algunos se identifican de manera inmediata con procesos que tienen lugar en la construcción de planes según el algoritmo STRIPS.

3. Satisfacer objetivo



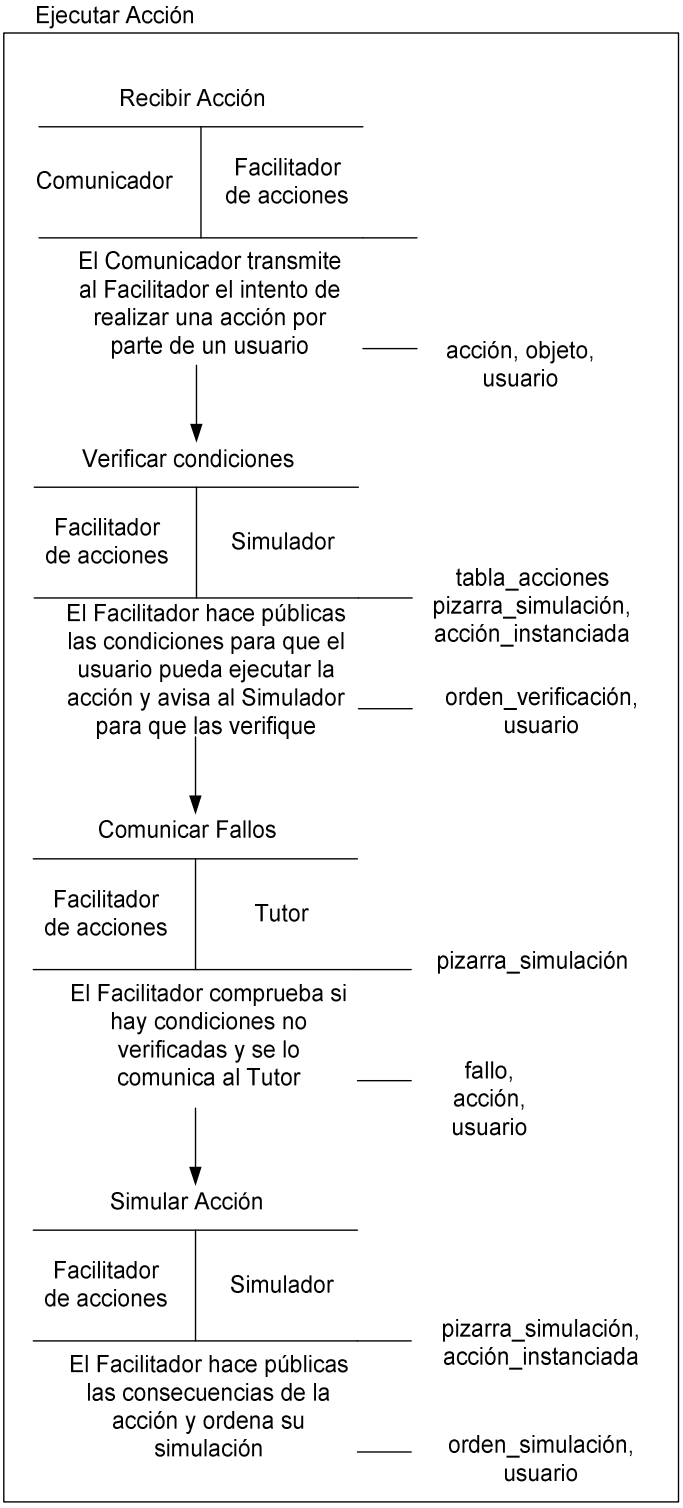
Durante la ejecución del algoritmo STRIPS, el *Planificador* necesitará conocer aquellas acciones, junto con sus precondiciones, que son capaces de hacer cumplir el objetivo que se está revisando en ese momento. Para ello realiza una solicitud al *Facilitador de acciones*, puesto que éste es el que posee ese conocimiento.

4. Aplicar Acción



El algoritmo STRIPS también requiere que, cuando sea necesario añadir una acción al plan que se está construyendo, se actualice el estado actual del problema con el resultado de aplicar dicha acción. Para ello el *Planificador* comunica al *Facilitador de acciones* la acción que en ese momento corresponde aplicar y el estado actual, y como respuesta, éste genera el conjunto de acciones asociadas a los objetos del entorno (puesto que distintos objetos pueden tener asociadas acciones con igual nombre) que coinciden con la acción transmitida, y, ligado a cada una de ellas, el estado siguiente resultado de aplicarla sobre el estado actual.

5. Ejecutar Acción



Este protocolo interviene durante la fase de simulación y es un poco más complejo que los anteriores. En este caso el iniciador es el *Comunicador*, al informar al *Facilitador* del intento de ejecución de una acción sobre un objeto por parte de un usuario. El *Facilitador*, entonces, identifica las precondiciones que han de cumplirse para poder realizar dicha acción sobre el objeto, y las publica en una pizarra común, avisando de ello al *Simulador*. El *Facilitador* comprobará después, a través de la pizarra, si todas las condiciones anotadas en ella han sido verificadas por el *Simulador*, y de no ser así informará de ello al *Tutor* para que quede constancia de que la acción no pudo realizarse. En caso de que todo haya sido verificado correctamente, el *Facilitador* publicará en la pizarra las consecuencias de realizar la acción y enviará al *Simulador* la orden para que efectúe la simulación.

5.3.2.3. Modelo de roles completamente elaborado

- ***Informador de acciones:***

Esquema de rol: Informador de acciones

Descripción:

Informa de las acciones que se pueden llevar a cabo sobre un objeto presente en el entorno virtual, y responde a las preguntas ¿cuál es el efecto de la acción que se acaba de hacer? y ¿qué pasa si realizo esta acción? formuladas por un usuario.

Protocolos y actividades:

FormularPregunta, ResponderPregunta, SolicitarAcciones, InformarAcciones, ElaborarRespuesta, ElaborarListaAcciones.

Permisos:

Lee:

(proporcionado) pregunta, (proporcionado) objeto,

(proporcionado) solicitud , tabla_acciones

Modifica:

Genera:

respuesta, acciones

Responsabilidades:

Vitalidad:

Informador de acciones =
(FormularPregunta.ElaborarRespuesta.ResponderPregunta)* ||
(SolicitarAcciones.ElaborarListaAcciones.InformarAcciones)*

Seguridad:

El recurso "objeto" ha de tener su correspondencia geométrica en el entorno virtual.

Aclaraciones:

1. Los protocolos asociados a **Preguntar** e **Informar** se ejecutan de forma coetánea un número indefinido de veces.

2. La actividad **ElaborarRespuesta** trata de instanciar, mediante la tabla de acciones de que dispone, una cierta acción a partir de la información contenida en la pregunta, y a partir de ella extraer sus consecuencias derivadas, las cuales constituyen la respuesta.
3. Por otro lado, la actividad **ElaborarListaAcciones** prepara el conjunto de acciones, junto con la información de sus parámetros respectivos, asociado al objeto para el que se solicita.
4. Todos los recursos "objeto" que intervienen en los protocolos especificados han de tener su correspondencia geométrica en el entorno virtual, para asegurar que se pueda responder a las preguntas sobre ellos o que exista un conjunto de acciones asociadas.

- ***Facilitador de acciones:***

Esquema de rol: Facilitador de acciones

Descripción:

Durante la fase de planificación informa de las acciones capaces de satisfacer un cierto objetivo y ayuda en la construcción de planes aplicando acciones sobre el estado del problema en cada momento, para modificarlo.

Durante la fase de simulación proporciona las condiciones de ejecución de las acciones y sus consecuencias a los roles de simulación.

Protocolos y actividades:

SatisfacerObjetivo, ElaborarListaAcciones, AplicarAccion, RecibirAccion, VerificarCondiciones, ComunicarFallos, SimularAccion, InstanciarAccion.

Permisos:

Lee:

(proporcionado)objetivo, (proporcionado)acción, (proporcionado) estado_actual, (proporcionado)objeto, (proporcionado)usuario, pizarra_simulacion, tabla_acciones

Modifica:

pizarra_simulacion

Genera:

lista_acciones, acciones_aplicadas, estados_siguientes, acción_instanciada, orden_verificación, orden_simulación, fallo

Responsabilidades:

Vitalidad:

Facilitador de acciones = (Planificar)*.(EjecutarAccion)*

Planificar = SatisfacerObjetivo.ElaborarListaAcciones | AplicarAccion

EjecutarAccion = RecibirAccion.InstanciarAccion.VerificarCondiciones.(ComunicarFallos | SimularAccion)

Seguridad:

El recurso proporcionado "acción" ha de tener su homónimo en "tabla_acciones".

El acceso al recurso "pizarra_simulacion" ha de efectuarse en exclusión mútua.

