Entornos virtuales basados en técnicas de aprendizaje activo para la enseñanza de la orientación a objetos

TESIS DOCTORAL

Guillermo Jiménez Díaz

Departamento de Ingeniería del Software
e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Marzo 2008

Entornos virtuales basados en técnicas de aprendizaje activo para la enseñanza de la orientación a objetos

Memoria que presenta para optar al título de Doctor en Informática

Guillermo Jiménez Díaz

Dirigida por los doctores

Mercedes Gómez Albarrán y **Pedro Antonio González Calero**

Departamento de Ingeniería del Software
e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Marzo 2008

Cuéntame y lo olvidaré.
Enséñame y lo recordaré.
Involúcrame y lo comprenderé.

Agradecimientos

Te lo juro por Dios, Jimmy; si sobrevivo a esto,
voy a ponerme a bailar.

El último Boy Scout (1991)

Ya estamos aquí, escribiendo las últimas líneas de esta Tesis, las primeras que leeréis. Quién me iba decir a mí que casi once años atrás, cuando decidí que iba a hacer la Ingeniería Técnica de Informática de Sistemas porque (y cito mis propios pensamientos por aquel entonces) "no sabía si me iba a apetecer estudiar más de tres años" terminaría haciendo un doctorado. Y no sólo eso. Quién me iba a decir que terminaría dando clases en la Universidad y preocupándome por enseñar a esos que ahora están donde yo me encontraba por entonces.

Uno de los responsables de esta decisión fue Pedro, que me "engañó" en una de esas charlas que se dan en los últimos años de carrera sobre las salidas profesionales. Nos contó todo lo bonito que era trabajar en la universidad, escribir artículos y asistir a conferencias en distintos lugares del mundo. Por supuesto, en esa charla nunca nos habló de lo que es escribir una Tesis. Posteriormente, no contento con *camelarme*, engaño a Mercedes para que, juntos, me acogieran como su *monaguillo*. Pedro y Mercedes, *la extraña pareja*; para mí, la combinación perfecta. Pedro, me has dado ideas frescas cuando yo ya no veía más allá de mi nariz. Mercedes, todo lo que te he dado lo has revisado y dado vueltas hasta conseguir que quedaran los mínimos cabos sueltos. Os habéis complementado a la perfección para hacerme todo esto un poco más fácil. Gracias.

Papá, mamá, Yola, os agradezco ese apoyo incondicional que siempre me dais en todo lo que he ido haciendo. Ese apoyo intenso pero, a la vez, sin ruido, desde la distancia, como el de la leona que duerme junto al juego de sus cachorros pero que en realidad tiene todos sus sentidos puestos en ellos. Siempre os siento cerca de mí y sé que lo habéis pasado mal conmigo cuando yo he tenido baches y no podíais ayudarme porque éste era un camino que tenía que hacer solo. Ya hemos llegado al final de este camino. Seguid conmigo en los que nos quedan por recorrer.

Para casi tres décadas de vida parece que dos años son poco, pero ese

es el tiempo que llevo en vuestra compañía, Ana y Juanan. Y ya sabéis lo importante que habéis conseguido ser para mí y lo mucho que me habéis ayudado para culminar todo esto. Las palabras se me hacen cortas. Juanan, tu apoyo ha sido incondicional y siempre que lo he necesitado me has conducido a que me olvidara de esos *"pajotes mentales"* que no me llevaban a ningún sitio. Ana, tú me has enseñado lo más importante para terminar esta Tesis pero que yo no había leído en ninguno de los artículos que aparecen en este trabajo: me has enseñado a hacerme preguntas, a tener un poco más de confianza en mí mismo y a creer en que lo iba a conseguir. Gracias por todo lo que compartís conmigo.

Si se me permite el símil ciclista, Pedro y Marco han sido mis gregarios (de lujo) en este duro puerto de montaña. Me han dejado ir a su rueda durante todos estos años y puedo decir que todo lo que sé sobre entornos virtuales y videojuegos (bueno, y de C++) se lo debo a ellos. Y para qué quedarnos solamente con la pareja si podemos montar un trío: Javi, tú bien me has ayudado con el aparataje estadístico y con tus teorías sobre probabilidad y *echar fichitas* para poder ganar algún día. Pero los tres no sólo me habéis dado conocimientos sino también una buena compañía y un necesario desahogo de tensiones en la pista de squash. Ese es el núcleo duro.

Si todavía no te has encontrado en estos agradecimientos puede que aparezcas en este párrafo. Alicia y Eloy, después de tantos años, con lo que hemos tenido que pasar juntos y aún seguimos tan cerca. Luis Domínguez, jugón, creador de monstruos. Qué bien viene tu compañía y tu bolsa llena de juegos de mesa para esos días en los que acababas saturado de Tesis. Ricardo, compañero de charlas que se prolongaban durante la noche y que nos desahogaban tanto durante esos años en los que convivimos juntos. Raquel, siempre una palabra de ánimo y una mano en el hombro cuando has creído que lo necesitaba. Eva y Carmen, las primeras que me fuisteis abriendo un camino en el departamento. Cuánto me ha ayudado que fueseis hablando de ese alumno que estaba interesado en hacer el doctorado y trabajar en la universidad. Luis Hernández, el *jefe* soñado. Qué fácil me lo has puesto siempre para combinar mi trabajo en el departamento con esta investigación y con todos esos cursos que nos ayudan a vivir un poco mejor. Y a todos esos becarios que me habéis sufrido y que me habéis ayudado en los trabajos de implementación, así como aquéllos que habéis participado en las evaluaciones, gracias.

Y si aún no has aparecido en estos agradecimientos no te sientas contrariado, hombre. Si me conoces bien sabes que siempre he sido un tipo despistado, que me pierdo y con tendencia a olvidarme de algunas cosas. A ti también te lo agradezco porque seguro que alguno de tus gestos me ha ayudado a llegar hasta aquí.

Ahora pónganse cómodos que la función va a comenzar. Apaguen sus teléfonos móviles. Bajamos las luces, encendemos el proyector y... ¡acción!

Resumen

La orientación a objetos facilita el desarrollo de software a gran escala y de calidad. A pesar del extendido uso de este paradigma, la enseñanza de la orientación a objetos no está exenta de dificultades. Ésta se fundamenta en una sólida base de conceptos elementales sobre la que ir añadiendo experiencias de diseño de aplicaciones orientadas a objetos.

Se han desarrollado un gran número de herramientas basadas en visualizaciones interactivas que dan soporte a la docencia de la enseñanza de la orientación a objetos. Sin embargo, estas herramientas no suelen proporcionar soporte para actividades de diseño ni promueven la realización de tareas colaborativas que fomentarían el intercambio de experiencias entre alumnos.

El alto valor pedagógico de técnicas de aprendizaje activo como el role-play en sesiones de diseño orientado a objetos ha conducido al estudio de la aplicabilidad del role-play en entornos virtuales como ayuda a la enseñanza de la orientación a objetos. En esta memoria de Tesis Doctoral se describe este estudio, cuyos principales resultados han sido los siguientes:

1. Realización de una propuesta de entornos virtuales de role-play, con gran capacidad de interacción, ricos en información de diseño y en los que se pueden realizar actividades similares a las desempeñadas en sesiones presenciales de role-play.

2. Diseño de una arquitectura de alto nivel para entornos virtuales de role-play. Esta arquitectura facilita el desarrollo de entornos que usan el role-play con distintos fines y que emplean distintas alternativas pedagógicas para controlar y evaluar las sesiones desempeñadas por los alumnos.

Para poner a prueba esta propuesta se han realizado dos instanciaciones diferentes de los entornos virtuales de role-play. Estas instanciaciones han ido acompañadas de sendos prototipos sobre los que se ha evaluado la idoneidad del traslado de los principales elementos de las sesiones de role-play a un entorno virtual.

Índice

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción

Buenos días... y por si no volvemos a vernos:
buenos días, buenas tardes y buenas noches.
El show de Truman (1998)

La orientación a objetos ha tomado una posición predominante en el desarrollo de software desde finales del pasado siglo. Esta tecnología maduró enormemente durante la década de los 90 gracias a la aparición de metodologías y técnicas de diseño orientado a objetos, el desarrollo de frameworks y la definición de patrones de diseño. Además, este paradigma facilita en gran medida el desarrollo de software a gran escala, reutilizable y de calidad.

¿A qué fue debido el auge de este paradigma? Muy sencillo: su naturalidad a la hora de describir y representar sistemas complejos. La orientación a objetos centra su atención en el problema que ha de resolver, lo descompone en pequeñas unidades con comportamiento y estado propios y compone una solución al problema haciendo uso de estas unidades, conocidas como *objetos*.

Algunos autores dan una especial relevancia a la proyección de las características humanas que se hacen sobre un objeto. Definen la orientación a objetos como una forma de desarrollo en la que la solución es similar a una comunidad de personas virtuales (agentes o máquinas). Consideran que existe una metáfora muy clara entre un objeto y una persona: un objeto es una entidad autónoma, con unos atributos propios, que es capaz de desarrollar un conjunto de tareas y que necesita de la interacción con otros objetos para llevar a cabo tareas de mayor complejidad. Esta metáfora se conoce por el nombre de *antropomorfización*.

La orientación a objetos no se reduce a los *objetos* sino que existe un conjunto de conceptos elementales relacionados que conforman la base de este paradigma. Estos conceptos han de ser entendidos y dominados como paso previo al desarrollo de aplicaciones siguiendo este paradigma. Tras comprender los conceptos elementales, hay que aprender a analizar problemas con la perspectiva de este paradigma y generar soluciones acordes que resuelvan el

problema planteado. Aunque antes se ha mencionado que la forma en la que la orientación a objetos aborda el análisis de un problema es *natural* para las personas, esto no implica que sea una tarea *trivial*. Un buen desarrollador requiere de una gran cantidad de experiencia a sus espaldas para saber analizar y diseñar software de calidad. Sólo así será capaz de seleccionar las soluciones más apropiadas de entre un conjunto de alternativas para cada problema concreto con el que se encuentre.

Existe una preocupación dentro de la comunidad docente sobre la forma en la que se enseña la orientación a objetos. Cada año aparecen un sinfín de artículos en los que los docentes relatan sus experiencias en la enseñanza de esta disciplina y en los que proponen nuevas actividades, nuevos lenguajes, nuevas aproximaciones pedagógicas o nuevas herramientas de apoyo a la docencia. Constantemente aparecen discusiones sobre si uno u otro lenguaje es el más apropiado o sobre cual es la alternativa pedagógica que da mejores resultados. Pero, de entre todas las alternativas, parece que existen dos ideas fundamentales que son universalmente compartidas a la hora de enseñar el paradigma de la orientación a objetos: es necesario la utilización de ejemplos complejos y sólo se adquieren las destrezas necesarias para hacer buenos diseños gracias a la experiencia.

Estas dos ideas están íntimamente ligadas. Por un lado, los ejemplos de suficiente complejidad ayudan a los alumnos a comprender el verdadero valor de este paradigma. Además, estos ejemplos llevan la impronta del desarrollador que los ha creado, la experiencia de diseño de esta persona. Por lo tanto, el estudio de estos ejemplos es la primera fuente de experiencia de diseño de la que los alumnos beben. Pero esto no es suficiente. Al igual que a montar en bicicleta sólo se aprende montando en bicicleta, las habilidades de un buen diseñador sólo se adquirirán diseñando. Por este motivo es necesario fomentar actividades formativas en las que los alumnos tengan la oportunidad de poner en práctica sus habilidades de diseño y de intercambiar experiencias con otros alumnos.

El diseño orientado a objetos no es la única habilidad que se adquiere mediante la experiencia. Muchas habilidades sociales requieren ser entrenadas para poder ser asimiladas. Para poner en práctica estás habilidades se utilizan técnicas de aprendizaje activo, que transforman al alumno, tradicionalmente observador pasivo de una clase teórica, en un elemento activo de su propio proceso de aprendizaje. Una de estas técnicas es la que se conoce como interpretación de roles o *role-play*. Esta técnica sugiere que el alumno interprete el papel (rol) de otra persona dentro de una simulación. De esta forma, el alumno es capaz de interiorizar y entender el comportamiento o el sentir de la persona cuyo rol está interpretando.

Volviendo a nuestro objeto de estudio, el role-play también se ha usado como técnica de diseño orientado a objetos. Esta técnica aprovecha la metáfora de la antropomorfización, anteriormente descrita, haciendo que una

persona se meta en la piel de un objeto de una aplicación software. En estas sesiones de role-play, un conjunto de personas interactúan, de igual forma que lo harían los objetos que representan, en un determinado escenario de ejecución de la aplicación. Durante estas sesiones, cada participante no sólo aprende del rol que está interpretando sino también de los roles y de la experiencia del resto de participantes. El role-play en la orientación a objetos se usa fundamentalmente para el análisis y diseño de aplicaciones. Pero el alto valor de adquisición de experiencia de diseño de esta técnica ha hecho que también se haya trasladado al ámbito de la enseñanza, tanto para el desarrollo de actividades para aprender diseño como para el análisis y comprensión de los conceptos elementales del paradigma de la orientación a objetos.

Volviendo a las aportaciones realizadas por la comunidad docente para mejorar los métodos de enseñanza de la orientación a objetos, es reseñable la gran proliferación de herramientas de soporte a la enseñanza que han ido apareciendo a lo largo de los años. La mayoría de estas herramientas comparten dos características: tienen una alta componente visual y promueven la interactividad con el alumno. La visualización facilita la comprensión de los conceptos abstractos elementales de este paradigma. Además, ayuda a entender la componente dinámica de la orientación a objetos: el paso de mensajes y la interacción entre objetos. Además, estas herramientas se esfuerzan en que el alumno no sea un mero observador de animaciones. Estas herramientas promueven que el alumno participe, ya sea creando las visualizaciones, ya sea controlando la ejecución y la información presentada en la visualización.

Del análisis realizado de estas herramientas se ha extraído que sólo una mínima parte de las mismas dan soporte a actividades de diseño, predominando las herramientas de soporte a la programación orientada a objetos. También hay que destacar que hacen especial énfasis en el soporte al trabajo individual del alumno, dejando de lado tareas colaborativas que fomentarían el intercambio de experiencias entre los mismos.

Para aliviar esta deficiencia encontrada en las herramientas de apoyo a la docencia, se propone aunar las experiencias de las herramientas de visualización con las de las técnicas de aprendizaje activo en las clases presenciales. De acuerdo a estas dos ideas, este trabajo de Tesis Doctoral presenta un estudio de la aplicabilidad del role-play en entornos virtuales de apoyo a la enseñanza en el dominio de la orientación a objetos. Este estudio ha conducido a la definición de los entornos virtuales de role-play, entornos virtuales a los que se han trasladado los elementos, las mecánicas y el conocimiento necesarios para el desarrollo de sesiones de role-play en la enseñanza de la orientación a objetos. Además, se ha propuesto una arquitectura de alto nivel para estos entornos, cuya instanciación permite la construcción de entornos virtuales de role-play con objetivos pedagógicos variados.

Este trabajo también incluye, a modo de ejemplo, el diseño de dos instanciaciones distintas de estos entornos virtuales de role-play. Para estos

entornos se han desarrollado sendos prototipos que han sido evaluados por docentes para determinar la adecuación del traslado de las sesiones de role-play a un entorno virtual.

1.1. Estructura de la memoria

La secuencia de capítulos que conforman esta memoria de Tesis es la siguiente:

Capítulo 2. Enseñanza de la orientación a objetos. Este capítulo describe la filosofía que hay tras el paradigma de la orientación a objetos y analiza las principales dificultades que encuentran tanto docentes como alumnos en el proceso de enseñanza/aprendizaje de este paradigma. De este análisis se han extraído una serie de directrices a tener en cuenta a la hora de enseñar exitosamente esta disciplina.

Capítulo 3. Recursos para la enseñanza de la orientación a objetos. Este capítulo contiene el estudio realizado sobre un numeroso conjunto de recursos de apoyo a la docencia de la enseñanza de la orientación a objetos. Una mayoría de estos recursos se caracterizan por tener una fuerte componente visual y por fomentar la participación. Aunque cubren buena parte de las dificultades identificadas en el capítulo anterior, suelen dejar de lado el diseño orientado a objetos y no fomentan la colaboración y el intercambio de experiencias entre los alumnos.

Capítulo 4. Uso de técnicas de aprendizaje activo para la enseñanza de la orientación a objetos. Este capítulo comienza con la descripción del role-play, una técnica de aprendizaje activo utilizada en diseño orientado a objetos. Más adelante se revisa su utilización en la enseñanza del paradigma orientado a objetos y se narran nuestras propias experiencias en la aplicación de esta técnica, combinada con tareas de refactorización, para la enseñanza de patrones de diseño. En el desarrollo de estas experiencias se ha seguido de manera fiel las directrices identificadas en el primer capítulo. El capítulo también incluye los resultados de las evaluaciones de dichas experiencias.

Capítulo 5. Traslado de las técnicas de aprendizaje activo a entornos virtuales de enseñanza. Este capítulo presenta la propuesta de traslado de las sesiones de role-play a entornos virtuales como apoyo a la enseñanza de la orientación a objetos. Para ello, primero se ha extrapolado, a partir de nuestras experiencias presenciales, un patrón general de las sesiones de role-play que da cobertura al uso de las mismas con distintos objetivos pedagógicos. Los ejes de variabilidad de dicho patrón han sido claramente identificados. A partir de ahí, se han

analizado las necesidades de representación (visual y de información), interacción y autoría que conlleva el traslado de las sesiones de role-play a entornos virtuales. Además, y teniendo en cuenta los ejes de variabilidad identificados, se ha propuesto una arquitectura general de los entornos virtuales de role-play.

Capítulo 6. Desarrollo de entornos virtuales de role-play: ViRPlay3D y ViRPlay3D 2. Este capítulo describe las dos instanciaciones realizadas para comprobar la aplicabilidad de la propuesta realizada en el anterior capítulo. El primero de los entornos incluye las representaciones y mecánicas principales de los entornos virtuales de role-play y tiene como objetivo pedagógico el análisis del comportamiento en ejecución de una aplicación orientada a objetos. El segundo desarrolla todas las mecánicas de estos entornos virtuales y se ha diseñado para dar soporte a sesiones de role-play para la enseñanza de patrones de diseño, similares a las descritas en el Capítulo 4. Ambas instanciaciones han seguido la arquitectura general para los entornos virtuales de role-play propuesta en el capítulo anterior. Para cada una de ellas se ha desarrollado un prototipo que ha sido evaluado por docentes de la Facultad de Informática de la Universidad Complutense de Madrid. Los resultados de estas evaluaciones se incluyen también en este capítulo.

Capítulo 7. Conclusiones y trabajo futuro. Esta memoria de Tesis concluye con un resumen de las principales aportaciones realizadas y con la descripción de algunas de las posibles líneas de trabajo futuro.

1.2. Publicaciones

El trabajo descrito en esta memoria de Tesis ha generado una serie de publicaciones a lo largo de los últimos años. Aunque cada una de ellas ha sido referenciada en esta memoria en el lugar en el que se trata su contenido, aquí se enumeran, a modo de resumen:

1. Jiménez-Díaz, G., Gómez-Albarrán, M., Gómez-Martín, M. A. y González-Calero, P. A. Software Behaviour Understanding Supported by Dynamic Visualization and Role-play. *10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE 2005). SIGCSE Bulletin*, vol. 37(3), páginas 54–58, 2005.

2. Jiménez-Díaz, G., Gómez-Albarrán, M., Gómez-Martín, M. A. y González-Calero, P. A. Understanding Object-Oriented Soft-

ware through Virtual Role-Play. En *5th IEEE International Conference on Advanced Learning Technologies (ICALT 2005)* (editado por P. Goodyear, D. G. Sampson, D. Yang, Kinshuk, T. Okamoto, R. Hartley y N.-S. Chen), páginas 875–877. IEEE Computer Society, Kaohsiung, Taiwan, 2005.

3. Jiménez-Díaz, G., Gómez-Albarrán, M., Gómez-Martín, M. A. y González-Calero, P. A. Virplay: Playing Roles to Understand Dynamic Behavior. En *9th Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts, at 19th European Conference on Object Oriented Programming (ECOOP 2005)*. Glasgow, UK, 2005.

4. Jiménez-Díaz, G., Gómez-Albarrán, M. y González-Calero, P. A. "Before and After": An active and Collaborative Approach to Teach Design Patterns. En *8th International Symposium on Computers in Education (SIIE 2006)* (editado por L. Panizo Alonso, B. Fernández Manjón, L. Sánchez González y M. Llamas Nistal), vol. 1, páginas 272–279. Servicio de Imprenta de la Universidad de León, León, Spain, 2006.

5. Jiménez-Díaz, G., González-Calero, P. A. y Gómez-Albarrán, M. Using Role-play Virtual Environments to Learn Software Design. En *European Conference on Games-Based Learning (ECGBL 2007)* (editado por D. Remenyi), páginas 143–151. Academic Conferences, Paisley, Scotland, UK, 2007.

6. Jiménez-Díaz, G., Gómez-Albarrán, M. y González-Calero, P. A. Pass the ball: Game-based Learning of Software Design. En *6th International Conference on Entertainment Computing (ICEC 2007)* (editado por L. Ma, M. Rauterberg y R.Nakatsu), vol. 4740, páginas 49–54. Springer, Shanghai, China, 2007.

7. Jiménez-Díaz, G., Gómez-Albarrán, M. y González-Calero, P. A. Role-play Virtual Environments: Recreational learning of software design. En *Second Conference on Technology Enhanced Learning (EC-TEL 2007). Addendum to Conference Proceedings* (editado por E. Duval, R. Klamma y M. Wolpers), páginas 23–27. Crete, Greece, 2007.

8. Jiménez-Díaz, G., Gómez-Albarrán, M., Gómez-Martín, M. A. y Gónzález-Calero, P. A. Visualization and Role-play to Teach Object-Oriented Programming. En *Computers and Education - Towards Educational Change and Innovation* (editado por A. J. Nunes-Mendes), páginas 167–177. Springer, London, UK, 2008.

9. Jiménez-Díaz, G., Gómez-Albarrán, M. y González-Calero, P. A. Teaching GoF Design Patterns through Refactoring and Role-play. *International Journal of Engineering Education*, (Special Issue: Trends in Software Engineering Education). Aceptado y pendiente de publicación, 2008.

Capítulo 2

Enseñanza de la orientación a objetos

Houston, tenemos un problema

Apollo 13 (1995)

Saber programar siguiendo el paradigma de la orientación a objetos no consiste únicamente en saber programar con, por ejemplo, Java o C++. Si elegimos cualquier texto relacionado con este paradigma nos encontraremos que todos ellos comienzan explicando que la orientación a objetos supone una manera diferente de pensar con respecto a la programación procedimental *tradicional*. Y esto es algo que hay de tener en cuenta, como docentes, a la hora de enseñar a los alumnos lo que es la orientación a objetos.

Existe un gran *miedo* entre muchos autores (Meyer, 1997; Rentsch, 1982) con respecto a la comprensión de la orientación a objetos. Este paradigma se ha hecho tan popular en la comunidad de desarrollo de software que palabras como *clase, objeto o herencia* nos son muy familiares. Pero, ¿se comprende realmente cuál es su significado, qué es lo que hay tras ellas? Lamentablemente se puede decir que en muchas ocasiones no es así. Suele haber una vaga idea de lo que son pero no se tiene una idea clara.

Revisando la literatura relacionada con la enseñanza de la orientación a objetos se llega a una conclusión muy clara: enseñar orientación a objetos no es fácil. La orientación a objetos incluye una serie de conceptos novedosos, incluso para alumnos que ya han sido formados en otros paradigmas de programación, y conlleva enseñar una forma no trivial de análisis de problemas y de modelado de sistemas complejos.

En este capítulo se tratará el paradigma de la orientación a objetos y los problemas que conlleva la enseñanza de este paradigma. Se comenzará con una somera definición de lo que es la orientación a objetos y la filosofía

que hay detrás de este paradigma. Además, se hará una distinción entre tres términos que generalmente son tratados indistintamente: análisis, diseño y programación orientada a objetos.

Posteriormente se ahondará en la historia de este paradigma desde su nacimiento en los años 60. Se explicarán las principales motivaciones que condujeron a la aparición de este nuevo paradigma. Además, se irá viendo cuándo fueron apareciendo algunos de los conceptos clave de la orientación a objetos.

La siguiente sección detalla los conceptos fundamentales del paradigma de la orientación a objetos. En esta sección también se explican cuáles son las nociones elementales que hay que tener sobre diseño orientado a objetos. Por último, la sección concluye introduciendo un concepto de diseño experto pero que cada vez más se considera como fundamental a la hora de conocer este paradigma: los patrones de diseño.

Más adelante se hace un análisis de cuáles son las principales dificultades con las que docentes y discentes se encuentran durante el proceso de enseñanza de la orientación a objetos. La catalogación de estos problemas no es fácil ya que en ocasiones unos solapan con otros. Se ha intentado clasificar estos problemas y dificultades de acuerdo a los conceptos desarrollados en la sección anterior.

Por último, este capítulo finaliza con un resumen de las principales conclusiones extraídas sobre las dificultades de enseñar orientación a objetos. Incluye un análisis de estas dificultades y una serie de directrices a tener en cuenta a la hora de enseñar orientación a objetos.

2.1. Una definición de la orientación a objetos

La orientación a objetos es una técnica de modelado de software a gran escala que sugiere que tratemos la complejidad del sistema software de manera análoga a como los seres humanos tratamos la complejidad en el mundo real: descomponemos el problema en pequeñas piezas con comportamiento propio que nos sirven para componer una solución al problema. La orientación a objetos centra su atención en el problema, en lugar de en la solución. El análisis del problemas es el que nos conduce a una solución.

La orientación a objetos se basa en la estructuración de un sistema en base a sus objetos, en lugar de en base a las acciones que es capaz de realizar. Existe una diferencia principal entre la programación procedimental y la orientación a objetos. La programación procedimental realiza acciones sobre las estructuras de datos. En cambio, en la orientación a objetos se solicita a las estructuras de datos (los objetos) que realicen un determinado servicio. Esta diferencia queda resumida en la siguiente máxima (Budd, 2002):

> *No preguntes* qué es lo que puedes hacer a tus estructuras de

> datos. *Pregunta* qué es lo que tus estructuras de datos pueden hacer por ti.

Un programa orientado a objetos se estructura como una comunidad de objetos que interaccionan entre sí. Cada objeto proporciona un conjunto de servicios o realiza acciones que los otros miembros de la comunidad usan (Budd, 2002).

Una idea fundamental de la orientación a objetos frente a otros paradigmas es que no hace falta un coordinador central que controle todo el funcionamiento del sistema. Los objetos saben cómo comunicarse entre ellos y actúan de acuerdo a los mensajes recibidos, sin importarles qué otros objetos están a su alrededor.

Otros autores consideran que la principal aportación de la orientación a objetos al diseño de software es la metáfora de la *antropomorfización* (West, 2004), que consiste en proyectar las características humanas en un objeto. Este autor define el paradigma orientado a objetos como una forma de pensar en la que la solución supone una comunidad de personas virtuales (agentes o máquinas). Cada una de estas personas es una entidad capaz de desarrollar una tarea y tiene acceso a todo el conocimiento y recursos para completarla.

Existen tres conceptos relacionados con la orientación a objetos que a menudo tendemos a confundir y que es necesario distinguir:

- El *análisis orientado a objetos* es un método que examina los requerimientos de un sistema desde la perspectiva de los objetos encontrados en el vocabulario de dominio del problema (Booch, 1994). Este método se compone de las siguientes actividades: buscar los objetos que conforman un sistema, organizarlos, decidir cómo van a interaccionar, definir las operaciones que proporciona cada objeto y definir la estructura interna de los objetos (Jacobson et al., 1992).

- El *diseño orientado a objetos* es el método encargado de decidir qué tipos de objetos y qué operaciones van a ser implementadas. Es responsable de la definición de las clases y de la organización de las mismas de acuerdo a relaciones de herencia y composición. El diseño orientado a objetos oculta los detalles de implementación. No existe una clara diferencia entre cuándo termina el análisis y se pasa al diseño, por lo que es muy fácil confundir ambos términos. Además, generalmente se realizan en paralelo.

- La *programación orientada a objetos* consiste en la implementación de un sistema orientado a objetos. Queda ligado a un lenguaje orientado a objetos. Aunque tendemos a referirnos a la orientación a objetos con el término "programación orientada a objetos", realmente deberíamos usar el término "diseño orientado a objetos", que se aproxima más a su filosofía. La calidad de un sistema orientado a objetos reside realmente en su diseño, más que en el lenguaje en el que es implementado.

La orientación a objetos se ha convertido en el más popular de los paradigmas de programación. Esto se debe a que la orientación a objetos facilita el desarrollo de aplicaciones complejas. El uso de este paradigma es especialmente relevante en el desarrollo de software a gran escala y de calidad. Aunque existen distintos factores (tanto externos como internos) para medir la calidad del software (Meyer, 1997) nos vamos a quedar con tres que sobresalen del resto y sobre los que la orientación a objetos hace una mejora significativa: extensibilidad, reutilización y fiabilidad.

Extensibilidad Indica el grado de dificultad que tiene realizar cambios en el software para adaptarse a nuevos requisitos. La construcción mediante objetos hace que las posibles modificaciones a realizar sobre el software suelan ser locales a un objeto —o a unos pocos de ellos.

Reutilización Es una medida del grado en el que los distintos módulos que componen un sistema pueden ser reutilizados en otros sistemas. Ayuda en el desarrollo rápido de nuevos sistemas. En la orientación a objetos la unidad mínima de reutilización es el objeto. Además, este paradigma dispone de un conjunto de mecanismos que incrementan enormemente la reutilización.

Fiabilidad Es un término genérico que cubre dos factores: corrección y robustez. Indica la capacidad del software para realizar con exactitud las tareas especificadas y para reaccionar adecuadamente ante situaciones inesperadas. La orientación a objetos se preocupa tanto por el desarrollo de estructuras sencillas, extensibles y reutilizables que se puede asegurar la fiabilidad del sistema de manera más sencilla. Un software orientado a objetos sencillo y legible es la mejor arma contra la falta de fiabilidad.