El recurso "pizarra_simulacion" no contendrá anotaciones antes de la ejecución de los protocolos VerificarCondiciones y SimularAccion

Aclaraciones:

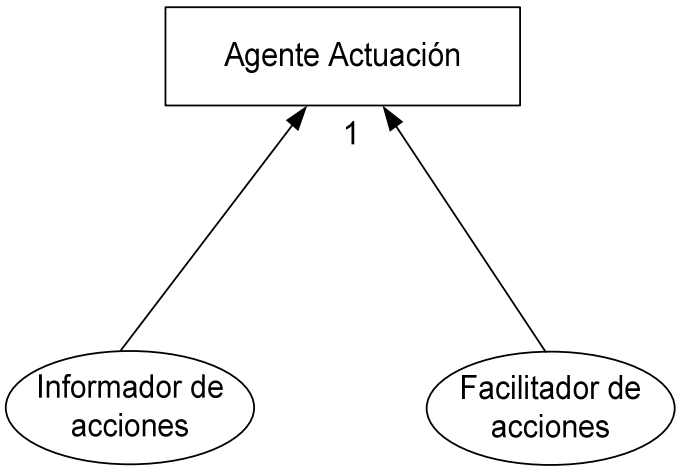
1. La actividad **InstanciarAccion** se refiere a la operación de instanciar, mediante el patrón dado por el recurso "acción", y utilizando el recurso "tabla_acciones", las precondiciones, postcondiciones y consecuencias de cada acción.
2. El recurso "pizarra_simulación" deberá ser accedido en exclusión mutua para garantizar la integridad de los datos allí anotados, ya que son varios los roles que comparten el acceso además del rol *Facilitador de acciones*.
3. Por último, para asegurar que en el recurso "pizarra_simulación" no se mezcla información proveniente de los protocolos **VerificarCondiciones** y **SimularAccion** se especifica la responsabilidad de seguridad de que dicho recurso esté vacío en los instantes especificados.

5.3.3. Fase de diseño de alto nivel

En esta fase del diseño, el nivel de abstracción disminuye para introducir la definición de los distintos tipos de agentes entre los que se reparten los roles identificados.

5.3.3.1. Modelo de agentes

El siguiente diagrama ilustra la decisión tomada en cuanto a la asignación de roles a tipos de agente:



Esta solución reduce la carga del sistema al minimizar el número de agentes, puesto que solo habrá una instancia de *Agente de Actuación* en el sistema que encarnará los dos roles. La desventaja de disponer de un solo tipo de agente es que si éste falla perderemos la funcionalidad asociada a los dos roles.

La alternativa más robusta para el sistema es crear un tipo de agente para cada rol. De esta manera evitamos el inconveniente anterior, pero analizando en detalle la situación en el ámbito del sistema completo, vemos que en realidad no tiene sentido disponer de uno solo de los dos roles si falla el otro, puesto que de nada sirve construir planes de actuación si no se ofrece al usuario la posibilidad de intentar ejecutarlos al no disponer éste de información sobre qué es lo que se puede hacer con los objetos en el entorno virtual. Y viceversa, no resulta útil para el aprendizaje informar al usuario de qué acciones puede efectuar sobre los objetos si dichas acciones no pueden ser contrastadas con un plan y ni siquiera se pueden simular sus efectos.

5.3.3.2. Modelo de servicios

La tabla siguiente muestra los servicios asociados a cada rol, entendiendo por servicio un bloque de actividad del que el agente se ocupará. La metodología GAIA especifica que todas las actividades identificadas en el modelo de roles completamente elaborado, y en concreto en las responsabilidades de vitalidad de cada rol, deberían aparecer como servicios. Sin embargo, no todos los servicios definidos aquí tienen por qué corresponderse con actividades.

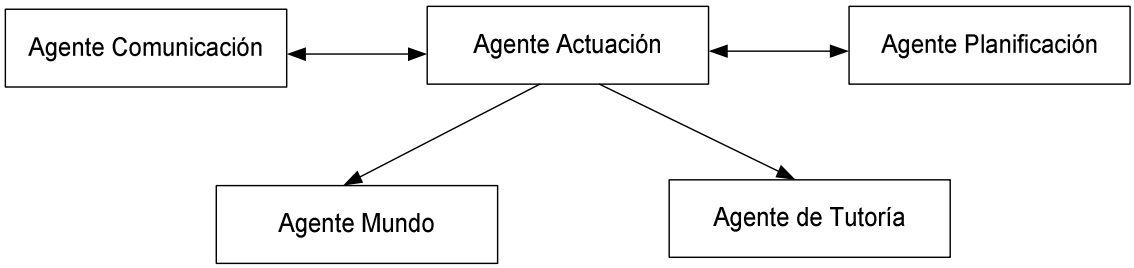
En la tabla vemos, por ejemplo, que no aparecen protocolos como *RecibirAcción*, *FormularPregunta*, *ResponderPregunta*, *SolicitarAcciones* o *InformarAcciones*; estos no se consideran servicios puesto que no constituyen funciones que el agente ofrezca. Más bien, estas funciones están reflejadas en actividades como *ElaborarRespuesta* o *ElaborarListaAcciones*.

De igual manera, observamos que tampoco aparece el protocolo *SatisfacerObjetivo*, puesto que la actividad que realmente encierra dicho protocolo es *ElaborarListaAcciones*.

Servicio	Entradas	Salidas	Pre-condiciones	Post-condiciones
ElaborarRespuesta	*Pregunta, tabla_acciones*	*respuesta*	-	(pregunta no valida ∧ respuesta = nil) ∨ (pregunta valida ∧ respuesta valida)
ElaborarListaAcciones	*objeto, solicitud, tabla_acciones*	*acciones*	∃objeto	acciones no vacía
AplicarAccion	*acción, estado_actual*	*acciones_aplicadas, estados_siguientes*	∃acción	(acciones_aplicadas no vacía) ∧ (estados_siguientes no vacío)
InstanciarAccion	*accion, objeto, usuario, tabla_acciones*	*acción_instanciada*	∃acción	acción_instanciada ≠ nil
VerificarCondiciones	*pizarra_simulación, acción_instanciada*	*orden_verificación, usuario*	pizarra _ simulación vacía	El simulador sabe que hay condiciones que verificar
ComunicarFallos	*pizarra_simulación*	*fallo*	-	El tutor sabe que no se puede realizar la acción
SimularAccion	*pizarra_simulación, acción_instanciada*	*orden_simulación, usuario*	pizarra _ simulación vacía	El simulador sabe que hay acciones que simular

5.3.3.3. Modelo de conocidos

El siguiente esquema representa el grafo de relaciones que mantiene el *Agente de Actuación* con otros agentes del sistema. Si existe una relación entre dos agentes, significa que, en un momento determinado, se establece un canal de comunicación entre ellos para el intercambio de información.



5.3.4. Fase de diseño detallado

Tanto en la fase de análisis del sistema como en el diseño de alto nivel del mismo se ha seguido la metodología GAIA. Pero a partir de esta fase del desarrollo en la que hay que realizar un diseño más detallado que dé paso, de la forma más inmediata posible, a la implementación, GAIA no establece heurísticas específicas. En nuestro caso seguiremos una serie de pasos [Moraïtis et al., 2003] que nos irán acercando poco a poco a dicha implementación, teniendo siempre presente que ésta se realizará usando la plataforma de desarrollo de agentes JADE.

5.3.4.1. Definición de mensajes ACL

Los siguientes esquemas definen el conjunto de mensajes ACL en los que interviene el *Agente de Actuación*, ya sea como emisor o como receptor. Su especificación ha sido extraída del modelo de interacción y refleja, además, el protocolo implicado, el tipo de mensaje según el estándar FIPA, y el contenido del mismo.

Para la definición de estos mensajes hay que tener en cuenta que un protocolo no tiene por qué dar lugar a una pareja de mensajes de petición y respuesta, sino que puede originar muchos más mensajes, como el protocolo *EjecutarAcción*, algunos de ellos sin respuesta.