2.2. Breve historia de la orientación a objetos

Aunque la orientación a objetos comenzó a despuntar durante la década de los 80, lo cierto es que las raíces de este paradigma se remontan a los años 60. A lo largo de estas casi cinco décadas, la orientación a objetos se ha convertido posiblemente en el paradigma más extendido en el diseño de sistemas software.

La lista de lenguajes de programación y, en los últimos tiempos también de *script*, ligados a la orientación a objetos es interminable. Por brevedad, en esta historia sólo se han incluido cuatro de los lenguajes más representativos y que más influencia han tenido para que la orientación a objetos sea lo que actualmente es: uno de los paradigmas de programación más usados y que se ha impuesto como uno de los pilares básicos en cualquier modelo de enseñanza de Informática.

2.2.1. Los años 60: Simula

Durante la década de los 60 se comenzó a percibir la necesidad de lenguajes con mayor potencia expresiva que facilitaran el desarrollo de los sistemas complejos que en esa época estaban comenzando a aparecer. A principios de esta década, los noruegos Kristen Nygaard y Ole-Johan Dahl comenzaron el desarrollo de Simula, un lenguaje cuyo principal propósito era servir para la descripción de sistemas complejos y programación de simulaciones (Nygaard y Dahl, 1981). Este lenguaje se basa en ALGOL y permite definir un sistema como un conjunto de procesos que interaccionan entre sí y que son ejecutados pseudo-paralelamente. Algunos de estos procesos pueden ser declarados colectivamente como actividades.

El refinamiento de Simula a un lenguaje de programación de propósito general conduce a estos autores al desarrollo de Simula-67. Esta generalización identifica la necesidad de añadir al lenguaje una serie de sofisticaciones que darán lugar a la definición de los dos términos más representativos de la orientación a objetos: objetos y clases. Los procesos de Simula son renombrados como objetos, y las actividades o definiciones de un tipo común de objetos pasan a ser conocidas como clases de objetos —o, abreviadamente, clases. Este lenguaje también soporta la ocultación de datos e implementación.

Además, observan que algunos objetos tienen atributos y acciones comunes pero que son estructuralmente diferentes en otros aspectos, por lo que han de ser definidos por clases diferentes. Para evitar la duplicidad del código, Simula-67 incluye la posibilidad de crear clases como especializaciones de otras clases llamadas subclases, de modo que las segundas contienen atributos ya declarados en las primeras. Esto es lo que inicialmente se conoció por herencia. Posteriormente se añadió la herencia no sólo de propiedades sino también de comportamiento, los procedimientos virtuales y la sobrecarga de métodos.

Los autores de Simula y Simula-67 han realizado tres aportaciones fundamentales a la orientación a objetos:

- Establecimiento de una nueva filosofía de desarrollo que centra la importancia en la comprensión y el modelado del problema.
- Definición de un incipiente vocabulario propio de la orientación a objetos que llega hasta nuestros días.
- Creación de las primeras implementaciones propias de la orientación a objetos —los tipos abstractos de datos y estructuras que son generadas por el compilador para poder ejecutar programas orientados a objetos.

2.2.2. Los años 70: SmallTalk

En la década de los 70 nace el lenguaje SmallTalk, desarrollado por Alan Kay (Kay, 1993). El desarrollo de SmallTalk comparte la misma motivación

que Simula: definir un lenguaje simple y expresivo que facilitara el diseño de sistemas complejos mediante módulos que ocultasen los detalles de más bajo nivel. A esto hay que añadir otra motivación más relacionada con las teorías cognitivas y educativas que por aquel momento abanderaba Minsky: SmallTalk debería ser un lenguaje que sirviera de ayuda a la educación. Minsky considera que a la hora de formar las habilidades cognitivas de una persona hay que tener en cuenta la forma en la que los seres humanos pensamos. Además, considera que los computadores pueden ser de gran ayuda para promocionar la forma en que pensamos, por lo que hay que fomentar su uso como metáfora para tratar con conocimiento complejo. Por otro lado, Kay ve el computador como un dispositivo de gran potencial creativo pero que no tiene el impacto esperado debido al modo de comunicación con el que el usuario se encuentra al hacer uso de él (el lenguaje de implementación). Teniendo en cuenta esto, Kay desarrolla SmallTalk como un lenguaje que ayuda en la educación y que facilite la interacción entre los niños y los computadores.

El diseño de SmallTalk se basa en el *diseño recursivo*, en la idea de que un sistema puede ser definido por un conjunto de partes que tienen la misma potencia que el conjunto y que pueden interaccionar intercambiándose mensajes. Kay afirma que "SmallTalk es una recursión de la noción de computador en sí misma, (...) cada objeto de SmallTalk es una recurrencia de todas las capacidades de un computador". SmallTalk nace bajo la influencia de los sistemas de ficheros de los computadores Burroughs 220 del ejército de los Estados Unidos (en los que cada fichero se dividía en los datos, los procedimientos para manipular dichos datos y un array de punteros que daban acceso a esos procedimientos), Sketchpad (un computador gráfico que permitía al usuario la creación de dibujos a partir de instancias de un dibujo "maestro" y definiendo unas restricciones modificaban dichas instancias) y el propio Simula.

SmallTalk fue principalmente usado para el desarrollo de aplicaciones gráficas. Pero, como Budd afirma en (Budd, 1993), la principal contribución de SmallTalk no es el lenguaje en sí sino que es el primero que define las bases del paradigma de la programación orientada a objetos. SmallTalk tiene seis características básicas, algunas de la cuales se conservan como propias de la orientación a objetos:

- En SmallTalk todo es un objeto (incluso las clases son objetos).
- Los objetos se comunican mediante el envío y la recepción de mensajes.
- Cada objeto tiene su propio espacio de memoria.
- Todo objeto es instancia de una clase.
- Las clases contienen el comportamiento que es compartido por todas sus instancias.

- Las clases se organizan en una estructura de árbol llamada jerarquía de herencia. La memoria y el comportamiento asociados a las instancias de una clase está disponible para cualquiera de sus clases descendientes —esta característica sólo fue incluida con las versiones más modernas de Smalltalk (76 y 80).

2.2.3. Los años 80: C++

En abril de 1979, Bjarne Stroustrup crea la primera versión del lenguaje "C with Clases", el predecesor de C++ (Stroustrup, 1993, 2007). Este lenguaje surge de la experiencia del autor en el desarrollo de su tesis doctoral usando Simula en la Universidad de Cambridge. Tras este trabajo, Stroustrup encuentra la necesidad de desarrollar un lenguaje que tenga la misma expresividad que Simula —en cuanto a clases, jerarquías de clases y comprobación estática de tipos—, la misma velocidad de ejecución que BCPL, la misma capacidad que tiene BCPL de combinar unidades que han sido compiladas por separado en un único programa, y que tuviese distintas implementaciones para ser portado fácilmente a otras máquinas. Para conseguir esto, desarrolla el pre-procesador Cpre, responsable de la compilación de "C with Classes", un lenguaje que añade a C las características de clases de Simula y que trata las clases definidas por los propios usuarios como tipos del lenguaje, lo que permite la comprobación estática de tipos. Stroustrup habla de este lenguaje como una herramienta que permite una mejor organización de los programas siguiendo la filosofía de Simula —esto es, la orientación a objetos—, garantizando la eficiencia, la flexibilidad y la portabilidad de C.

A partir de 1982, Stroustrup deja a un lado este preprocesador para crear un lenguaje propio que se separa de C: C++. Para ello construye el primer compilador de C++, llamado Cfront, usando su propio lenguaje "C with Classes". Aunque muchos hablan de C++ como un lenguaje orientado a objetos, lo cierto es que su propio autor lo define como un lenguaje multiparadigma que, además de soportar los estilos de programación procedimental y genérica usando plantillas, también soporta la programación orientada a objetos. En 1988 comienza la estandarización de C++, completándose como estándar ISO en 1998. Actualmente se está trabajando en una revisión de dicho estándar que, presumiblemente, concluirá en 2010. C++ es actualmente uno de los lenguajes más usados en el desarrollo de infinidad de aplicaciones, muchas de las cuales usamos a diario.

2.2.4. De los 90 a nuestros días: Java

El desarrollo de C++, la aparición de una serie extensiva de artículos sobre Smalltalk en la revista Byte y la celebración de la primera conferencia de programación orientada a objetos (la primera edición de OOPSLA, en el año 1986) hacen que la programación orientada a objetos se haga el

paradigma más popular entre los desarrolladores de sistemas. Esta popularidad se vio incrementada aún más si cabe con el desarrollo del lenguaje Java (Gosling et al., 2005; Byous). El predecesor de Java, Oak, nació debido a las deficiencias de C++ encontradas por los autores durante el desarrollo del proyecto Green. El propósito de este proyecto era la construcción de un sistema compuesto por múltiples electrodomésticos inteligentes que se pudieran comunicar entre sí y que fuesen controlados y programados a distancia a través de un dispositivo de control remoto. La creación de este sistema obliga a tener un lenguaje que ha de ser independiente de la plataforma. Aunque los primeros esfuerzos se centraron en la modificación de C++, éstos fueron en vano, por lo que se optó por la creación de un nuevo lenguaje. Éste se llama Oak y se caracteriza por ser un lenguaje interpretado, que se ejecuta sobre una máquina virtual y que es traducido a un código intermedio o *byte-code* que le permite que pueda viajar por la red y ser ejecutado —interpretado— en cualquier máquina de manera segura (ya que carece de punteros que posibilitan acceder a posiciones "prohibidas" de memoria). Posteriormente, Oak fue renombrado a Java. El escaso éxito de los electrodomésticos y el auge de Internet permitieron que los esfuerzos en el desarrollo de este lenguaje no cayeran en saco roto. Las características de Java (independencia de la plataforma, seguridad y fiabilidad) hicieron que encajara perfectamente en el desarrollo de software para Internet. Para demostrarlo, en 1994 se desarrolla WebRunner, un navegador implementado en Java y que sería el predecesor de HotJava. La presentación de HotJava, en el año 1995 y el soporte a la ejecución del código Java que brindaron Netscape, el navegador más popular en aquella época, y Microsoft fueron el último espaldarazo a la hegemonía de Java en el desarrollo de aplicaciones Web.

2.3. Conceptos de la orientación a objetos

La orientación a objetos supone que el desarrollador ha de conocer un conjunto de conceptos e ideas que usará durante el diseño y la implementación de un sistema software. En esta sección vamos a definir cuáles son esos conceptos que son necesarios para tener una adecuada formación en la orientación a objetos. Primeramente definiremos cuáles son los conceptos elementales de la orientación a objetos. Posteriormente pasaremos a describir las nociones básicas que se han de tener sobre el diseño orientado a objetos. Finalmente veremos un concepto más avanzado pero que es considerado como imprescindible para el conocimiento de la orientación a objetos: los patrones de diseño.

2.3.1. Conceptos elementales

Si revisamos la literatura relacionada con la orientación a objetos vemos que prácticamente cada autor tiene una lista de los conceptos relacionados con la orientación a objetos. Por ejemplo:

- Jacobson (Jacobson et al., 1992) considera que los conceptos fundamentales de la orientación a objetos son los objetos, las clases, el polimorfismo y la herencia.
- El modelo de objetos descrito en (Booch, 1994) se compone de cuatro elementos fundamentales (abstracción, encapsulación, modularidad y jerarquía) y tres elementos menores (tipos, concurrencia y persistencia).
- En (Gosling y McGilton, 1996) se afirma que un lenguaje orientado a objetos ha de soportar, como mínimo, encapsulación, polimorfismo, herencia y vinculación dinámica.
- Meyer (Meyer, 1997) afirma que un método y lenguaje orientado a objetos ha de dar soporte a términos como clases, aserciones, paso de mensajes, ocultación de información, tratamiento de excepciones, comprobación estática de tipos, genericidad, herencia (simple y múltiple), redefinición, polimorfismo, vinculación o ligadura dinámica, gestión automática de memoria y recolección de basura.
- El curriculum de informática desarrollado por ACM (ACM, 2001) incluye dentro de la asignatura *PL6. Programación orientada a objetos* los siguientes conceptos: diseño orientado a objetos, encapsulación y ocultación de la información, separación de comportamiento e implementación, clases y subclases, herencia, polimorfismo, jerarquías de clases, colecciones y protocolos de iteración y representación interna de tablas de objetos y de métodos.
- Budd recoge en (Budd, 2002) las ideas fundamentales de la orientación a objetos: objetos, paso de mensajes, métodos, responsabilidades, clases e instancias, jerarquías de clases o herencia, polimorfismo y redefinición.
- En (West, 2004) se define un vocabulario propio de la orientación a objetos que clasifica los términos en esenciales (objeto, responsabilidad, mensaje y protocolo o interfaz), términos de extensión (clase, colaboración, jerarquía de clases, abstracto, herencia, delegación, polimorfismo, encapsulación, componente, framework y patrón), términos de implementación (método, variable y vinculación dinámica) y otros términos auxiliares.

Como se puede ver, es difícil enseñar una disciplina en la que no hay un consenso sobre cuáles son sus conceptos elementales o en la que hay tantas diferencias terminológicas para referenciar un mismo aspecto. En lugar de quedarnos con uno de ellos vamos a utilizar el estudio de Deborah Armstrong (Armstrong, 2006), que realiza una revisión exhaustiva de la literatura en orientación a objetos para generar una lista con los 8 conceptos que son más comúnmente utilizados como conceptos elementales de la orientación a objetos. Estos conceptos son los siguientes: abstracción, encapsulación, objeto, clase, paso de mensaje, método, herencia y polimorfismo.

2.3.1.1. Abstracción

La abstracción es un término de los años 50 que es aplicable al desarrollo de software clásico pero que es de gran importancia en la orientación a objetos. La abstracción es la supresión u ocultación de detalles de un proceso o artefacto con el fin de mostrar más claramente otros aspectos, detalles o estructuras. Desde el punto de vista de la orientación a objetos es el mecanismo que permite crear un modelo simplificado de la realidad, que elimina las distinciones entre objetos para crear las clases que los engloban de manera común.

La abstracción es un concepto muy importante en la orientación a objetos ya que separa el comportamiento de los objetos de su implementación. Esto facilita el desarrollo de sistemas extensibles y de módulos reutilizables, como veremos en la Sección 2.3.2.

2.3.1.2. Encapsulación

Este es un concepto que se considera anterior a la orientación a objetos aunque algunos autores afirman que fue empleado por primera vez en Simula. Es muy similar a la ocultación de información. De hecho, ciertos autores no encuentran diferencia alguna entre ambos términos. La encapsulación es una técnica para el diseño de clases y objetos que restringe el acceso a los datos y al comportamiento, definiendo el conjunto de mensajes que un objeto de la clase es capaz de recibir.

Es importante añadir que la encapsulación también es la responsable de la ocultación de los detalles de implementación de las clases y que mantiene toda la información de la clase (datos y comportamiento) cohesionada.

2.3.1.3. Objeto

Este término aparece por primera vez en Simula-67 y es la base de la orientación a objetos. Los sistemas software se descomponen en objetos que se responsabilizan de la ejecución de unos determinados servicios. Un objeto es una entidad con un comportamiento bien definido y que se caracteriza por

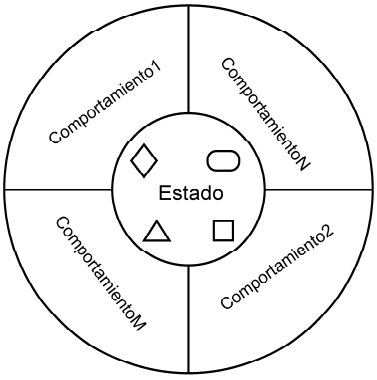


Figura 2.1: Representación de un objeto: el estado se encuentra encapsulado y sólo es manipulable a través del comportamiento.

su estado, su comportamiento y su identidad (Booch, 1994). Generalmente, un objeto suele representarse usando un esquema como el que aparece en la Figura 2.1, conocido como "el balón de fútbol". En ella se destaca la encapsulación, de modo que la única forma de llegar al estado de un objeto es a través de la capa de comportamiento que el propio objeto define.

Un objeto contiene información sobre su estado. El estado se define por el valor actual de las propiedades contenidas en el objeto, también conocidas como atributos. Esta información, a la que el propio objeto tiene acceso, es la que le ayudará a completar la realización de un servicio. El estado puede contener tanto los datos necesarios para realizar el servicio como referencias a otros objetos a los que solicitar la información que necesita o que le ayuden a completar el servicio solicitado.

El comportamiento es la manera en la que un objeto actúa y reacciona de acuerdo a su estado y al servicio solicitado. El comportamiento es definido mediante responsabilidades, es decir, los servicios que un objeto ofrece y que son traducidas a acciones. Estas acciones pueden realizarse sobre el objeto mismo, para cambiar su estado, o sobre otros objetos, delegando así parte de la tarea para llevar a cabo el servicio solicitado. Estas acciones suelen permitir examinar los datos del objeto o modificar los mismos. Dos objetos de la misma clase pueden tener comportamientos distintos debido a que el comportamiento depende del estado actual de cada uno de los objetos.

Un objeto tiene identidad propia y es una instancia de una clase. Suele tener un identificador único y éste suele ser descriptivo del objeto que identifica.

Desde el punto de vista de la metáfora de la antropomorfización (West, 2004), un objeto es un agente capaz de proveer de unos determinados servicios a una comunidad. Tiene acceso a unos determinados conocimientos internos, que usa para responder a las peticiones de servicio. Los objetos,

como las personas, se especializan y son *vagas*, por lo que suelen acudir a otros grupos de objetos para distribuir el trabajo.

Existen dos tipos fundamentales de relaciones entre objetos: cliente y herencia. La primera se produce cuando un objeto hace uso de otro objeto para completar alguna de sus responsabilidades. La segunda es otro de los conceptos básicos de la orientación a objetos y será tratada más adelante.

2.3.1.4. Clase

Al igual que el anterior, este concepto aparece por primera vez en Simula-67. Una clase es una abstracción que describe la estructura y el comportamiento compartidos por uno o más objetos similares. Es el elemento básico para el modelado de un sistema orientado a objetos. Un objeto es una instancia de una clase. En ejecución, la clase describe el comportamiento de los objetos y es la fuente para la creación de nuevos objetos; durante el desarrollo, describe la interfaz que los programadores usarán para interactuar con los objetos de dicha clase.

El conocimiento sobre un objeto, ya sea tanto su estructura como su comportamiento, se basa en su apariencia pública, esto es, la clase a la que pertenece y el protocolo o servicios que ofrece.

Las clases se pueden relacionar entre sí de forma que los objetos que instancian dichas clases puedan colaborar unos con otros para contribuir al comportamiento de un sistema. Existen tres clases principales de relaciones (Meyer, 1997):

- Generalización/especialización: Esta relación se define mediante el concepto de herencia, que se trata más adelante.
- Parte/todo. También se conoce como composición o agregación y denota la relación estructural entre una clase que representa un todo, y un conjunto de clases que denotan sus partes. Por ejemplo, una clase `Automóvil` (un todo) puede presentar relaciones de este tipo con las clases `Rueda`, `Motor` o `Transmisión`, entre otras.
- Asociación: Representa una relación semántica entre dos clases que aparentemente no tienen relación estructural entre ellas. Por ejemplo, la clase `Vehículo` puede tener una relación de asociación con la clase `Calzada` a pesar de que un vehículo no se compone de "calzadas" ni viceversa.

2.3.1.5. Paso de mensajes

Es el proceso de comunicación formal por el que un objeto solicita a otro un servicio. El mensaje contiene la información sobre la petición que se solicita. El receptor, si acepta el mensaje, es responsable de llevar a cabo

la acción asociada al mismo. Para responder al mensaje, el objeto receptor invocará la ejecución de un método que satisfaga la petición.

Algunos autores no ven diferencia con lo que sería la llamada a un procedimiento. Sin embargo, existe una diferencia clara: un procedimiento realiza una acción sobre unos datos, mientras que el paso de mensajes realiza una petición a otro objeto para realizar una acción sobre unos datos. Es importante destacar que la relación entre ellos se basa en la delegación que hace el primer objeto, que no tiene los datos ni sabe como manipularlos, sobre el segundo, que, o tiene y puede realizar acciones sobre los datos o sabe cómo delegar sobre terceros objetos para manipularlos.

Un mensaje se caracteriza por el receptor, un selector que define el significado del mensaje, unos argumentos que contienen la información que el emisor envía al receptor del mensaje para realizar el servicio, y el objeto retornado, que contiene información sobre el resultado de la ejecución de la petición.

Hay que destacar el papel clave del receptor. Éste es el encargado de interpretar el mensaje y, por tanto, de decidir qué método va a ser usado en respuesta al envío de dicho mensaje. Hay que tener en cuenta que dicha interpretación depende del objeto receptor, de modo que el mismo mensaje puede ser interpretado de distinta forma por distintos objetos, ya sean de la misma o de distintas clases.

2.3.1.6. Método

Este concepto ha existido desde hace mucho tiempo en el ámbito del software. Aunque también es conocido por el nombre de procedimiento, acción u operación, el concepto método ha quedado ligado a la orientación a objetos con la llegada de SmallTalk. Un método es el mecanismo que nos permite acceder, fijar y manipular la información de un objeto. Es el bloque de código que es ejecutado en respuesta a un mensaje. Es uno de los elementos fundamentales de la orientación a objetos ya que son los responsables del paso de mensajes a otros objetos.

2.3.1.7. Herencia

Este término aparece por primera vez en Simula-67 y hay autores que lo consideran como la única contribución propia que hace la orientación a objetos a los paradigmas de programación. La herencia se puede definir como un mecanismo que permite que los datos y el comportamiento de una clase (superclase) estén incluidos o sean usados como la base de otra clase (subclase). Realmente, la herencia también es el mecanismo que soporta la abstracción y el polimorfismo, ya que la herencia posibilita que las subclases puedan ser tratadas de igual forma que las superclases.

Analizando la descripción anterior podemos ver que la herencia tiene una

perspectiva dual (Meyer, 1997) que hace que sea tenida en consideración desde dos puntos de vista:

- Desde el punto de vista de la reutilización, la herencia permite que una clase se defina en términos de otra para no tener que repetir el código que se encuentra en la superclase.
- Desde el punto de vista de la extensibilidad, la herencia facilita que una subclase pueda redefinir, ampliar o restringir el comportamiento de su superclase y que pueda ser usada en cualquier lugar en la que es usada ésta.

La herencia puede ser usada con múltiples propósitos (Budd, 2002):

- Para modificar el comportamiento de una clase, redefiniendo algunos de sus métodos.
- Para extender el comportamiento de la superclase, añadiendo nuevos métodos en las subclases.
- Para limitar el comportamiento de la superclase, haciendo que el comportamiento de las subclases sea más restrictivo.
- Para especificar el comportamiento de una clase en el caso de que la superclase no defina ningún comportamiento sino tan solo la interfaz común a un conjunto de subclases.
- Para la implementación del comportamiento de una subclase en base al de su superclase (Ej. Una clase `Diccionario` puede heredar de la clase `TablaHash` para hacer uso del comportamiento de la misma, aunque la primera nunca vaya a ser un subtipo de la segunda).
- Para combinar el comportamiento y la implementación de varias superclases. Como se verá más adelante, esto se conoce como herencia múltiple.

Aunque herencia y encapsulación son conceptos imprescindibles en la orientación a objetos ambos son antagónicos. Si la encapsulación oculta la representación interna de una clase, la herencia es la responsable de romperla, ya que deja que las subclases puedan tener acceso a las partes internas de su superclase. Para reducir esta ruptura de la encapsulación, muchos lenguajes de programación añaden el concepto de visibilidad, que establece qué mensajes y atributos son accesibles para otras clases. De esta forma, una superclase puede limitar el acceso a determinados métodos y atributos a sus subclases, conservando parte de la encapsulación.

Las principales ventajas de la herencia están relacionadas con su propio uso. La herencia facilita la creación de nuevas clases ya que reutiliza la

estructura y el comportamiento de las superclases. Además, hace que la modificación y extensión del comportamiento de una clase sea sencillo. Pero hay que tener en cuenta que ésta impone una relación muy fuerte entre clases. Además de romper la encapsulación, el comportamiento heredado se fija de manera estática, de modo que éste no puede ser cambiado en tiempo de ejecución. Además, cualquier cambio en las superclases obliga también a la revisión y modificación de todas sus subclases.

Existe un caso especial de herencia que no está soportado por todos los lenguajes orientados a objetos que es la herencia múltiple, es decir, una clase hereda no sólo de una sino de varias clases base. Este tipo de herencia es útil para la construcción de software robusto mediante la combinación de distintas abstracciones. Aunque útil, no es soportada por todos los lenguajes de programación ya que, mal usada, genera problemas de ambigüedad y de herencia repetida, también conocida como *herencia en diamante*.

2.3.1.8. Polimorfismo

El polimorfismo es la habilidad que presentan distintas clases para responder al mismo mensaje, de modo que cada una de ellas implementa el método invocado por el mensaje de la manera apropiada. Se fundamenta en el hecho de que es el receptor, y no el emisor, el responsable de interpretar un mensaje. Implica que el emisor de un mensaje no necesita conocer cuál es la clase del objeto al que se pasa el mensaje; tan solo se sabe que este objeto es capaz de responder a dicho mensaje.

El polimorfismo hace uso de los mecanismos de vinculación, tanto estáticos como dinámicos, pero no hay que confundir el concepto de polimorfismo en sí mismo con dichos mecanismos. Los últimos son el medio para llevar a cabo el polimorfismo. Éste es responsable de ocultar distintas implementaciones bajo una interfaz común y hace que distintos objetos puedan responder al mismo mensaje produciendo un comportamiento distinto.

El polimorfismo es un mecanismo muy importante en la flexibilidad que los sistemas orientados a objetos ofrecen ya que facilita la modificación de un comportamiento mediante la creación de nuevas clases, sin tener que realizar modificaciones sobre las ya existentes.

2.3.2. Conceptos de diseño orientado a objetos

El diseño orientado a objetos es un método que se basa en la descomposición de un sistema en clases de objetos. El diseño orientado a objetos también engloba la notación para describir los modelos lógicos y físicos, tanto estáticos como dinámicos, del sistema que se está diseñando (Booch, 1994).

Es complicado hablar de diseño orientado a objetos sin explicar los métodos y procesos de la ingeniería del software usados para el desarrollo de sistemas orientados a objetos. Consideramos que, en lo que sería la enseñan-

za de la orientación a objetos, el alumno ha de adquirir los conocimientos básicos de diseño orientado a objetos, pero se dejará para otras disciplinas los procesos de ingeniería para el desarrollo de software orientado a objetos, como el diseño por contrato (Mitchell et al., 2002), el diseño dirigido por responsabilidades (Wirfs-Brock y McKean, 2003) o el proceso unificado de modelado (Jacobson et al., 1999), así como las arquitecturas y los instrumentos de modelado.

El diseño orientado a objetos es una tarea de descomposición y composición. El desarrollador primero ha de estudiar el sistema para descomponer éste en un conjunto de módulos. Posteriormente, ha de determinar las relaciones que van a establecerse entre estos módulos y cómo se va a producir la comunicación entre ellos, es decir, cómo se van a componer para hacer funcional el sistema. Esta tarea producirá los objetos que compondrán el sistema y las clases que definen dichos objetos.

A nivel de clase y objeto, la calidad de un diseño se mide de acuerdo a dos conceptos fundamentales:

Acoplamiento Mide la fuerza de la asociación establecida entre dos clases u objetos. Establece la relación entre las operaciones y los datos de distintas clases. Cuanto mayor es el acoplamiento entre dos o más clases más complicado será la comprensión y el mantenimiento de dichas clases por separado ya que éstas se encontrarán muy interrelacionadas.

Cohesión Mide el grado de conectividad de los elementos que conforman una clase o un objeto. A diferencia de la anterior, establece la relación entre los datos y las operaciones dentro de una clase. Se espera que la cohesión sea lo mayor posible ya que facilita la comprensión y el mantenimiento de las clases.

En cualquier diseño orientado a objetos hay que tender a reducir el acoplamiento al mínimo, maximizando la cohesión. Pero hay que tener en cuenta que es necesario que exista un mínimo acoplamiento ya que, si no, se tiene un diseño compuesto por objetos totalmente independientes. Hay que recordar que, generalmente, un objeto por sí solo no sirve de nada.

Se han catalogado un conjunto de principios de diseño orientado a objetos que promueven la reducción del acoplamiento y la mejora de la cohesión, facilitando el diseño de software de mayor calidad (Martin, 2002). Los principales principios son los siguientes:

Principio abierto-cerrado *The Open-Closed Principle* enuncia que un módulo ha de estar abierto a la extensión pero cerrado a la modificación. Este es el más importante de los principios y ya fue enunciado por Bertrand Meyer en (Meyer, 1997). Este principio implica que una clase ha de ser diseñada de tal forma que se pueda cambiar su comportamiento sin necesidad de modificar el código antiguo sino extendiendo dicha

clase y añadiendo nuevo código. Este principio se basa en el uso de la abstracción, la herencia y el polimorfismo. Aunque no es posible que todas las clases satisfagan este principio hay que esforzarse en minimizar el número de las que no lo cumplen.

Principio de sustitución de Liskov *The Liskov Substitution Principle* fue enunciado por Barbara Liskov (Liskov y Wing, 1994) en base a los trabajos de diseño por contrato de Meyer (Meyer, 1997). Afirma que toda subclase ha de poder sustituir a su superclase. Este principio sostiene que la relación de herencia es una relación que se ha de definir en términos de comportamiento, y que una subclase siempre ha de satisfacer, al menos, el comportamiento que los clientes esperan de su superclase. Garantiza que una subclase puede ser usada allá donde su superclase es usada, favoreciendo la extensibilidad.