Mensaje ACL: FormularPregunta **Emisor:** Agente de Comunicación **Receptor:** Agente de Actuación **Performativa FIPA:** REQUEST **Protocolo:** Preguntar **Ontología:** OntMAEVIF **Contenido:** concepto: Pregunta	**Mensaje ACL:** ResponderPregunta **Emisor:** Agente de Actuación **Receptor:** Agente de Comunicación **Performativa FIPA:** INFORM **Protocolo:** Preguntar **Ontología:** OntMAEVIF **Contenido:** concepto: Respuesta

Mensaje ACL: SolicitarAcciones **Emisor:** Agente de Comunicación **Receptor:** Agente de Actuación **Performativa FIPA:** REQUEST **Protocolo:** Informar **Ontología:** OntMAEVIF **Contenido:** concepto: solicitud, objeto	**Mensaje ACL:** InformarAcciones **Emisor:** Agente de Actuación **Receptor:** Agente de Comunicación **Performativa FIPA:** INFORM **Protocolo:** Informar **Ontología:** OntMAEVIF **Contenido:** concepto: acciones

Mensaje ACL: ComunicarObjetivo **Emisor:** Agente de Planificación **Receptor:** Agente de Actuación **Performativa FIPA:** REQUEST **Protocolo:** Satisfacer Objetivo **Ontología:** OntMAEVIF **Contenido:** concepto: objetivo	**Mensaje ACL:** SatisfacerObjetivo **Emisor:** Agente de Actuación **Receptor:** Agente de Planificación **Performativa FIPA:** INFORM **Protocolo:** Satisfacer Objetivo **Ontología:** OntMAEVIF **Contenido:** concepto: lista _acciones

Mensaje ACL: ComunicarAcciónPlan **Emisor:** Agente de Planificación **Receptor:** Agente de Actuación **Performativa FIPA:** REQUEST **Protocolo:** Aplicar Acción **Ontología:** OntMAEVIF **Contenido:** concepto: acción, estado_actual	**Mensaje ACL:** AplicarAcciones **Emisor:** Agente de Actuación **Receptor:** Agente de Planificación **Performativa FIPA:** INFORM **Protocolo:** Aplicar Acción **Ontología:** OntMAEVIF **Contenido:** concepto: acciones_aplicados, estados_siguientes

Mensaje ACL: ComunicarAccion **Emisor:** Agente de Comunicación **Receptor:** Agente de Actuación **Performativa FIPA:** INFORM **Protocolo:** Ejecutar Acción **Ontología:** OntMAEVIF **Contenido:** concepto: accion,objeto,usuario

Mensaje ACL: OrdenarVerificación **Emisor:** Agente de Actuación **Receptor:** Agente Mundo **Performativa FIPA:** INFORM **Protocolo:** Ejecutar Acción **Ontología:** OntMAEVIF **Contenido:** concepto: orden_verificación, usuario

Mensaje ACL: ComunicarFallos **Emisor:** Agente de Actuación **Receptor:** Agente de Tutoría **Performativa FIPA:** INFORM **Protocolo:** Ejecutar Acción **Ontología:** OntMAEVIF **Contenido:** concepto: fallo, acción, usuario

Mensaje ACL: OrdenarSimulacion **Emisor:** Agente de Actuación **Receptor:** Agente Mundo **Performativa FIPA:** INFORM **Protocolo:** Ejecutar Acción **Ontología:** OntMAEVIF **Contenido:** acción: orden_simulacion, usuario	

5.3.4.2. Diagrama de clases de diseño

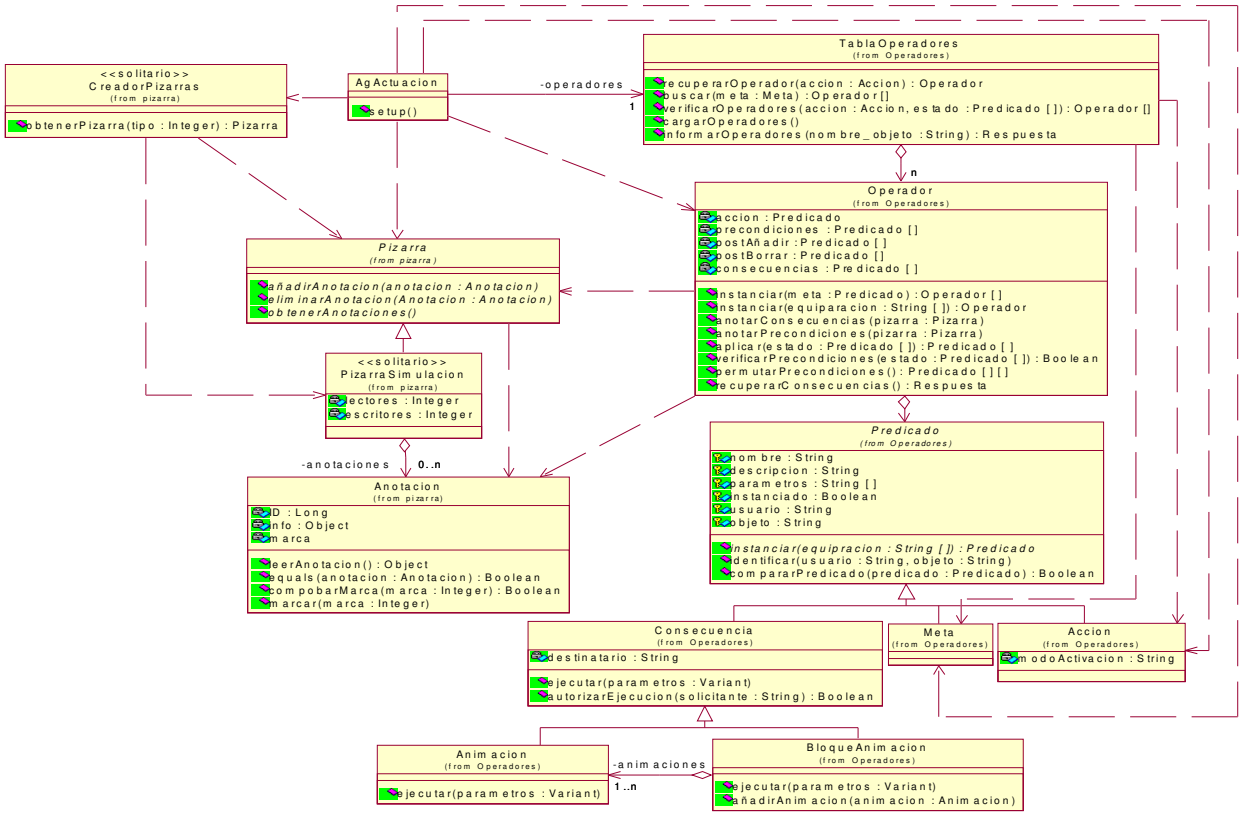


Figura 55: Diagrama de clases de diseño para el *Agente de Actuación*

DEFINICIÓN DE CLASES, ATRIBUTOS Y MÉTODOS:

Las definiciones que a continuación se presentan son puramente descriptivas y únicamente reflejan las características más relevantes del diagrama de clases.

CLASE	DESCRIPCIÓN
AgActuacion	Clase que define los comportamientos del Agente de Actuación.
TablaOperadores	Colección de operadores clasificados según el objeto virtual al que pueden ser aplicados.
Operador	Clase que define un operador STRIPS con algunas ampliaciones.
Predicado	Clase abstracta que representa una fórmula compuesta por un nombre de predicado y una lista de argumentos.
Consecuencia	Predicado que define el efecto que provoca en la simulación el hecho de ejecutar un operador.
Meta	Predicado que representa un objetivo a alcanzar dentro del proceso de cálculo de planes.
Accion	Predicado que representa una acción llevada a cabo en el entorno virtual.
Animación	Consecuencia que define las animaciones que han de ejecutarse en el entorno ante una acción realizada.
BloqueAnimacion	Conjunto de animaciones cuya ejecución ha de simultanearse.
CreadorPizarras	Clase encargada de crear instancias de distintos tipos de pizarras para compartir información.

CLASE	DESCRIPCIÓN
Pizarra	Clase abstracta que define una interfaz pública genérica para el acceso a los distintos tipos de pizarras.
PizarraSimulacion	Clase que encapsula un recurso compartido para el intercambio de información entre agentes durante la fase de simulación del sistema.
Anotacion	Representa a cada uno de los datos individuales que pueden estar contenidos en una pizarra.

CLASE			DESCRIPCIÓN
AgActuacion	**ATRIBUTOS**	operadores	Tabla en la que se almacena la definición de operadores para todos los objetos virtuales.
	MÉTODOS	setup()	Inicialización del agente y definición de comportamientos.
TablaOperadores	**ATRIBUTOS**		
	MÉTODOS	buscar(in: meta) : Operadores	Proporciona todos los operadores instanciados de todas las maneras posibles que son capaces de añadir una meta al estado actual.
		recuperarOperador (in: accion): Operador	Proporciona el operador instanciado a partir de una acción.

CLASE			DESCRIPCIÓN
		verificarOperadores (in: accion,in: estado) : Operadores	Proporciona el conjunto de operadores instanciados a partir de una acción y cuyas precondiciones son ciertas en el estado.
		cargarOperadores()	Inicializa la tabla de operadores a partir de un fichero de definición de operadores.
		informarOperadores (in: objeto): respuesta	Transmite al agente destinatario la información sobre el conjunto de operadores asociados a un objeto.
Operador	**ATRIBUTOS**	accion	Predicado de la clase Acción formado por el nombre y los parámetros de un operador.
		precondiciones	Conjunto de predicados de la clase Meta que han de cumplirse para ejecutar el operador.
		postAñadir	Conjunto de predicados de la clase Meta que se cumplen tras haber ejecutado el operador
		postBorrar	Conjunto de predicados de la clase Meta que dejan de ser ciertos tras haber ejecutado el operador
		Consecuencias	Conjunto de predicados de la clase Consecuencia que definen los efectos de la ejecución del operador en el sistema.