Principio de inversión de dependencia *The Dependency Inversion Principle* afirma que es conveniente diseñar basándose en abstracciones en lugar de basarse en implementaciones concretas. También se resume como que las abstracciones no deben depender de los detalles sino que éstos han de depender de las abstracciones. Durante el diseño es muy común crear primero las clases de más bajo nivel para posteriormente diseñar las de alto nivel, que se definen en términos de las primeras. Este diseño no es flexible ya que las modificaciones en las clases de más bajo nivel obliga a cambiar también las de más alto nivel. Es mejor añadir una capa de abstracción que separa las clases de alto de las de bajo nivel. Las clases de alto nivel se crean usando la capa de abstracción mientras que las de bajo nivel son creadas en base a la abstracción. Este principio también se ha enunciado en ocasiones como "Programar (o diseñar) para interfaces, no para implementaciones" (Gamma et al., 1995)

Principio de Segregación de interfaces *The Interface Segregation Principle* enuncia que es mejor varias interfaces específicas para distintos clientes que una única de propósito general. Si una clase es usada por distintos clientes, en lugar de tener un único interfaz con los métodos de todos los clientes, es preferible tener una interfaz específica para cada uno de los clientes y usar herencia múltiple. Tampoco hay que excederse a la hora de aplicar este principio de diseño y crear una interfaz para cada posible cliente de una clase ya que complicaría en gran medida el diseño.

Principio de responsabilidad única *The Single-Responsibility Principle* indica que sólo ha de haber una razón por la que modificar una clase. En este contexto, una responsabilidad se define como una razón para cambiar. Cuantas más responsabilidades tenga una clase, más posi-

bilidades de cambio tiene, afectando además a clientes que no tienen que ver con las responsabilidades modificadas. Éste es tal vez el más sencillo de los principios pero uno de los más complicados de aplicar correctamente.

De cara a la realización de buenos diseños, es importante que el alumno aprenda a percibir indicios que le pueden indicar que un diseño no es correcto o, al menos, es mejorable. Estos indicios suelen reflejarse en el código fuente de una aplicación y es lo que en (Fowler, 1999) se conoce con el nombre de "malos olores" (del inglés, *bad smells*). A continuación se realiza un resumen del catálogo de "malos olores" que se pueden encontrar en el código de una aplicación y que han sido descritos en (Fowler, 1999) y en (Kerievsky, 2004):

Código duplicado Un signo de un mal diseño es encontrarse con fragmentos de código que se repiten en más de un lugar, ya sea dentro de una misma clase, en clases relacionadas por herencia o en dos clases que no se encuentran relacionadas. Un caso especial de código duplicado es lo que se conoce como explosión combinatoria, en la que hay partes de código que realizan las mismas operaciones pero usando distintos tipos y cantidades de datos. Aunque cada uno de estos trozos de código se refactorizara agrupándolo en un método, el número de métodos distintos seguiría siendo muy grande, por lo que se requerirán técnicas más avanzadas de refactorización. Otro caso especial de duplicidad de código se produce cuando un mismo problema se resuelve en distintas partes del código de maneras distintas. Para facilitar su mantenimiento será necesario decidir cuál de las soluciones se va a usar y sustituir las descartadas por ésta.

Clases y métodos largos Generalmente, una clase que tiene demasiados atributos suele tener código duplicado y no suele usar todos estos atributos al mismo tiempo. Por otro lado, las clases con demasiadas responsabilidades son difíciles de mantener y suelen tener código duplicado. En otras ocasiones, la excesiva extensión de una clase se debe a la presencia de métodos largos. Éstos suelen ser difíciles de comprender, de mantener y dificultan su reutilización. Además, un método puede dar señales de no haber sido diseñado correctamente si la lista de sus parámetros es larga. Además, métodos que necesitan gran cantidad de información externa para su ejecución suelen ser sinónimo de una mala asignación de responsabilidades.

Consecuencias de cambios funcionales Este "mal olor" agrupa dos hechos opuestos pero relacionados con los cambios funcionales. Por un lado, un indicativo de mal diseño es que una clase sufra frecuentes cambios debidos a diferentes razones. Esto suele indicar que la funcionalidad de dicha clase debería ser distribuida en distintas clases, de

modo que cada una de ellas se vea afectada por un determinado tipo de cambios. Por otro lado, si un cambio funcional hace que sea necesario realizar cambios en un gran número de clases diferentes, entonces el mantenimiento y la modificación del sistema suele ser complicado ya que es difícil encontrar cuáles son todos los puntos que hay que cambiar para realizar dichas modificaciones. En un buen diseño, un cambio que produce una variación del comportamiento de la aplicación debería tan solo afectar a una clase. Dentro de este "mal olor" también se engloban las jerarquías paralelas, en las que las modificaciones o inclusiones de nuevas subclases en una jerarquía hacen que sea necesario modificar o incluir clases similares en otra jerarquía, lo que implica que se está realizando una duplicación que dificulta el mantenimiento de la aplicación.

Individuos y grupos de datos primitivos En muchas ocasiones se suelen usar datos primitivos que luego han de sufrir transformaciones a lo largo del código. Por ejemplo, una variable que representa una cantidad de dinero representada como un real puede sufrir transformaciones de acuerdo al tipo de moneda en el que está representado o puede necesitar ser convertido a una cadena de caracteres para ser impresa en un recibo. Esto suele ser indicativo de que estos datos han de ser encapsulados en un objeto que conozca cómo realizar estas transformaciones. Por último, se pueden observar grupos de datos que siempre aparecen juntos a lo largo de todo el código. Esto suele indicar que dichos datos deberían quedar encapsulados formando una clase.

Estructuras condicionales complejas Este tipo de código también es conocido como "código spaghetti". Generalmente, un código orientado a objetos suele tener un número menor de sentencias condicionales que un código en el paradigma de programación estructurada. Si, además, estas sentencias condicionales se repiten a lo largo del código entonces es signo de que probablemente no se esté haciendo uso del polimorfismo. En ocasiones se hace una utilización indebida de tipos de datos primitivos para representar el código de un tipo. Esto produce que en el código aparezcan sentencias condicionales que, en función de este código de tipo, decidan un determinado comportamiento. Generalmente esto indica que no se está realizando un buen diseño orientado a objetos, ya que deberían ser sustituidos por la utilización de clases polimórficas.

Clases especulativas, vagas y alternativas Las primeras son clases que intentan adelantarse a todas las posibles variaciones futuras que se puedan producir (o no) en una clase. Estas clases se vuelven demasiado generalistas, volviendose ininteligibles e inmantenibles, por lo que suele ser recomendable eliminarlas y colapsar las jerarquías de clases que generan. Las segundas son clases que tienen una mínima responsa-

bilidad. Cada clase tiene un coste de mantenimiento y cada objeto de esta clase tendrá un coste en ejecución. Por tanto, es necesario eliminar estas clases y distribuir su comportamiento entre otras que estén capacitadas para llevarlo a cabo. Las últimas son clases cuyo comportamiento y responsabilidades son similares pero sus interfaces son distintas. Estas clases deberían ser refactorizadas para unificar su interfaz, ya sea compartiendo un mismo interfaz o creando nuevas clases que adapten un interfaz a otro.

Cadenas de mensajes Se produce cuando una clase navega a través de una serie de relaciones de clases para alcanzar una determinada responsabilidad. Esta cadena de mensajes hace que la clase se encuentre acoplada a cada una de las que conforman dicha cadena. Además, suele romper la encapsulación de las clases que forman parte de la cadena. Generalmente este "mal olor" se debe al mal uso de la delegación de responsabilidades en otros objetos.

Clases contenedoras de datos Existen clases que han sido diseñadas con una única finalidad: contener datos. Suelen ser equiparables a los registros de la programación estructurada. En un pésimo diseño estas clases no están encapsuladas mediante métodos accesores. Generalmente este "mal olor" suele ser signo de que se ha hecho un mal reparto de responsabilidades, ya que habrá otras clases que accedan continuamente a sus datos privados para manipularlos y realizar operaciones con ellos, responsabilidad que debería pertenecer a la clase contenedora.

2.3.3. Patrones de diseño

Los patrones de diseño orientado a objetos son pequeñas perlas que concentran un conocimiento profesional del diseño de software. Un patrón de diseño describe un problema recurrente en el diseño de software y proporciona los aspectos claves de una solución reusable a este problema. El concepto de patrón de diseño tiene sus raíces en la arquitectura, en los trabajos de Christopher Alexander (Alexander et al., 1977). Más tarde, este concepto fue adoptado por la comunidad informática y adaptado a sus necesidades, dando lugar a un conjunto de artefactos que promueven la reutilización a nivel de diseño (mucho más potente que la reutilización de código) y que constituyen un vocabulario esencial para la comunicación entre los integrantes de los equipos de desarrollo de software. Aunque existen diferentes tipos de patrones de diseño software (patrones para el paradigma imperativo, patrones para programación distribuida, anti-patrones...), los currículos de enseñanza de informática centran su interés en los patrones de diseño orientado a objetos, como los catalogados por Gamma, Helm, Johnson y Vlissides (Gamma et al., 1995).

En general, un patrón de diseño tiene cuatro elementos esenciales:

- El nombre del patrón. Nos permite comunicar un problema de diseño y su solución en una o dos palabras. El nombre del patrón da a sus usuarios un vocabulario con el que compartir información de diseño y a un mayor nivel de abstracción.

- El problema. Define el contexto en el que se va a aplicar el patrón de diseño.

- La solución. Es una descripción en abstracto de los elementos que compondrán la solución al problema descrito y la relación entre estos elementos. Al ser una solución abstracta, podrá ser reutilizada en problemas concretos de diseño.

- Las consecuencias. El uso del patrón de diseño acarrea una serie de consecuencias, tanto positivas como negativas, que han de ser tenidas en cuenta a la hora de evaluar distintas alternativas de diseño.

En los patrones de diseño orientados a objetos la solución identifica las clases que van a participar en la solución así como sus instancias (objetos), cuáles son los roles que van a interpretar cada una de ellas, cómo se relacionan y cuáles son las colaboraciones e interacciones que se producen entre ellas para alcanzar la solución del problema descrito. Los patrones de diseño están a caballo entre el diseño y la implementación, por lo que en (Gamma et al., 1995) se ha optado por acompañar la solución con ejemplos concretos de implementación en C++ y Smalltalk.

Los patrones de diseño son esenciales para el desarrollo de buenos diseños en sistemas complejos. Algunos autores (Astrachan et al., 1998; Alphonce y Ventura, 2002) consideran que son imprescindibles ya que ayudan a sobrellevar la complejidad de desarrollar software reutilizable. Además, los patrones sirven como ejemplo de buenos diseños, ya que provienen de los conocimientos de desarrolladores expertos. Los patrones sirven para transmitir esta sabiduría a los principiantes en la orientación a objetos (Chambers et al., 2000). El conocimiento de los mismos también ayuda al trabajo futuro con frameworks y librerías y a sacar el mayor partido de ellas, ya que suelen hacer un uso exhaustivo de los patrones.

Debido a su naturaleza, a caballo entre el diseño y la implementación, los patrones también sirven como ejemplo para el traslado de las buenas ideas de diseño a la implementación de un sistema (Clancy y Linn, 1999). No sólo explican las ideas que conducen a un diseño reutilizable sino que también permiten que el alumno vea cómo se trasladan los conceptos de diseño que favorecen la reutilización a un modelo de implementación concreto.

2.4. Dificultades en la enseñanza de la orientación a objetos

A continuación se va a profundizar en el estudio de las principales dificultades encontradas a la hora enseñar orientación a objetos. Una revisión de la literatura relacionada con la orientación a objetos y la enseñanza de la misma, además de la experiencia del propio autor, ha conducido a la inclusión en este estudio de las dificultades que se enumeran a continuación. Se ha intentado clasificar estas dificultades de acuerdo a los conceptos elementales expuestos en la sección anterior. Esta clasificación no es sencilla, ya que determinados problemas solapan y salpican distintos conceptos de la orientación a objetos. Además, existen dificultades relacionadas propiamente con la forma en la que enseña este paradigma, más que con los conceptos inherentes a la orientación a objetos.

2.4.1. El lenguaje

El principal problema de este paradigma es que enseñar orientación a objetos no consiste únicamente en enseñar un lenguaje orientado a objetos, cosa que algunos docentes olvidan. Enseñar orientación a objetos consiste en inculcar al alumno una determinada manera de pensar cuando se enfrenta a un problema de programación o de diseño.

El uso de un determinado lenguaje para ejemplificar algunos de los conceptos es un medio útil para enseñar este paradigma. Pero el lenguaje en sí mismo —su sintaxis, sus particularidades de implementación— no puede ser el centro del proceso de enseñanza.

A esto hay que añadirle los problemas de terminología (Bacvanski y Börstler, 1997). Como ya se ha podido ver en la Sección 2.3.1, se ha puesto de manifiesto que es complicado definir cuáles son los conceptos básicos. Además, distintos autores usan un vocabulario distinto, de modo que conceptos similares se identifican con distintas palabras. Esto complica el uso de diferentes recursos relacionados con la orientación a objetos. Más aún, pueden aparecer nuevos problemas en el caso de alumnos que han recibido una formación previa en este paradigma y con una terminología distinta a la usada por el docente actual.

2.4.2. El uso de ejemplos

Otro de los problemas descritos en la literatura es el uso de ejemplos que no son adecuados para la enseñanza de este paradigma. Hay que insistir en que la orientación a objetos es especialmente relevante para el desarrollo de software a gran escala. Algunos autores afirman que muchos de los ejemplos empleados son demasiado sencillos y que no recogen la esencia de la orientación a objetos (Bacvanski y Börstler, 1997). Con estos ejemplos es

difícil motivar algunas de las ideas que han favorecido la expansión de la orientación a objetos, como la extensibilidad y la reutilización.

Prácticamente desde que Nygaard propusiera el uso de ejemplos complejos para la enseñanza de la programación a objetos (Nygaard, 2001; Berge et al., 2003), se produjo una proliferación de este tipo de ejemplos. De entre las numerosas propuestas de este tipo de ejemplos (de entre la que se destaca el ejemplo del restaurante, propuesto por el propio Nygaard), caben destacar las aplicaciones que se conocen como casos de estudio. Un caso de estudio es una aplicación software completa que incluye una explicación sobre el proceso de desarrollo que ha llevado a su consecución y preguntas dirigidas al alumno para que éste analice, juzgue y evalúe distintos aspectos de su diseño (Clancy y Linn, 1999).

Uno de los casos de estudio más extensamente usado y citado en enseñanza de orientación a objetos es el simulador de biología marina de College Board (MBSCS), que se describirá en profundidad en la Sección 3.5. Este caso de estudio propone una serie de ejercicios y preguntas para que el alumno se familiarice con el caso de estudio, para posteriormente extenderlo añadiendo nueva funcionalidad y nuevos tipos de entidades (en este caso, peces). Este caso de estudio combina clases de código público con el uso de clases de caja negra, de las que se conoce su interfaz pero de las que se desconoce su implementación. Estas últimas son empleadas para que el alumno se familiarice con la reutilización, mecanismo muy común en la orientación a objetos. Aunque durante el curso 2007/2008 este caso de estudio ha sido sustituido por GridWorld (GridWorld), el tipo de ejercicios y preguntas asociadas siguen siendo las mismas.

Los cuestionarios que acompañan a estos casos de estudio fomentan la actividad del alumno y hacen que éste se haga consciente de si está comprendiendo correctamente el diseño y el funcionamiento de la aplicación. También se ligan bastante al código fuente y proponen al alumno realizar modificaciones y extensiones sobre el mismo. Realmente, los casos de estudio aquí descritos se encuentran demasiado ligados al código (por ejemplo, el simulador de biología marina fue creado inicialmente en C++ pero, debido a las recomendaciones de uso de Java para la enseñanza de la programación orientada a objetos, fue traducido a este lenguaje), por lo que fijan los conceptos de orientación a objetos al lenguaje en el que están implementados (en este caso, Java).

2.4.3. Encapsulación y abstracción

Centrar el proceso de enseñanza de la orientación a objetos en aprender a usar un lenguaje de programación afecta principalmente a la comprensión de la abstracción. La orientación a objetos implica la comprensión y la abstracción de un sistema. El uso de un lenguaje concreto a este nivel liga el sistema a los detalles de su implementación. El alumno ha de comprender que

la orientación a objetos supone la descomposición de un sistema en objetos, da igual la forma en la que posteriormente éstos sean implementados.

La abstracción es un concepto cuyo aprendizaje puede verse favorecido por el uso de visualizaciones en lugar de utilizar un lenguaje de programación. Generalmente, la mente humana no es buena almacenando conceptos abstractos por lo que una imagen mental refuerza la forma en la que dicho concepto queda almacenado en nuestra mente. Por ejemplo, los desarrolladores suelen construir un mapa mental de su conocimiento que luego plasman en representaciones gráficas y diagramas (Ryan et al., 1997). Al igual que éstos, es interesante que la enseñanza de conceptos como la abstracción o el diseño, como se verá más adelante, se vea apoyada por representaciones visuales que clarifiquen los conocimientos que está adquiriendo el alumno y eviten las falsas suposiciones que pueda estar creando en su interior.

Uno de los malos hábitos más comunes es la ruptura de la encapsulación. Es necesario inculcar al alumno que hay que minimizar la accesibilidad de las clases. De esta forma será más fácil cambiar la implementación de una clase sin modificar el código que haga uso de dicha clase. Un caso grave de ruptura de la encapsulación es permitir que se pueda acceder directamente al estado de un objeto. Si esto es estrictamente necesario, hay que promover el uso de métodos accedentes y mutadores en lugar de dar acceso directo a los atributos de un objeto. Estos métodos permiten añadir restricciones a los valores válidos para un atributo. Además, permiten realizar cambios en la implementación del objeto sin necesidad de que sus clientes se hagan conscientes de ello, ya que estos métodos serán los responsables de adaptar los tipos de datos publicados por la interfaz a los tipos de datos de la implementación.

En (West, 2004) se usa la metáfora de la antropomorfización como un medio de gran utilidad a la hora de aliviar esta tendencia. El autor propone que se piense en qué pasaría si el objeto fuese una persona y que, desde esta perspectiva, el desarrollador se ponga en su lugar en dos situaciones muy concretas:

- ¿Accedería directamente a la información particular de una persona en lugar de pedirle la información que necesito?
- Si tuviese la posibilidad de poder mover a mi voluntad las piernas y las manos de una persona y quisiera que bailara ¿manipularía yo sus extremidades o dejaría a él que interpretara el baile que yo le solicite?

2.4.4. Clases y objetos

La primera de las dificultades encontradas es la necesidad de diferenciar entre lo que es una clase y lo que es un objeto (Börstler, 2005). Esto generalmente se debe a que los ejemplos usados emplean una única instancia

para cada clase. De esta forma el alumno hace la incorrecta suposición de que ambos son lo mismo.

Obviando que algunos lenguajes no hacen distinción entre ambos, es necesario que el alumno sea consciente de la diferencia entre la clase y las instancias de la clase. Para ello, la mayoría de los autores defienden el uso de ejemplos lo suficientemente complejos como para que puedan convivir distintas instancias de la misma clase. Por ejemplo, Nygaard usa el ejemplo de un restaurante para tener un primer contacto con la orientación a objetos. Por otro lado, el Advanced Placement Central College Board, una organización formada por un gran número de escuelas y universidades americanas que, entre otros, da soporte a la enseñanza de informática y programación, propone el uso del ya mencionado simulador de biología marina (MBSCS) para los cursos de programación orientada a objetos.

Algunos autores también destacan la importancia de la creación de clases y objetos de suficiente complejidad para que no sean confundidos con variables de programación procedimental (Holland et al., 1997). Es necesario que los objetos tengan más de un atributo para la representación de ese estado y que los atributos sean de distintos tipos, para que no sean interpretados como contenedores de variables del mismo tipo.

2.4.5. Herencia y polimorfismo

Uno de los principales problemas relacionados con la herencia se encuentra en la dificultad de los alumnos para distinguir entre la herencia de implementación y de subtipo. Es muy común que los alumnos abusen de la herencia para reutilizar código sin tener en cuenta si posteriormente va a ser necesario garantizar el Principio de Sustitución de Liskov. Esto se debe en gran parte a que la herencia ha sido sobreusada, sobre todo en los principios de la orientación a objetos. Por ejemplo, en (Meyer, 1997) existen varios ejemplos en los que se ven este tipo de abusos. Se define una clase `FiguraCompuesta` que hereda de una clase `Lista<Figuras>`, ya que una figura compuesta se implementa como una lista de figuras. En esta misma referencia se hace mención de una clase `Punto` y que hereda de la clase `Aritmetica` ya que `Punto` hace uso de operaciones como `sqrt` (raíz cuadrada), que se encuentra implementada en su superclase.

Ejemplos como los anteriores han de ser tenidos muy en cuenta a la hora de explicar la herencia. Si un alumno se encuentra con este tipo de ejemplos pensará que la única forma de tener relaciones entre clases es mediante herencia. Si abusa de esta relación se encontrará con arquitecturas muy rígidas, en las que pequeñas modificaciones afectan a un gran número de clases. En cambio, hay que inculcar al alumno que un buen uso de la herencia permite la creación de arquitecturas flexibles y altamente reutilizables, en las que se hace un uso extensivo del polimorfismo. Esto obliga a buscar ejemplos de una mayor complejidad en la que se haga patente la flexibilidad del diseño

creado.

2.4.6. Paso de mensajes y métodos

Un requisito importante para comprender la orientación a objetos es tener unos conocimientos sólidos sobre la ejecución de un sistema orientado a objetos y, por tanto, de las interacciones que se producen entre los objetos que conforman el sistema. Algunos autores han destacado las dificultades que tienen los alumnos para comprender las colaboraciones que se producen entre los objetos durante la ejecución para la resolución de un determinado problema (Schulte y Bennedsen, 2006).

La ejecución de un sistema tiene una naturaleza meramente dinámica, por lo que suele ser más complicada de explicar y de comprender usando imágenes estáticas. La comprensión de las colaboraciones entre objetos se puede mejorar mediante visualizaciones dinámicas o animaciones y mediante simulaciones (Esteves y Mendes, 2004). De acuerdo al comportamiento de un sistema, estos medios son muy útiles para solventar las discrepancias entre cómo cree un alumno que funciona un sistema y cómo funciona en realidad.

Algunos autores hacen hincapié en el hecho de que los alumnos piensan en los objetos como registros (en programación procedimental) que tan solo contienen información. Es necesario que el alumno se haga consciente de que un método no se basa exclusivamente en operaciones de asignación (Holland et al., 1997) sino que el comportamiento de un objeto puede variar conforme a un estado y un objeto suele completar la responsabilidad fijada por un método necesitando la ayuda de otros objetos.

Hay que tener siempre presente que una clase define unos datos y un comportamiento. Inicialmente se tiende a que el único comportamiento sea el que aportan los métodos accedentes y mutadores, que no aportan comportamiento real al objeto. Hay que promover el uso de ejemplos con métodos de mayor complejidad que los anteriores. Generalmente, los métodos complejos se completan delegando determinadas partes en objetos responsables de pequeñas tareas, y coordinando todas estas delegaciones.

2.4.7. Diseño orientado a objetos

El diseño orientado a objetos es una de las tareas más complicadas para los alumnos (Schulte y Bennedsen, 2006). Los primeros diseños suelen ser pobres y necesitan grandes mejoras pero, con la experiencia, el alumno va adquiriendo poco a poco las destrezas necesarias para ser crítico con su propio trabajo y tener capacidad de decisión ante distintas alternativas de diseño.

El principal problema con el que se encuentra un alumno al realizar un diseño orientado a objetos es la descomposición del problema en clases. Esta tarea no es nada fácil y sólo se va mejorando a medida que se va adquiriendo experiencia. Además, no hay una única forma de descomponer un problema,

por lo que el alumno deberá aprender a evaluar un diseño para poder elegir la solución más apropiada.

Las clases y objetos tratados de manera aislada no sirven por sí mismos. Es muy importante que el alumno comprenda que gran parte de la importancia de la filosofía de la orientación a objetos se basa en la interacción entre los objetos y en la delegación de responsabilidades de unos sobre otros. Un error muy común entre los principiantes es el uso de grandes clases que concentran todo el conocimiento de una parte del sistema. Estas clases suelen contener un gran número de métodos, lo que indica que no se ha hecho un refinamiento correcto del sistema. En este caso, hay que señalar la necesidad de refinar el diseño de estas clases, dividiéndolas en otras que, en conjunto, realicen el mismo comportamiento que las primeras.

A esto hay que añadir que los alumnos han de aprender a adecuar la reutilización de diseño y objetos ya realizados con anterioridad. Un objeto no es bueno o malo por sí mismo y no puede, por tanto, ser reutilizado siempre de igual forma. Un objeto puede ser óptimo para un diseño mientras que resulta pésimo para otro. Esto se debe a que durante el diseño es necesario poner a cada objeto en su contexto, en el que va a interaccionar con otros objetos.

Además, no todos los objetos son entidades tangibles extraídas del dominio. Algunos son artefactos propios de diseño que proporcionan un comportamiento de más alto nivel. Inicialmente parece que estos objetos van en contra de la filosofía de la orientación a objetos, ya que no aparecen en el modelo real. Pero los alumnos han de ir aprendiendo con la experiencia que este tipo de objetos son imprescindibles para la consecución de sistemas bien diseñados.

Estas entidades ficticias generalmente surgen para cumplir algunos de los principios que se describieron en la Sección 2.3.2, principalmente para satisfacer el principio de inversión de independencia. Los alumnos han de ir adquiriendo la costumbre de fomentar el diseño basado en interfaces en lugar del basado en implementación, ya que estos diseños son mucho más flexibles.

Relacionado con el abuso de la herencia explicado anteriormente está el problema de tomar decisiones de elección de alternativas de diseño. Una muy común es decidir entre relacionar dos clases por herencia o por composición. Como se explicó anteriormente, la tendencia es la de sobreutilizar la herencia. Pero es necesario que los alumnos aprendan a valorar y, en los casos en los que haya alternativa, favorecer la composición frente a la herencia (Jacobson et al., 1992; Gamma et al., 1995). La relación de herencia es mucho más fuerte que la de composición, ya que acopla más fuertemente dos clases. Si hay posibilidad de elección, hay que favorecer a la relación más débil.

Frente a la herencia, la composición no rompe la encapsulación de las clases ya que los objetos contenidos se usan como caja negra. Además, esta relación posibilita modificar el comportamiento de un objeto en tiempo de

ejecución ya que es posible definir la composición durante la ejecución del sistema. Sin embargo, no todo son ventajas ya que los sistemas basados en composición tienden a tener muchos más objetos y el diseño de las interfaces de dichos objetos ha de ser mucho más cuidadoso.

Aunque ambos métodos son importantes de cara a la reutilización, se ha tendido a favorecer el uso de la composición ya que ésta conduce a diseños más reutilizables y sencillos. Pero los dos métodos no son mutuamente excluyentes sino que suelen usarse conjuntamente. Generalmente, los objetos que se usan en la composición suelen definirse mediante una jerarquía de herencia (esto se hace tangible en los patrones de diseño *Estado* o *Estrategia*, entre otros (Gamma et al., 1995)).

Por último, es muy importante que el alumno entienda que hay casos en los que no hay posibilidad de elegir entre herencia y composición. Por ejemplo, si se necesita tener acceso a atributos de la superclase que no son accesibles mediante métodos no hay más posibilidad que hacer uso de la herencia. El alumno no sólo ha de saber tomar decisiones entre distintas alternativas sino, también, saber valorar cuándo hay alternativas.

Es claro que las dificultades relacionadas con el diseño orientado a objetos van siendo solucionadas a medida que el alumno va adquiriendo experiencia. Por tanto, hay que proporcionar al alumno oportunidades para ir adquiriéndola. La primera fuente de la que el alumno puede ganar experiencia es de la observación de buenos diseños. Es necesario que el alumno disponga de buenos ejemplos de suficiente entidad sobre los que pueda observar y valorar las ideas de diseño que en ellos se encuentran. La segunda fuente de experiencia es la propia del alumno. No basta con que el alumno observe las experiencias de otros sino que hay que fomentar que se involucre en tareas de diseño, que observe las consecuencias de sus propias decisiones de diseño y que evalúe las bondades y deficiencias de sus diseños.

2.4.8. Patrones de diseño

En los últimos años, los patrones de diseño se han convertido en parte fundamental del currículum de enseñanza de informática (ACM, 2001; ACM, 2004). Pero la enseñanza de patrones de diseño, sobre todo a alumnos de los cursos iniciales, no es tarea sencilla. El principal problema que el docente ha de salvar es la falta de experiencia de diseño del alumnado. Como se apuntó con anterioridad, los patrones de diseño surgen de la experiencia de los profesionales, de personas que, tras años de trabajo de diseño, se han encontrado con numerosos problemas y que han aprendido a darles solución. Enseñar patrones de diseño no consiste sólo en explicar cómo se implementan. El docente ha de ayudar a que los alumnos sean capaces de identificar y comprender un problema de diseño, que lo evalúen y que decidan cuál es la solución de diseño óptima (Chambers et al., 2000). Para esto, los alumnos han de aprender cuáles son las implicaciones de la aplicación de un patrón

de diseño y han de tener un espíritu crítico que les permita evaluar y decidir qué patrón aplicar, en caso de tener distintas alternativas. La enseñanza de estos aspectos avanzados de diseño orientado a objetos requiere el fomento de las experiencias de diseño de los alumnos. Es necesario que los alumnos se enfrenten a problemas de diseño concretos para adquirir estas destrezas.

Tradicionalmente se han empleado las clases magistrales para la enseñanza de los patrones de diseño. A lo largo de los años se ha considerado que el empleo de los ejemplos descritos en (Gamma et al., 1995) era suficiente, ya que se ha considerado que éstos eran lo suficientemente explicativos como para que el alumno comprendiera el fin del patrón y su uso. Lamentablemente, estos ejemplos están puestos en un contexto que puede resultar demasiado alejado para el alumno (especialmente aquél de cursos iniciales). Por ejemplo, para situar el patrón *Estado*, se ha usado el protocolo TCP, un concepto de redes que no tiene por qué conocer el alumno. Por otro lado, un alumno puede no ser consciente de la necesidad de un algoritmo de ruptura de renglones en un editor de texto, como aparece en la descripción del patrón *Estrategia*. Este tipo de ejemplos hacen que el alumno no comprenda la auténtica utilidad de estos patrones debido a que no es capaz de entender el contexto en el que están siendo empleados o porque dichos contextos no tengan ningún interés para él.