CLASE			DESCRIPCIÓN
	MÉTODOS	instanciar(in: meta) : Operadores	Comprueba si la meta está entre sus postcondiciones a añadir y en ese caso devuelve todas sus posibles instanciaciones.
		instanciar (in:equiparación) : Operador	Devuelve su instanciación a partir de un conjunto de variables instanciadas correspondientes a sus parámetros.
		anotarPrecondiciones (in: pizarra)	Anota en una pizarra el conjunto de las precondiciones del operador.
		anotarConsecuencias (in: pizarra)	Anota en una pizarra el conjunto de las consecuencias del operador.
		aplicar(in: estado) :estado	Proporciona el estado siguiente que resultaría de la ejecución de un operador sobre un estado de partida.
		verificarPrecondiciones (in: estado) : boolean	Informa sobre si las precondiciones del operador se cumplen en un determinado estado.
		permutarPrecondiciones() : permutaciones	Proporciona una lista de todas las permutaciones de las precondiciones del operador.
		recuperarConsecuencias() : respuesta	Informa sobre las consecuencias de la ejecución de un operador al agente destinatario.

CLASE			DESCRIPCIÓN
Predicado	**ATRIBUTOS**	nombre	Nombre del predicado.
		descripción	Descripción del predicado.
		parámetros	Argumentos del predicado.
		instanciado	Indicador de si está instanciado o no.
		usuario	Usuario que hace uso del predicado.
		objeto	Objeto virtual sobre el que actúa el predicado.
	MÉTODOS	instanciar (in:equiparación) : predicado	Método abstracto que instancia un predicado a partir de una equiparación de todos sus parámetros.
		identificar(in: usuario, in: objeto)	Asocia el predicado a un usuario y un objeto del entorno.
		compararPredicado (in: predicado) : boolean	Indica si dos predicados son iguales en nombre y argumentos.
Consecuencia	**ATRIBUTOS**	destinatario	Agente responsable de hacer efectiva la consecuencia.
	MÉTODOS	ejecutar(in: parámetros)	Ordena al sistema la ejecución de la consecuencia.
		autorizarEjecucion (in: solicitante) : boolean	Informa al solicitante sobre si es su responsabilidad hacer efectiva la consecuencia.

CLASE			DESCRIPCIÓN
Accion	**ATRIBUTOS**	modoActivacion	La acción puede ser llevada a cabo mediante su selección o por colisión con un objeto del entorno.
	MÉTODOS		
Animación	**ATRIBUTOS**		
	MÉTODOS	ejecutar(in: parámetros)	Transmite la orden de ejecución de una animación al sistema de comunicación con el entorno gráfico.
BloqueAnimacion	**ATRIBUTOS**	animaciones	Conjunto de animaciones cuya ejecución ha de realizarse simultáneamente.
	MÉTODOS	ejecutar(in: parámetros)	Transmite la orden de ejecución de un bloque de animaciones al sistema de comunicación con el entorno gráfico.
		añadirAnimacion (in: animación)	Añade una animación al conjunto.
CreadorPizarras	**ATRIBUTOS**		
	MÉTODOS	obtenerPizarra (in:tipo): pizarra	Proporciona una instancia de un tipo de pizarra.
Pizarra	**ATRIBUTOS**		
	MÉTODOS	añadirAnotacion (in: anotacion)	Escribe una información determinada en la pizarra.

CLASE			DESCRIPCIÓN
		eliminarAnotacion (in: anotacion)	Borra una información determinada de la pizarra
		obtenerAnotaciones() : anotaciones	Proporciona el conjunto de todos los datos anotados en la pizarra.
PizarraSimulacion	**ATRIBUTOS**	lectores	Atributo de sincronización que contabiliza el número de agentes que están leyendo la pizarra en un instante determinado.
		escritores	Atributo de sincronización que contabiliza el número de agentes que están escribiendo en la pizarra en un instante determinado.
		anotaciones	Conjunto de anotaciones escritas en la pizarra
	MÉTODOS		
Anotacion	**ATRIBUTOS**	ID	Identificador único de la anotación en la pizarra.
		info	Información que se anota en la pizarra.
		marca	Indica el estado de verificación de la anotación en procesos en los que la anotación ha de ser comprobada por algún agente.
	MÉTODOS	leerAnotacion(): info	Proporciona la información asociada con la anotación.

CLASE			DESCRIPCIÓN
		equals(in: anotacion)	Comprueba si dos anotaciones son iguales en base a su ID.
		comprobarMarca (in: marca) : boolean	Comprueba si la marca de la anotación coincide con la pasada como argumento.
		marcar(in: marca)	Establece la marca de la anotación al valor pasado como argumento (verificada o no verificada).

OBSERVACIONES:

El diagrama muestra la organización de clases para las dos estructuras de información con las que trabaja el *Agente de Actuación*: la pizarra compartida y los operadores.

El primer cometido del *Agente de Actuación*, nada más comenzar su ciclo de vida, es construir su estructura de conocimiento desde algún dispositivo de almacenamiento externo. Esta estructura es, básicamente, una tabla indexada por nombres de objetos presentes en el entorno virtual, a cada uno de los cuales se le ha asociado un conjunto de operadores genéricos no instanciados. La responsabilidad de la gestión de esta tabla reside en la clase "TablaOperadores", que contiene métodos para rellenar la tabla, para realizar búsquedas de operadores según distintos criterios, y para comunicar los operadores asociados a un objeto según el protocolo "Informar".

El núcleo del contenido de la tabla de operadores es la clase "Operador". Esta clase se ajusta a la definición de operador dada anteriormente y se apoya en una estructura jerárquica, cuya raíz es la clase "Predicado", para definir sus precondiciones, postcondiciones y consecuencias. La clase "Operador" contiene métodos para instanciar el operador a partir de la información de equiparación proporcionada, para publicar sus precondiciones y consecuencias en una pizarra compartida, para aplicar el operador y modificar el estado del problema durante la fase de planificación, para realizar las distintas ordenaciones de sus precondiciones cuando el *Agente de planificación* necesite expandir su árbol de búsqueda en STRIPS, y finalmente, para enviar el conjunto de sus consecuencias como respuesta a las preguntas formuladas según el protocolo "Preguntar". Como apunte extra al cuaderno de trabajo futuro, cabría hacer una reflexión sobre las ventajas de dividir esta clase en dos para separar los conceptos y la funcionalidad asociada a un operador instanciado y a uno no instanciado.

La clase "Predicado" es una abstracción del concepto lógico de predicado compuesto por un nombre y una lista de parámetros. Una instancia de "Predicado" representa una acción llevada a cabo por un usuario, o una acción planificada, o una meta por cumplir, o bien una consecuencia de la ejecución de un operador. Todas estas concreciones tienen sus respectivas clases en la jerarquía definida a partir de "Predicado". Los predicados, al igual que los operadores o, mejor dicho, como parte de ellos, pueden ser instanciados; adicionalmente poseen una descripción de sí mismos útil para responder preguntas formuladas por el usuario, y por último, pueden ser asociados a un nombre de objeto y a un usuario en particular como iniciador de la acción o destinatario de la consecuencia, aunque, como futura mejora, se propone eliminar esto último, haciendo que sean los destinatarios de las consecuencias los que identifiquen las consecuencias que deben ejecutar por el tipo de las mismas.

Las clases "Accion", "Meta" y "Consecuencia", particularizan y a la vez extienden la definición de "Predicado".

La clase "Acción", por ejemplo, añade información sobre el modo en que un usuario puede solicitar la ejecución de dicha acción, bien por el simple hecho de tocar un objeto, bien mediante la selección entre una lista de acciones posibles.

La clase "Meta", aunque funcionalmente no aporta nada relevante a la definición de predicado y su comportamiento es similar al de la clase "Acción", representa un concepto que maneja el sistema y que ha de ser diferenciado de los conceptos de acción y consecuencia.

Por su parte, la clase "Consecuencia" añade funcionalidad para ordenar la ejecución de los efectos asociados a la realización de una acción, y a su vez establece una nueva jerarquía al ejercer de clase base para las clases "Animación" y "BloqueAnimación". Estas dos clases particularizan el tipo de consecuencias denominadas de animación, ya sean éstas simples o compuestas por una serie de animaciones que el sistema ha de efectuar simultáneamente.

La razón de esta nueva jerarquía es la de conseguir polimorfismo en el tratamiento de las consecuencias. Ya que éstas deberán ser procesadas por agentes ajenos a sus detalles, se trata de permitirles ejecutarlas sin necesidad de que sean conscientes del tipo de consecuencia que realmente están procesando.