Ante este problema, algunos de los creadores de los patrones de diseño software desarrollaron un framework que sirviera de apoyo a la enseñanza de patrones de diseño: JHotDraw (JHotDraw; Kaiser, 2001). Este framework, descrito más en profundidad en la Sección 3.5, facilita la construcción de editores gráficos y ha sido creado haciendo un uso intensivo de patrones de diseño. El contexto de los editores gráficos es mucho más familiar para los alumnos (¡quién no ha usado en algún momento herramientas como Paint, Photoshop o Gimp!) y los conceptos y mecanismos usados en estas herramientas son conocidos.

Una alternativa a los frameworks es el uso de aplicaciones mucho más sencillas pero con la suficiente complejidad como para necesitar hacer uso de patrones de diseño. Estas aplicaciones con fines pedagógicos se conocen como casos de estudio, según se vio en la Sección 2.4.2. Existe una gran variedad de casos de estudio que han sido empleados en la enseñanza de patrones de diseño. El gestor de los ascensores de un edificio (Nevison y Wells, 2003), un laberinto (Nevison y Wells, 2004), un videojuego (Gestwicki, 2007), un compositor musical (Hamer, 2004) o un framework muy sencillo para la generación de presentaciones (Alphonce et al., 2007) son algunos ejemplos.

Aunque los casos de estudio parecen una herramienta de gran utilidad para ver de forma más sencilla el uso de los patrones de diseño, es necesario que el alumno comprenda que éstos son una herramienta de mejora de diseños y que añaden una complejidad que, en ocasiones, puede no ser necesaria. Los

patrones de diseño han de aparecer de manera natural. Pero la tendencia es que cuando a los alumnos se les da el "martillo" de los patrones de diseño, cualquier problema de diseño se convierte en el "clavo" sobre el que usar un patrón de diseño. Hay que inculcar la idea de que los patrones se usan para refinar un diseño que lo necesita. Hay que comenzar diseñando sistemas de manera sencilla y es prácticamente seguro que los patrones no aparecerán en el análisis del modelo inicial. A medida que se tenga un conocimiento claro del sistema y se detecten problemas, habrá que valorar si dichos problemas pueden ser resueltos mediante el uso de algún patrón de diseño. Como afirma el propio Gamma, los alumnos han de sentir los defectos de un mal diseño para poder apreciar el valor real del uso de los patrones (Venners, 2005).

2.5. Conclusiones

En este capítulo se ha analizado el objeto de estudio de esta Tesis: la enseñanza de la orientación a objetos. La orientación a objetos obliga a descomponer la solución de un problema en un conjunto de objetos independientes que ofrecen servicios, que guardan su estado y la información necesaria para proveer esos servicios, y que son capaces de interactuar entre sí para completar una responsabilidad. Para ello, el desarrollador ha de conocer claramente un conjunto de conceptos que son elementales en este paradigma. Como se ha destacado a lo largo de este capítulo, la enseñanza de estos conceptos y del diseño orientado a objetos tiene una serie de dificultades de las que el instructor ha de ser consciente a la hora de enseñar esta disciplina.

Es un error pensar que la enseñanza de la orientación a objetos consiste únicamente en enseñar un lenguaje orientado a objetos. En muchas ocasiones, la formación en orientación a objetos se ha centrado más en la enseñanza de un lenguaje de programación que en los conceptos de la orientación a objetos en sí mismos. El proceso de enseñanza de estos conceptos no se puede diseñar guiándose exclusivamente por la sintaxis de un lenguaje ni ha de ligarse a dicho lenguaje. La formación en orientación a objetos ha de centrarse en la comprensión de los conceptos elementales de este paradigma y en el diseño de sistemas de acuerdo a estos conceptos.

No es la intención del autor afirmar que este paradigma ha de ser enseñado sin realizar implementación alguna. A fin de cuentas, el fin último de la orientación a objetos es el desarrollo de sistemas software por lo que es conveniente dedicar tiempo a las tareas de implementación. La enseñanza de la orientación a objetos debe ir acompañada de actividades que favorezcan la experiencia de los alumnos con este paradigma. El lenguaje ha de ser un medio pero no el fin del proceso de enseñanza ya que los conceptos de orientación a objetos son comunes a cualquier lenguaje que soporte este paradigma.

Es recomendable apoyar el proceso de enseñanza en ejemplos que sirvan

al alumno para mejorar su comprensión del paradigma y del diseño. Como se ha mencionado ampliamente a lo largo del capítulo, los ejemplos deben tener una determinada complejidad, incluyendo varias clases y distintos objetos de una misma clase. Hay que evitar el uso de ejemplos demasiado simplificados y aquellos que conduzcan a posibles equívocos del alumno. Los ejemplos han de tener un objetivo pedagógico concreto. Tienen que ser estudiados con detenimiento por el docente antes de ser usados. El diseño de clases que incluya el ejemplo ha de ser bueno, por lo que se debe evitar la creación de ejemplos al vuelo, durante las sesiones de enseñanza (Börstler et al., 2007).

El uso de ejemplos tiene varios cometidos. Por un lado, los alumnos tratan desde el principio con una muestra real de diseño orientado a objetos, eliminando la necesidad de diseñar y programar desde el principio. Esto permite que los alumnos puedan centrarse en los conceptos más que en la sintaxis de un lenguaje. Por otro lado, el uso de un ejemplo complejo hace que el alumno se acostumbre a tratar con diseños ya creados y fomenta la reutilización, una de las principales armas de la orientación a objetos. El alumno, en lugar de programar un sistema desde cero, hace uso de uno ya desarrollado para ampliar la funcionalidad del sistema. Finalmente, el uso de ejemplos proporciona al alumno sus primeras experiencias de diseño. El alumno aprende de las buenas ideas de diseño que en ellos encuentra y analiza y valora distintas alternativas, fomentando su espíritu crítico.

El diseño, así como los conceptos elementales de la orientación a objetos, se basan en abstracciones. Dice el refrán que "una imagen vale más que mil palabras" por lo que no hay que dejar de lado el valor que pueden tener las visualizaciones de cara a la enseñanza de este paradigma, como se estudiará en capítulos posteriores. A lo largo del presente capítulo se ha destacado que la enseñanza de conceptos como la abstracción y el diseño han de apoyarse en medios que los propios desarrolladores de software utilizan, como son el uso de diagramas. Estas visualizaciones favorecen la comunicación del diseño de un sistema entre desarrolladores y sirven para afianzar y clarificar los conocimientos y las suposiciones que el alumno realiza durante su proceso de aprendizaje.

También hay que tener en cuenta que la enseñanza de conceptos de naturaleza dinámica, como los objetos, el paso de mensajes o la invocación de métodos, puede verse favorecida, aún más que el diseño y la abstracción si cabe, por el uso de visualizaciones. Es fundamental que el alumno comprenda el comportamiento de los objetos y sus cambios de estado durante la ejecución de una aplicación. El uso de visualizaciones dinámicas y simulaciones facilita en gran medida la comprensión de este tipo de conceptos.

La naturaleza de la orientación a objetos hace que no sólo las visualizaciones faciliten su aprendizaje. Muchos autores consideran que la orientación a objetos, además, acepta muy bien el uso de metáforas (Andrianoff y Levine, 2002; West, 2004; Ducasse y Wuyts, 2002). Una de las más aceptadas es la

metáfora de la antropomorfización(West, 2004), en la que los objetos son tratados como personas que interaccionan en una comunidad y que ha sido descrita en la Sección 2.4. Aunque ésta no es la única metáfora es una de las más usadas y revisadas en la literatura. El docente puede hacer uso de ella como herramienta para mejorar la comprensión tanto de los conceptos como del diseño orientado a objetos.

De todos modos, el uso de estas metáforas y visualizaciones no es suficiente. Es necesario que el alumno sea partícipe y que experimente. Como se ha hecho hincapié a lo largo del capítulo, la experiencia es clave en el afianzamiento de los conceptos de la orientación a objetos y, sobre todo, en el diseño de sistemas orientados a objetos. Como se afirma en (Astrachan et al., 1998) "los buenos diseños provienen de la experiencia, y la experiencia proviene de los malos diseño". El alumno no puede ser un mero observador durante la enseñanza de la orientación a objetos. Ha de ser parte activa del proceso de enseñanza y tiene que intervenir en tareas de diseño y de análisis de soluciones propuestas por los docentes. Para una enseñanza efectiva de la orientación a objetos es necesario convertir las clases tradicionales, centradas en el docente, en sesiones en las que el alumno pasa de ser un oyente pasivo a ser el motor del proceso de enseñanza.

Capítulo 3

Recursos para la enseñanza de la orientación a objetos

¡Alto! Lo que están viendo en esas pantallas
no es real; es sólo una alucinación
creada por el computador.

Juegos de guerra (1983)

El ser humano tiene una gran capacidad para procesar e interpretar información visual. Desde la prehistoria, los seres humanos hemos usado las imágenes como medio para comunicarnos. El refrán "una imagen vale más que mil palabras" representa perfectamente la fuerza que tiene este medio de comunicación para los humanos. La creación de estas imágenes es especialmente importante en el entendimiento de conceptos abstractos, sobre todo de aquéllos relacionados con el comportamiento y la evolución en el tiempo de conceptos de naturaleza dinámica.

La visualización es un medio ampliamente utilizado en la enseñanza. La concepción tradicional de la visualización supone que el alumno es un elemento pasivo que tan solo observa y que es capaz de extraer de una sucesión de imágenes el conocimiento necesario para comprender un concepto abstracto. Aunque esta concepción es válida en la enseñanza de muchos conceptos, puede no ser suficiente para otros.

La enseñanza de la orientación a objetos tiene por objeto que el alumno comprenda un conjunto de conceptos abstractos y de mecánicas y que sea capaz de ponerlos en práctica durante el proceso de desarrollo de software. Ha existido una gran dedicación en la comunidad docente en la búsqueda de prácticas y medios que ayuden a docentes y discentes en la enseñanza y aprendizaje, respectivamente, de esta disciplina. Esta búsqueda ha generado una gran cantidad de trabajo relacionado con el desarrollo de herramientas con las que facilitar la enseñanza de conceptos relacionados con la orientación

a objetos. La mayoría de estas herramientas abanderan la visualización y la interacción del alumno como medios para mejorar el proceso de aprendizaje.

La gran variedad de herramientas encontradas nos ha conducido a realizar un estudio de las mismas. Este capítulo contiene los resultados obtenidos en dicho estudio. La motivación principal del mismo es la detección de las buenas prácticas y de las debilidades de estas herramientas de acuerdo a las dificultades encontradas en la enseñanza de la orientación a objetos. Por otro lado, y como ya se ha destacado en el capítulo previo, los frameworks y los casos de estudio también representan un recurso altamente valioso para los docentes.

Dada la relevancia de la visualización la Sección 3.1 comienza presentando las visualizaciones como medio de enseñanza de conceptos relacionados con la programación en general. En particular, esta sección se centra en las visualizaciones interactivas y en los distintos mecanismos usados para involucrar al alumno en el proceso de aprendizaje. Así mismo, se describen las principales características de los entornos de visualización interactiva.

La Sección 3.2 propone una clasificación de los distintos recursos de apoyo a la enseñanza de la orientación a objetos encontrados. Las siguientes tres secciones proporcionan una descripción general de estos tipos de recursos, así como de algunos de los ejemplos más representativos de cada uno de estos tipos: entornos de desarrollo pedagógicos (Sección 3.3), micromundos (Sección 3.4) y frameworks y casos de estudio (Sección 3.5).

En la Sección 3.6 se elabora un estudio comparativo de los distintos recursos. Se analiza la forma en la que abordan los principales problemas detectados en la enseñanza de la orientación a objetos, recopilados en el capítulo anterior. Además de centrarse en el tratamiento de los conceptos fundamentales del paradigma, este estudio también refleja el uso que se ha hecho de distintos lenguajes de programación, la utilización de ejemplos y el tratamiento que cada uno de los recursos dan al diseño orientado a objetos y a los patrones de diseño.

Por último, la Sección 3.7 destaca los resultados obtenidos en el estudio de estos recursos desde el punto de vista de su interactividad con el alumno. Se analizan las distintas estrategias empleadas por las herramientas y se clasifican de acuerdo al nivel de participación del alumno que promueven.

3.1. Visualizaciones y enseñanza de la programación

El diccionario de la RAE (Real Academia Española) define el verbo *visualizar*, en su tercera acepción, como "formar en la mente una imagen visual de un concepto abstracto". Pero también la RAE da una última acepción en la que dice que *visualizar* es "hacer visible una imagen en un monitor". Si conjugamos ambas definiciones podremos entender que, en el ámbito que nos concierne, la visualización es el acto de crear representaciones de conceptos

abstractos mediante imágenes proyectadas en un monitor.

Las visualizaciones se han usado en la enseñanza de todo tipo de disciplinas: física, química biología, mecánica... Esto se debe a que la visualización proporciona una metáfora para conceptos complicados, especialmente para aquéllos relacionados con comportamientos dinámicos y cambios de estado.

La enseñanza de la Informática es una de las áreas en la que hay mucho trabajo de investigación relacionado con el estudio y desarrollo de visualizaciones. Las visualizaciones han sido empleadas prácticamente en todos sus ámbitos pero nuestro interés se centra en la visualización de software. Ésta se define como el uso de tipografía, diseño gráfico, animación por ordenador, cinematografía y gráficos interactivos para facilitar la comprensión y el uso efectivo del software (Price et al., 1993). Desde el punto de vista de la enseñanza, las visualizaciones son usadas fundamentalmente con tres objetivos de aprendizaje:

- Comprensión de programas.
- Evaluación de programas ya existentes.
- Desarrollo de nuevos programas.

Es difícil evaluar el impacto de las visualizaciones en el proceso de aprendizaje. En la literatura no hay resultados empíricos concluyentes que constaten su efectividad. Sin embargo, la creencia general de los docentes es que las visualizaciones son de gran ayuda para enseñar conceptos relacionados con la programación. Un estudio realizado entre 186 docentes pertenecientes a más de 20 países distintos (Naps et al., 2002) destaca que la mayoría de los docentes consideran que las visualizaciones mejoran la capacidad de aprender del alumno. Esto es debido a que una amplia mayoría considera que los alumnos están más participativos, ya que las clases resultan más entretenidas y disfrutan más de su propio proceso de aprendizaje. Además, los docentes creen que los alumnos se sienten más motivados y que mejoran en su aprendizaje. Por último, uno de los principales beneficios de las visualizaciones que los docentes destacan es que sirven como medio para generar discusiones relativas a conceptos de programación o algoritmia.

El mayor problema de la visualización es que tradicionalmente ha tenido una naturaleza pasiva: el alumno simplemente observa lo que ocurre en la pantalla de su ordenador o, como mucho, controla la ejecución de la animación. En el ámbito de la programación, el valor pedagógico de la visualización se ve muy reducido si no se involucra al alumno (Naps et al., 2002). Por tanto ha sido necesaria una evolución de las visualizaciones, de modo que éstas fomenten la participación del alumno, modificando el rol pasivo que tradicionalmente ha interpretado.