Esta filosofía se refleja también en otras partes del diseño, como la jerarquía de predicados o la estructura de la pizarra de la que hablaremos enseguida. El propósito general es reducir el acoplamiento y la cantidad de dependencias de entidades ajenas a estas estructuras pero que tengan que hacer uso de ellas. Además, se ha intentado dotar al diseño de la característica de modificabilidad, haciendo que las clases sean autocontenidas y respeten, en la medida de lo posible, el *"principio de ocultación de la información"* [Parnas, D.L. 1971] mejorando al mismo tiempo la cohesión de cada clase. Este principio se hace efectivo en procesos como el de instanciar un operador o ejecutar determinada consecuencia. Desde el exterior de estas clases no se precisa conocer sus características intrínsecas, pues son ellas mismas las únicas encargadas de realizar estas tareas complejas y gestionar sus propios datos. Si en algún momento se deseara modificar su estructura interna, este cambio solo afectaría a los algoritmos contenidos en ellas y no al resto de clases que los utilizan, para las que el cambio sería transparente.

Todo esto repercute en un aumento de complejidad en el diseño pero es perfectamente compensado por los beneficios en modificabilidad.

Por último, en cuanto a la pizarra, se puede observar que se ha empleado un patrón de diseño de factoría o "creador", combinado con el patrón "solitario". La pizarra es un recurso compartido cuyo ámbito ha de ser global a los distintos agentes que accedan a ella; además, por el hecho de ser compartida, en el sistema sólo habrá una única instancia de la misma, de ahí el patrón "solitario". En lo referente al patrón "creador", el objetivo es aislar a los agentes que accedan a la pizarra de los detalles de creación de la misma, dando esta responsabilidad a una clase intermediaria llamada "CreadorPizarras". Este patrón es útil también a la hora de usar la pizarra. Los agentes que quieran acceder a ella lo harán a un nivel de abstracción alto a través de una interfaz común declarada en la clase abstracta "Pizarra". Esta clase permite implementar distintos tipos de pizarra con distintas políticas de acceso a ellas, y al mismo tiempo ocultar al exterior sus detalles. Se puede observar que el atributo "anotaciones", que representa la estructura de datos en la que la pizarra almacena la información, reside en la clase "PizarraSimulacion", que es una subclase de "Pizarra"; esto es debido a que distintas implementaciones de una pizarra pueden requerir que el almacenamiento de la información se estructure de una manera diferente, es decir, una instancia de la clase "PizarraSimulacion" puede almacenar su información en una estructura de tipo lista, mientras que otro tipo de pizarra puede necesitar un árbol o una tabla.

5.3.4.3. Definición de los comportamientos JADE

En el modelo de agente especificado por la plataforma JADE cada comportamiento corresponde a un objeto interno al agente, que se ejecuta como un hilo concurrente con los demás comportamientos y cuya inicialización se realiza en el método "setup" del propio agente. La plataforma es responsable de la planificación de todos los comportamientos.

La definición de comportamientos es extraida directamente de las responsabilidades de vitalidad de cada rol asociado al agente.

El *Agente de Actuación* presenta tres comportamientos cíclicos y concurrentes cuya duración está ligada a la vida del agente y que han sido extraídos del modelo de roles:

- Comportamiento "Informar": Responsable de llevar a cabo los protocolos "Preguntar" e "Informar", y las actividades, especificadas en el rol *Informador de acciones*, "ElaborarRespuesta" y "ElaborarListaAcciones".
- Comportamiento "Planificar": Responsable de los protocolos "SatisfacerObjetivo" y "AplicarAcción".
- Comportamiento "EjecutarAcción": Este comportamiento es más complejo, tiene como misión realizar el protocolo del mismo nombre y para ello define dos subcomportamientos o etapas que se ejecutan en secuencia y son:
    - Comportamiento "RecibirAcción": ejecuta el protocolo homónimo y realiza la actividad "InstanciarAcción" especificada por el rol *Facilitador de acciones*.
    - Comportamiento "VerificarAcción": lleva a cabo el protocolo "Verificar condiciones" además de los protocolos "Comunicar Fallos" y "Simular Acción".

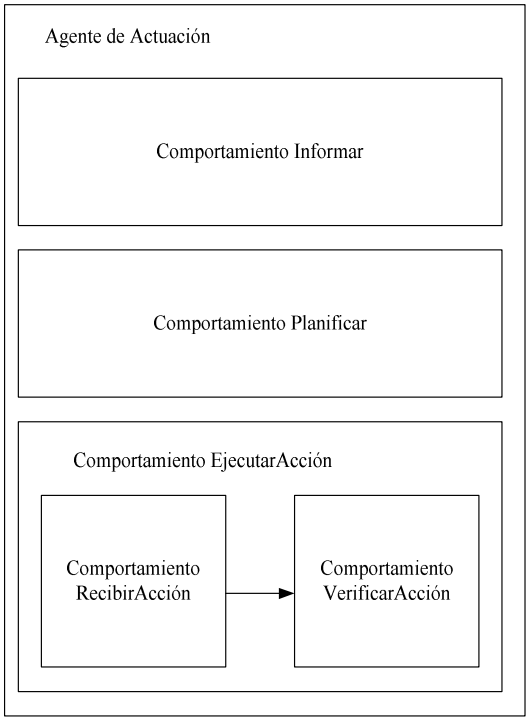


Figura 56: Esquema de comportamientos del Agente de Actuación

A continuación se detallan los diagramas de interacción de bajo nivel correspondientes a los protocolos ejecutados en cada comportamiento:

Protocolo Informar:

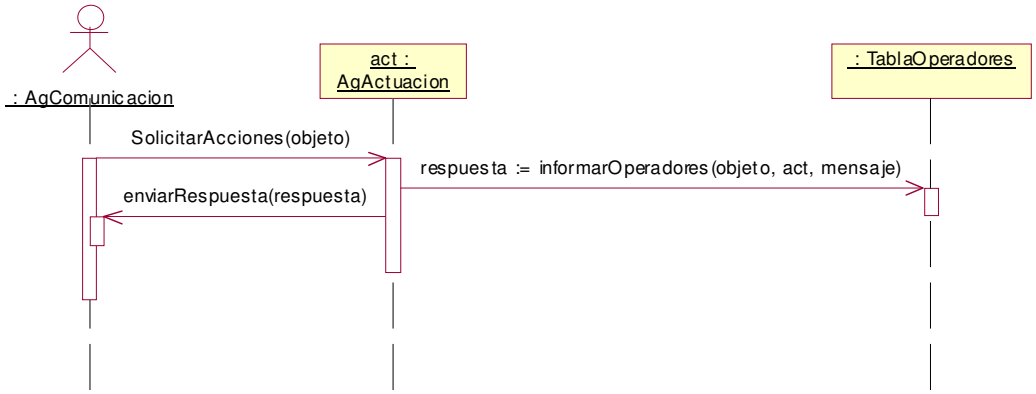


Figura 57: Protocolo *Informar*

Protocolo Preguntar:

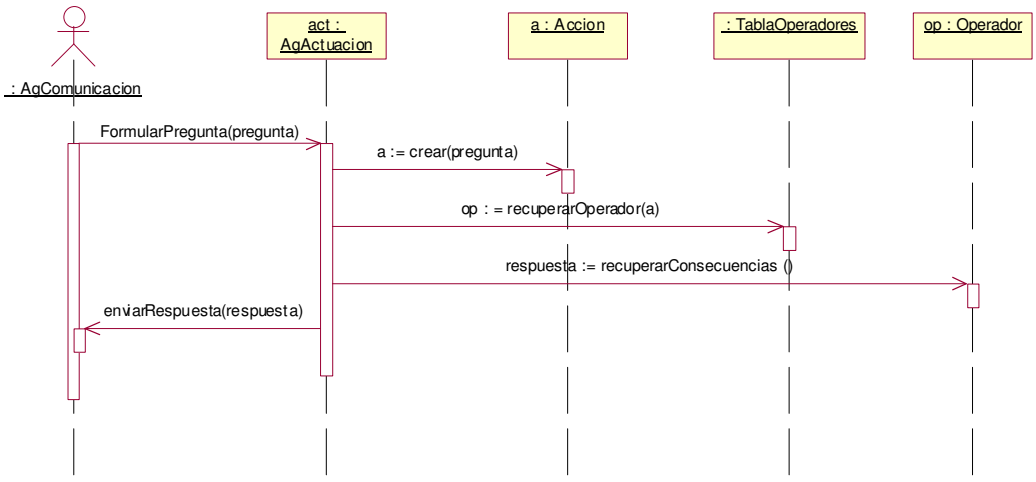


Figura 58: Protocolo *Preguntar*

Protocolo Satisfacer Objetivo:

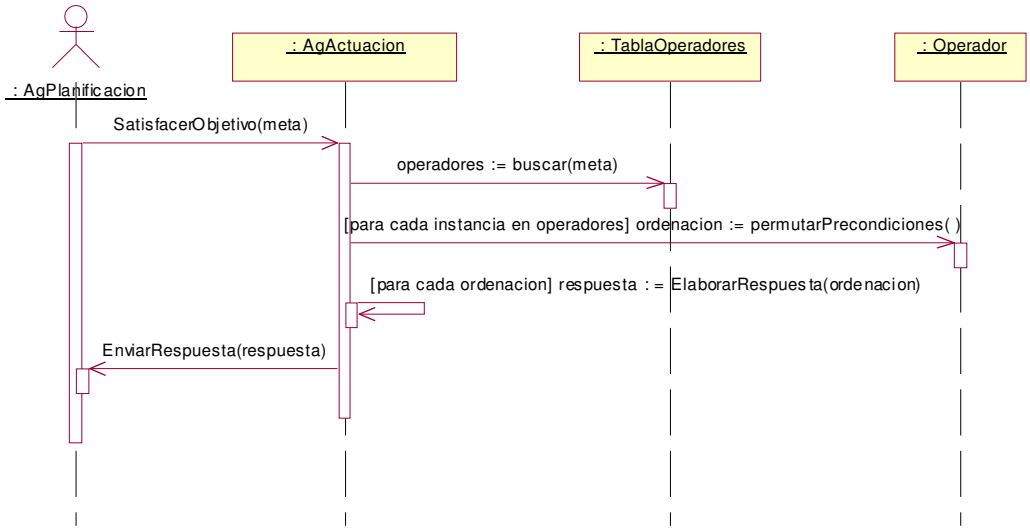


Figura 59: Protocolo *Satisfacer Objetivo*

Protocolo Aplicar Acción:

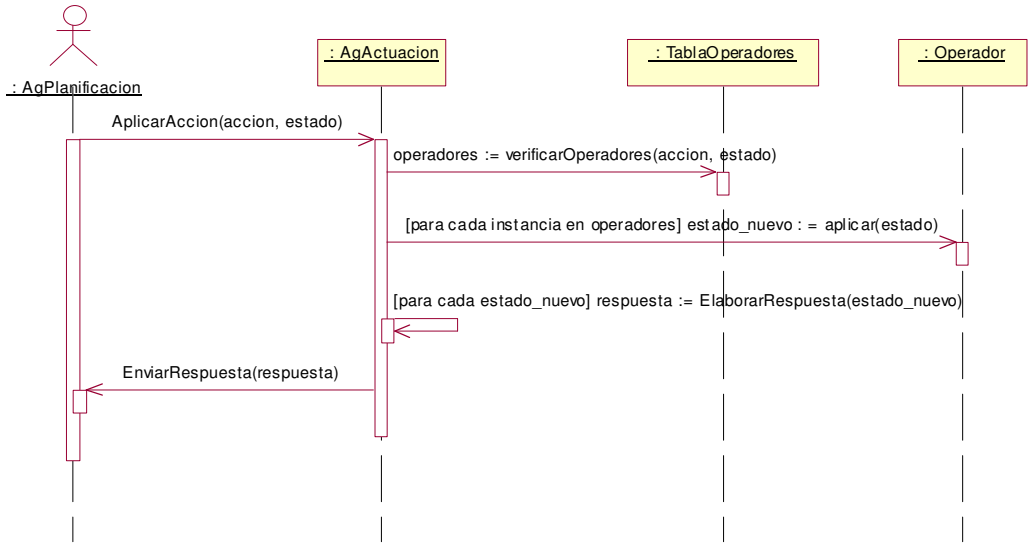


Figura 60: Protocolo *Aplicar Acción*

Protocolo Ejecutar Acción:

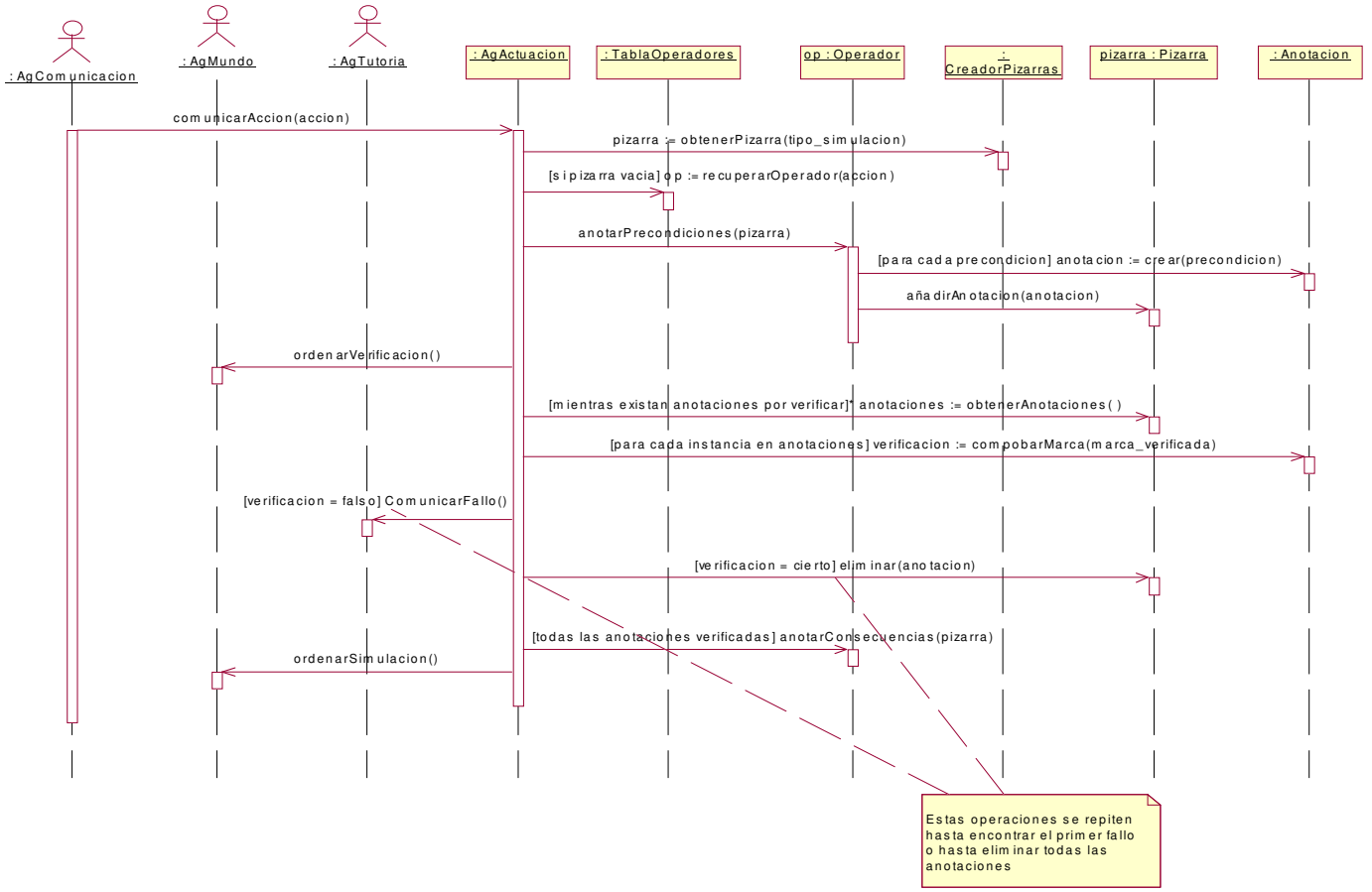


Figura 61: Protocolo *Ejecutar Acción*

5.3.5. Fase de implementación

Tras tener un diseño detallado consistente y completo, la fase de implementación se reduce a desarrollar algunos algoritmos que por su relativa complejidad se especifican aquí en un lenguaje de pseudocódigo.

En primer lugar se detalla la implementación de las políticas de acceso a la pizarra.

La pizarra declara dos variables que utiliza para controlar el número de agentes que acceden en lectura y también en escritura:

```
lectores = 0;
escritores = 0;
obtenerAnotaciones() {
   mutex {
      while(escritores > 0) wait();
      lectores++;
   }
   //operaciones de lectura sobre el recurso
   mutex {
      lectores--;
      if (lectores = 0) signal();
   }
}
```

```
eliminarAnotacion() {
   mutex {
      while(lectores > 0) wait();
      escritores++;
   }
   //operaciones de escritura sobre el recurso
   mutex {
      escritores--;
      //código de reestructuración de datos
      if (escritores = 0) signal();
   }
}
```

Para la implementación de este código se ha utilizado el lenguaje Java, por lo que se han aprovechado las primitivas y mecanismos de sincronización que ofrece el propio lenguaje.

La estructura que almacena los datos contenidos en la pizarra es una estructura de tipo lista. Es de vital importancia que las operaciones que afectan a la integridad de esta lista se efectúen en rigurosa exclusión mutua, por eso los bloques de código que han de realizar modificaciones estructurales están protegidos por secciones críticas.

Otro aspecto que se puede incluir dentro de los detalles de la implementación es el formato de definición de operadores. El *Agente de Actuación* cargará la información referente a los operadores desde un fichero de tipo XML que establece una organización jerárquica de la siguiente manera:

Nodo: ACTUACION
- ***Atributos***: Name="ConfiguracionActuacion"
- **Nodo**: OBJETOS
  - **Nodo**: OBJETO
    - ***Atributos***: Name
    - **Nodo**: OPERADOR
      - ***Atributos***: Name Descripcion Activacion
      - **Nodo**: PARAMETROS (nodo obligatorio)
        - ***Atributos***: ninguno
        - **Nodo**: ARGUMENTO (nodo opcional)
          - ***Atributos***: Name
      - **Nodo**: PRECONDICIONES (nodo obligatorio)
        - ***Atributos***: ninguno
        - **Nodo**: META (nodo opcional)
          - ***Atribuos***: Name Descripcion
          - **Nodo**: ARGUMENTO (nodo opcional)
            - ***Atributos***: [%]Name (% indica nombre de constante)
      - **Nodo**: POSTA (nodo obligatorio)
        - ***Atributos***: ninguno
        - **Nodo**: META (nodo opcional)
          - ***Atribuos***: Name Descripcion
          - **Nodo**: ARGUMENTO (nodo opcional)
            - ***Atributos***: [%]Name (% indica nombre de constante)
      - **Nodo**: POSTB (nodo obligatorio)
        - ***Atributos***: ninguno
        - **Nodo**: META (nodo opcional)
          - ***Atribuos***: Name Descripcion
          - **Nodo**: ARGUMENTO (nodo opcional)
            - ***Atributos***: [%]Name (% indica nombre de constante)

Nodo: CONSECUENCIAS (nodo obligatorio)
Atributos: ninguno
Nodo: INVENTARIO (nodo opcional)
Atributos: Destinatario Descripcion
Nodo: ARGUMENTO (nodo opcional)
Atributos: [%]Name (% indica nombre de constante)
Nodo: SIMULACION (nodo opcional)
Atributos: Destinatario Descripcion
Nodo: ARGUMENTO (nodo opcional)
Atributos: [%]Name (% indica nombre de constante)
Nodo: ANIMACIONES (nodo opcional)
Atributos: ninguno
Nodo: ANIMACION (nodo opcional)
Atributos: Name Destinatario Descripcion
Nodo: ARGUMENTO (nodo opcional)
Atributos: [%]Name (% indica nombre de constante)
Nodo: BLOQUE (nodo opcional)
Atributos: Destinatario Descripcion
Nodo: ANIMACION (nodo opcional)
Atributos: Name Descripcion
Nodo: ARGUMENTO (nodo opcional)
Atributos: [%]Name (% indica nombre de constante)

Aclaraciones:

1. En este fichero pueden existir un número indefinido de nodos OBJETO, y en cada uno de ellos un número indefinido de nodos OPERADOR. Sin embargo, todos los operadores deben tener cinco nodos sucesores obligatoriamente: PARAMETROS, PRECONDICIONES, POSTA, POSTB y CONSECUENCIAS.
2. El comentario "nodo opcional" indica que dicho nodo puede aparecer o no en el fichero, y en caso de hacerlo podrá estar un número indefinido de veces.

3. Los atributos precedidos del carácter % indican que son nombres de constantes. El resto son nombres de variables. Esto tiene su repercusión en el momento de la instanciación de operadores.
4. Para lograr la correcta instanciación de operadores en tiempo de ejecución, estos han de estar bien definidos sintácticamente. Para ello unicamente es necesario tener en cuenta una regla, y es que todos y cada uno de los nombres de los argumentos que aparecen en las precondiciones, postcondiciones o consecuencias, y que no son constantes (no están precedidos por el carácter %) han de estar incluidos entre los parámetros del operador.
5. La semántica del conjunto de operadores definidos es responsabilidad del diseñador del entorno virtual.

Por último, cabe decir que el formato de este fichero se corresponde exactamente con estructura de la tabla de operadores que el *Agente de Actuación* carga en memoria al inicio de su ejecución.

6. Conclusiones

Al final de este trabajo, tras haber adquirido una cierta perspectiva sobre lo que en él se expone, se puede proceder a realizar un pequeño ejercicio de introspección para extraer algunas conclusiones a modo de síntesis.

Al igual que en la realidad, en un mundo virtual lo que sucede nos llega a través de los sentidos, la vista, el oído, el tacto, etc., y de la misma forma que nosotros razonamos sobre estos estímulos, un sistema informático basado en realidad virtual también debe poder hacerlo y así adaptarse y cambiar y reaccionar en base a ellos. De lo que se está hablando y lo que virtualmente se intenta imitar no es ni más ni menos que de un concepto tan sencillo (aparentemente) como es el principio de acción-reacción.

"Toda acción tiene su reacción proporcional y en sentido opuesto", esto reza el principio físico que, si se extrapola, está presente en cada acción que realizamos tanto real como virtual. La diferencia estriba en que en la realidad este efecto nos viene dado por la naturaleza física de las cosas, mientras que en un mundo virtual el sistema ha de estar capacitado para percibir, razonar y posteriormente simular dicho efecto en su sentido más general.

Estas capacidades de percepción adquieren mayor relevancia en el ámbito de sistemas orientados a la supervisión del aprendizaje, pero por supuesto no se pretende que sean exclusivas de éstos. Aunque esta investigación ha oscilado en torno al objetivo de la asistencia en el entrenamiento, y lo que se ha presentado en ella se enmarca en el conjunto de un sistema inteligente de tutoría basado en entornos virtuales, no obstante podría generalizarse a otros tipos de sistemas de realidad virtual, puesto que la interacción a través de la percepción está presente en todos ellos sea cual fuere su propósito.

Algunas de las claves y dificultades de esta investigación son las siguientes:

En primer lugar se ha puesto de manifiesto la enorme complejidad que supone el desarrollo de entornos virtuales. El hecho de abarcar un gran número de materias distintas en cuanto a comunicaciones, interacción hombre-máquina, representación gráfica, representación del conocimiento, inteligencia artificial, etc., supone no sólo un

esfuerzo de investigación sobre cada uno de los temas en cuestión, sino también respecto a la manera de integrar y combinar todas estas áreas de la informática para ponerlas al servicio del objetivo perseguido.

Centrándonos en la interacción, núcleo de todas las reflexiones de estas páginas, llegamos al entendimiento de la necesidad de adoptar como guía los conceptos de generalización y abstracción. Conseguir construir sistemas basados en realidad virtual donde el tratamiento y gestión de la interactividad con los usuarios se haga de manera general y abstracta en cuanto a fabricantes, tecnologías, dispositivos de acceso empleados o procesamiento de la información percibida, tiene como fruto que se pueda volcar toda la atención en el desarrollo de las posibilidades que el entorno ofrece, su dinamismo o su inteligencia, relegando a un segundo plano aspectos que hasta ahora, y como se ha podido observar a lo largo de la lectura, no son nada triviales.

Al fin y al cabo, lo que se pretende en el desarrollo de entornos virtuales, ya sean inteligentes o no, es, dicho en términos coloquiales, no tener que "reinventar la rueda" constantemente y poder concentrar esfuerzos en mejorar las prestaciones de los mismos, su realismo, su inteligencia.

Para lograr esto, una cosa sí es segura, y es que el avance de las diferentes tecnologías debe ir en consonancia con la mejora en el modo en que éstas son utilizadas y combinadas.

7. Trabajo futuro

A lo largo de estas páginas se han planteado las bases de un posible modelo de diseño para la gestión de dispositivos de realidad virtual de diferentes características y su integración en el desarrollo de entornos virtuales. Sin embargo, la continua investigación en esta área hace que, en absoluto, el trabajo pueda ser dado por concluido. Como líneas de actuación inmediatas se sugieren las siguientes:

- En primer lugar, a pesar de que el modelo ha sido puesto en práctica con un conjunto variado de dispositivos, sería conveniente analizar el comportamiento y la capacidad de adaptación frente a un conjunto más amplio de diversos fabricantes y tecnologías.
- Como ha sido expuesto, se ha intentado independizar totalmente la gestión de dispositivos del diseño del mundo virtual, pero en ciertas partes esto ha resultado imposible, puesto que el objetivo final es actuar sobre los objetos del mundo para cambiar sus posiciones y orientaciones. Ya que esta dependencia es lógicamente inevitable, sería bueno revisar el diseño de la interfaz que el mundo ofrece al exterior para hacerla aún más general y conseguir, de esta forma, que en la implementación de ciertos niveles de la jerarquía de dispositivos mostrada haya que realizar tratamientos casuísticos que dependen de la combinación de dispositivos elegida.
- Se ha propuesto una posible solución para la gestión de aquellos dispositivos cuyo funcionamiento está dirigido por eventos; sin embargo, ésta ha sido una profundización ligera puesto que se ha reducido a dispositivos conocidos como es el teclado. Convendría ahondar mucho más en este aspecto.
- Otra de las posibles mejoras que se podrían llevar a cabo es la de dotar al sistema de la capacidad para calibrar los dispositivos, ajustar los protocolos de comunicación y reducir ruidos de transmisión, sin necesidad de recurrir a aplicaciones externas.

- Por último, sería recomendable estudiar la posibilidad de hacer que el denominado objeto de referencia dentro del mundo virtual, que en este caso se corresponde con la mano virtual, sea todavía más genérico de cara al dispositivo responsable de su control.

En cuanto a la gestión de las acciones realizada a través del *Agente de Actuación*, es posible llevar a cabo ciertas mejoras. Por ejemplo, reducir las responsabilidades que dicho agente tiene en las funciones de planificación, como las de calcular el estado resultante de aplicar una acción o hacer el cálculo de las distintas ordenaciones de precondiciones en los operadores que satisfacen un objetivo. Estas operaciones pueden entenderse como propias del algoritmo de planificación empleado, y por tanto parece lógico que sean realizadas por la entidad responsable de su ejecución.

Actualmente se está trabajando en la mejora de la comunicación existente entre esta entidad de planificación y los agentes que, como el de *Actuación*, tienen como misión colaborar en la construcción de planes. La investigación se dirige hacia el desarrollo de un mecanismo de pizarra que evite que el intercambio de información se haga mediante protocolos de petición-respuesta, proporcionando a dichos agentes más capacidad de proactividad, al dejar que sean ellos mismos los que, mediante el análisis de la información de la pizarra, decidan si han de actuar o no.

También se está pensando sobre la posibilidad de hacer una definición de operadores más general, en la que distintos objetos puedan compartir la misma definición para un cierto operador, y además que éstos estén clasificados en función de la actividad a realizar para evitar definiciones innecesarias de operadores que no van a ser usados.

Por último, se sugiere el estudio de la posibilidad de distribuir el papel del *Agente de Actuación*, dotando al sistema de varias instancias del mismo. La razón es evitar posibles situaciones de "cuello de botella", al ser un entorno multiusuario en el que todos los usuarios realizan acciones simultáneamente que han de ser comunicadas a este agente para el procesado de sus precondiciones y sus consecuencias.

8. Bibliografía

[Aston, 2003] **Virtual Environments for Training.** Benjamin Aston. 3rd Annual CM316 Conference on Multimedia Systems. University of Southampton, UK. (Enero del 2003).

http://mms.ecs.sotonac.uk/mms2003/index.php

[Bates et al. 1992] **An architecture for action, emotion and social behaviour.** Bates, J., Bryan Loyall, A., Scott Reilly, W. Technical Report CMU-CS-92-144. School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA. (1992).

[Bates, 1994] **The role of emotion in believable agents.** Bates, J. Communications of the ACM, 37(7):122-125. (1994).

[Bellifemine et al., 2002] **Jade Programmer´s guide.** Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, Giovanni Rimmasa. http://sharon.cselt.it/projects/jade/ (2002).

[de Antonio et al. 2005] **An Agent-Based Architecture for Virtual Environments for Training.** de Antonio, A., Rámirez, J., Méndez, G. En Developing Future Interactive Systems, M.I.Sánchez (Ed.), Idea Group. (2005)

[Ferré, 2002] **Desarrollo orientado a objetos con UML. v2.3** Xavier Ferré Grau. No publicado. Facultad de Informática. Universidad Politécnica de Madrid. (2002).

[Fikes & Nilsson, 1971] **STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving.** Fikes, R., Nilsson, N. Artificial Intelligence, Vol. 2. (1971).

[Fob, 1999] **The Flock of Birds. Installation and operation guide.** Ascension Technology Corporation. 910002-A Rev A, pp. 12. http://www.ascension-tech.com. (Enero de 1999).

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

[Hayes-Roth, 1995] **An Architecture for Adaptive Intelligent Systems.** Hayes-Roth, B. Artificial Intelligence 71 (1-2), pp. 329-365. (1995).

[Huhns, 1998] **Readings in Agents.** Huhns, M., Singh, M.P. Morgan Kauffmann Publishers. (Enero de 1998).

[Kepell, 1997] **Is the Elephant Really There? – Virtual Reality in Education.** Presentación del seminario realizada por el Dr. Colin Macpherson y el Dr. Mik Keppell en Central Queensland University (3 de Octubre de 1997).

http://infocom.cqu.edu.au/Units/aut99/00101/00101/RESOUCE/TUTORIAL/VR-PRES.PDF

[Larman, 1999] **UML y patrones.** C. Larman. Prentice Hall, 1999.

[Maes, 1995] **Artificial Life Meets Entertainment: Life like Autonomous Agents.** Maes, P. Communications of the ACM, 38(11):108-114. (1995).

[Méndez et al. 2003] **Steve Meets Jack: the Integration of an Intelligent Tutor and a Virtual Environment with Planning Capabilities.** Méndez, G., Rickel, J., de Antonio, A. 4th Internacional Workshop on Intelligent Virtual Agents, Lecture Notes on Artificial Intellingence Vol. 2792, pp. 325-332, IVA'03, (2003).

[Moraïtis et al. 2003] **Engineering JADE Agents with the Gaia Methodology.** Moraïtis, P., Petraki, E. y Spanoudakis, N.I. Dept. of Computer Science. University of Cyprus. (2003).

[Parnas D. L. 1971] **On the criteria to be used in descomposing systems into modules.** D.L. Parnas. Programming Techniques. R. Morris Editor. Carnegie-Mellon University. (1971).

[Perry et al. 2001] **Una investigación sobre interfaces actuales de realidad virtual.** Perry, D.S. Linellen, Smith, M. Christopher, Yang, S. ACM Crossroads Student Magazine. (2001). http://www.acm.org/crossroads/

[Rickel & Johnson, 1998] **STEVE: A Pedagogical Agent for Virtual Reality**. Jeff Rickel and W. Lewis Johnson. Information Sciences Institute & Computer Science Department. University of Southern California. (1998). http://www.isi.edu/isd/VET/vet.html

[Rickel & Johnson, 1999] **Animated agents for procedural training in virtual reality: Perception, cognition and motor control.** Rickel, J., Johnson, W.L. Applied Artificial Intelligence 13 pp. 343-382. (1999).

[Rickel & Johnson, 2000] **Task Oriented Collaboration with Embodied Agents in Virtual Worlds.** Rickel, J., Johnson, W.L. En J.Cassell, J. Sullivan, S.Prevost y E. Churchill (Eds.). Embodied Conversational Agents, pp 95-122. Boston: MIT Press. (2000).

[Russel & Norving, 1995] **Artificial Intelligence: a modern approach.** Russel, S., Norving, P. Prentice-Hall Inc. (1995).

[Selker, 1994] **A Teaching Agent that learns.** T. Selker. Communications of the ACM, 37(7): 92-99 (1994).

[Shoham, 1993] **Agent-oriented programming.** Shohan, Y. Artificial Intelligence, 60(1):51-92. (1993).

[Smith et al. 1994] **Programming Agents without a programming language.** Smith, D.C., Cypher, A., Spohrer, J. Communications of the ACM, (37)7:55-67. (1994).

[Stetter & Penrose, 2001] **The Electrochemical nose.** Stetter, J.R, Penrose, W.R. Department of Biological, Chemical and Physical Sciences. Illinois Institute of Technology. Chicago. (2001). http://electrochem.cwru.edu/ed/encycl/

[Tachi, 1995] **Whither Force Feedback?.** Tachi, S. En *IEEE: Proceedings of the Virtual Reality Annual International* Symposium en Research Triangle Park, NC. IEEE Computer Society Press. Washington: IEEE Computer Society Press, 227. (1995).

[Wooldridge and Jennings, 1995] **Intelligent Agents: Theory and Practice.** Wooldridge, M. J., Jennings, N.R. The Knowledge Engineering Review. (1995).

[Wooldridge et al. 2000] **The Gaia Methodology for agent-oriented análisis and design.** Wooldridge, M., Jennings, N.R, Kinny, D. Autonomous Agents and Multi-Agent Systems, 3(3), pp.285-312. Kluwer Academic publishers. (2000).