Esta evolución dio lugar a lo que se conocen como visualizaciones interactivas y se han empleado en el ámbito de la Informática desde la década de los

80 (Naps et al., 2002). Este tipo de visualizaciones promueven la participación del alumno con distintas propuestas como la modificación de los datos de entrada que configuran la visualización, la creación de sus propias visualizaciones o la predicción del comportamiento mediante preguntas estratégicas. De acuerdo a las estrategias empleadas se ha elaborado una categorización de las distintas formas de participación de los alumnos en las visualizaciones (Naps et al., 2002):

1. **Visionado**. Es la más pasiva de todas las alternativas y es la forma de participación que mayor número de variantes tiene. Entre estas variantes se encuentran el control de la ejecución de la visualización, la capacidad de poder cambiar de vista (si la visualización dispone de múltiples vistas) o la inclusión de descripciones textuales y sonoras que acompañen a la visualización.

2. **Respuesta**. Esta forma de participación propone involucrar al alumno mediante la realización de preguntas relacionadas con la visualización. Estas preguntas pueden estar relacionadas con múltiples ámbitos de la programación, como depuración, código o eficiencia. Estas preguntas pueden formar parte de un cuestionario independiente de la herramienta de visualización o pueden ser integradas dentro del entorno de visualización, de modo que una respuesta correcta permita que el alumno pueda ver nuevas visualizaciones.

3. **Cambio**. Este forma de participación fomenta que los alumnos alteren la visualización mediante la modificación de los datos de entrada que varían el comportamiento de la visualización.

4. **Construcción**. Este forma de participación promueve la construcción de nuevas visualizaciones. Dependiendo del entorno de visualización empleado existen distintos tipos de construcción. En algunos, la construcción consiste en la codificación de un programa. El entorno será el encargado de analizarlo y de construir automáticamente la visualización. En el extremo opuesto estaría la creación artesanal de visualizaciones, de modo que el entorno proporcione las primitivas necesarias para que el alumno pueda generar nuevas animaciones.

5. **Presentación**. Esta forma de participación propone que el alumno exponga una visualización a una audiencia y discuta sobre los contenidos de la misma. Esta visualización puede haber sido creada por el alumno o puede ser un recurso externo.

La efectividad de la mayoría de los recursos basados en visualización no ha sido medida empíricamente. Sin embargo, la experiencia en el desarrollo de este tipo de herramientas ha conducido a la elaboración de una lista de

buenas prácticas que han proporcionado una serie de resultados muy satisfactorios y que recogen las distintas formas de participación presentadas (Naps et al., 2002). Estas buenas prácticas han sido resumidas en las siguientes recomendaciones:

- Incluir recursos para la interpretación de las representaciones gráficas. Es fundamental que el alumno entienda el significado de los elementos visuales que aparecen en la escena. Si la relación entre el concepto que se pretende enseñar y su representación no es clara, entonces es necesario incluir información sobre el significado de las metáforas visuales empleadas.

- Facilitar visualizaciones adaptables a los alumnos. Generalmente, un alumno novato necesita visualizaciones con pocos detalles, poca interactividad y metáforas conocidas. Por el contrario, los más avanzados prefieren tener un mayor control sobre la visualización y sobre la información que se les presenta, desean personalizarla y añadir sus propios datos y no les importa tener distintas vistas de la misma información. Por tanto se recomienda que estas herramientas de visualización proporcionen los medios para la personalización de la misma.

- Incorporar múltiples vistas. Un programa puede ser observado desde distintos puntos de vista: algoritmo, pseudo-código, código, comportamiento en ejecución... La posibilidad de tener múltiples vistas simultáneas facilita la comprensión del alumno. Por ejemplo, si un alumno observa el comportamiento de un algoritmo y, a medida que está siendo ejecutado, las líneas de código ejecutadas aparecen resaltadas, el alumno será capaz de relacionar las acciones del algoritmo con su código fuente.

- Incluir información sobre rendimiento. Esta información es especialmente útil para las visualizaciones de algoritmos ya que permiten que el alumno se haga consciente del coste de los mismos en tiempo y en memoria. Esta información puede presentarse de manera explícita o comparativamente. Por ejemplo, se pueden ejecutar simultáneamente dos algoritmos de ordenación, de modo que el alumno infiera la eficiencia en tiempo de los mismos viendo cuál termina antes.

- Incluir un historial de ejecución de la visualización. Se debería permitir que el alumno fuese capaz de volver atrás a puntos de especial interés, sobre todo si puede realizar cambios que le permitan modificar el comportamiento de la visualización.

- Permitir un control de ejecución flexible. Muchas herramientas tratan las visualizaciones de igual modo que se manipula el visionado de una cinta de vídeo. El alumno es capaz de ir hacia adelante y hacia atrás,

al principio y al final de la visualización, pararla, congelarla o moverse paso a paso por la misma.

- Permitir que el alumno pueda crear sus propias visualizaciones. Por supuesto, este tipo de entornos son los que más involucran al alumno, por lo que son los que resultan más interesantes de cara a la efectividad pedagógica de la visualización.
- Permitir que los alumnos puedan introducir su propio conjunto de datos de entrada, ya que da la posibilidad de reafirmar los conocimientos del alumno realizando múltiples pruebas.
- Proporcionar preguntas dinámicamente. Es de gran interés que el entorno sea capaz de generar preguntas al alumno en determinados puntos de la visualización de acuerdo en el contexto en el que se encuentra. De este modo el alumno es capaz de evaluar sus propios conocimientos. Estas preguntas pueden ser, en ocasiones, críticas, de modo que la visualización no se reanude hasta que no sean respondidas correctamente.
- Proporcionar retroalimentación. En los entornos en los que el alumno participe activamente durante la simulación, es recomendable que se proporcione información sobre lo que se está realizando y sobre la importancia de dichas actividades.
- Completar las animaciones con explicaciones. Estas explicaciones pueden estar incluidas en algún tipo de documento de apoyo para el alumno durante la ejecución de la simulación. En realidad, lo verdaderamente interesante es que dichas explicaciones se encuentren coordinadas con la visualización y que aparezcan en paralelo con la misma, ya sea en modo texto o mediante audio.

En las secciones posteriores se hablará más en detalle de cada uno de los tipos en los que se clasifican los recursos de apoyo a la docencia. Así mismo, se describirán ejemplos de los principales recursos encontrados.

3.2. Clasificación de los recursos de apoyo a la enseñanza de la orientación a objetos

Al igual que "cada maestrillo tiene su librillo", cada instructor relacionado con la enseñanza de la programación tiene sus técnicas y recursos para enseñar su disciplina de la manera que él considera mejor para sus alumnos. En la comunidad investigadora hay un gran número de artículos que hablan sobre distintas herramientas, entornos y artefactos desarrollados por los docentes para tratar de mejorar la forma en la que hacer llegar al alumno los conocimientos que enseña.

El tipo de recursos de apoyo usados para la enseñanza de la orientación a objetos es de lo más variado. En (Bruce, 2004) se afirma que, en el ámbito de la enseñanza de la orientación a objetos (usando Java), hay tres tipos fundamentales de recursos de apoyo a docentes y alumnos: los entornos de desarrollo pedagógicos, los micromundos, y herramientas Java destinadas a la pedagogía (*toolkits*). Esta clasificación ha servido como base para el estudio sobre estos recursos realizado en esta memoria de Tesis, que serán descritos en las próximas secciones y clasificados de acuerdo a los siguientes tipos:

1. **Herramientas de visualización interactiva.** Dentro de este tipo se engloban las siguientes:

   a) **Entornos integrados de desarrollo pedagógicos.** Son entornos para el desarrollo de programas orientados a objetos. Incluyen una importante componente de visualización del comportamiento y del estado de los objetos y han sido diseñados para simplificar las labores de programación a alumnos novatos.

   b) **Micromundos.** Son aplicaciones que sirven como banco de pruebas del conocimiento de los alumnos. Los micromundos son entornos habitados por entidades cuyo comportamiento es programado por el alumno de acuerdo al repertorio de operaciones que cada entidad sabe realizar. Estos entornos tienen siempre una representación visual con la que se muestran los cambios de estado y el comportamiento de las entidades en respuesta a las acciones programadas por el alumno.

2. **Recursos para el desarrollo de aplicaciones visuales.** Dentro de este tipo destacan los **frameworks y casos de estudio**, empleados como herramientas de apoyo para alumnos y profesores para el desarrollo de aplicaciones. Estos recursos también son empleados como ejemplo de aplicaciones bien diseñadas de acuerdo al paradigma orientado a objetos.

3.3. Entornos integrados de desarrollo pedagógicos

Un entorno integrado de desarrollo o IDE (del inglés, *Integrated Development System*) es un entorno de programación que incluye varias herramientas relacionadas con el desarrollo de programas. Generalmente, un IDE está especializado para el desarrollo con un lenguaje concreto aunque hay algunos como Eclipse que soportan distintos lenguajes de programación.

En Kölling (1999b) se detallan los principales requisitos que un IDE dedicado a la enseñanza de programación a objetos ha de cumplir:

Facilidad de uso La curva de aprendizaje de estos entornos ha de ser reducida para que los alumnos puedan rápidamente centrarse en los conceptos de programación a objetos en lugar de perder el tiempo en aprender el funcionamiento del IDE. Generalmente, esto se facilita mediante la ocultación de detalles innecesarios que puedan ser útiles en otros entornos profesionales. A fin de cuentas, un alumno tan solo va a escribir el código, compilarlo y testearlo mediante la ejecución de la aplicación creada.

Herramientas integradas Es recomendable que todas las herramientas de las que el alumno pueda hacer uso se encuentren integradas en el mismo IDE. Esta característica está íntimamente ligada a la anterior de modo que el IDE tendrá una interfaz unificada y reducida para que sea fácil de aprender e incremente la productividad de desarrollo de los alumnos.

Soporte a la orientación a objetos La programación imperativa se basa en el hecho de que existe un programa principal en torno al cual se articula la aplicación. Por esto, la mayoría de los IDEs dedicados a este paradigma se centran en dar soporte a la creación de una aplicación con un punto de entrada único y que será compilada y ejecutada para comprobar el desarrollo de la misma. En cambio, un IDE para la enseñanza de la programación a objetos tiene que estar centrado en instancias, es decir, ha de permitir la creación de objetos de cualquier clase, manipularlos y ejecutar métodos de su interfaz sin necesidad de construir la aplicación completa. Estos entornos han de tener a la clase y el objeto como la abstracción principal.

Soporte para la reutilización de código Como ya se ha destacado en anteriores capítulos, la reutilización es una de las motivaciones principales de la programación orientada a objetos. Por tanto, estos IDEs han de dar soporte a la misma mediante la presencia de exploradores de clases. Éstos permitirán a los alumnos tanto la búsqueda de librerías como de las nuevas clases que el alumno implemente para el desarrollo de nuevas aplicaciones.

Soporte a la enseñanza Esta característica implica que estos entornos han de estar adaptados para que el alumno incremente la comprensión de los conceptos referentes a la orientación a objetos. Se sugiere que proporcionen la capacidad de ser interactivos con las clases y objetos desarrollados. Además, deberán facilitar el control de la ejecución.

Visualización Estos entornos han de proporcionar información visual tanto de la estructura estática como dinámica de las aplicaciones que en ella se desarrollan. Es recomendable que proporcionen diagramas en los que poder observar la estructura de las clases, así como las relaciones entre las mismas, y las relaciones que se producen entre los objetos

en tiempo de ejecución. También han de permitir que los alumnos puedan inspeccionar el estado de los objetos en ejecución. Por otro lado, también se recomienda que haya tanto una representación gráfica como textual de las clases desarrolladas.

Soporte para grupos Ya que tradicionalmente las tareas de programación no se realizan de manera individual, los IDEs de enseñanza deberían dar soporte para controlar el proceso de integración, en el caso que distintos alumnos trabajen en diferentes partes del sistema simultáneamente.

Disponibilidad Este requisito se refiere principalmente al coste asociado al IDE. Aunque los destinados al desarrollo profesional tienen un alto precio (por ejemplo, una licencia completa de JBuilder 2007 cuesta 1999$), sería recomendable que los entornos destinados a la enseñanza tengan el coste más bajo posible (a poder ser, gratuitos).

Uno de los entornos de enseñanza de programación orientada a objetos más utilizado hoy día en la comunidad académica es BlueJ (Barnes y Kölling, 2005; Kölling y Rosenberg, 2001, 2003). Este entorno nació como una evolución del entorno Blue (Kölling y Rosenberg, 1996a), un entorno desarrollado para cubrir las necesidades descritas anteriormente. Este entorno estaba dedicado al lenguaje Blue (Kölling y Rosenberg, 1996b), un lenguaje orientado a objetos creado ante la problemática de usar lenguajes como C++ para enseñar este paradigma.

El precursor de Blue, BlueJ, es un entorno para la programación en Java y que destaca por tener una interfaz muy simple, como se puede observar en la Figura 3.1. En ella se puede ver que hay una clara distinción entre clases y objetos. Las primeras (en la parte superior) son representadas mediante diagramas UML de clase. Los segundos (en la parte inferior) aparecen en lo que se conoce como banco de objetos y pueden ser creados en cualquier momento directamente desde una clase. El alumno puede interactuar con ellos mediante la ejecución de cualquiera de sus métodos.

Las clases en BlueJ son creadas haciendo uso de un sencillo editor de texto incluido en el IDE. También dispone de un depurador que permite a los alumnos revisar la ejecución de sus aplicaciones.

Otro entorno de características similares a BlueJ pero que da un especial énfasis a la visualización de las estructuras de datos durante la ejecución es jGRASP (Hendrix et al., 2004; Cross II et al., 2007). Al igual que BlueJ, la estructura de clases viene representada mediante diagramas UML y es posible crear instancias de clases y comprobar su comportamiento sin necesidad de crear una aplicación completa. Este entorno dispone además de varios exploradores de clases y objetos y de múltiples vistas de la misma información. El editor de código permite ocultar ciertas secciones e incluye información visual sobre las estructuras de control.

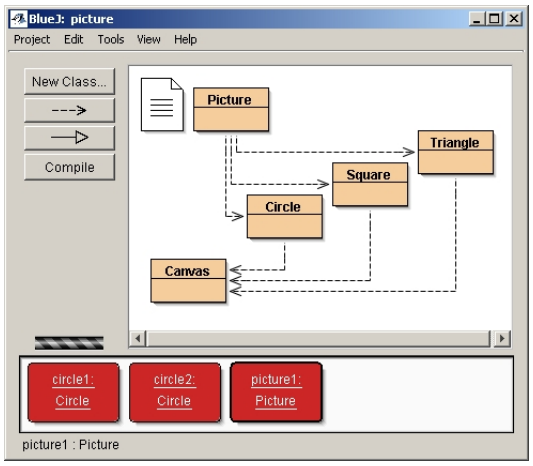


Figura 3.1: Vista principal del entorno BlueJ.

Como se ha mencionado anteriormente, una de las principales cualidades de jGRASP es la capacidad de visualizar, de manera amigable, estructuras de datos complejas. El entorno dispone de un módulo identificador de estructuras de datos que analiza las clases y los enlaces existentes entre las mismas para generar automáticamente una visualización específica para esa estructura de datos a partir de unos modelos de visualización. Como ejemplo, en la Figura 3.2 se puede observar el estado de una lista doblemente enlazada durante la ejecución de un programa de prueba.

Otros entornos prestan mucha más atención al paso de mensajes durante la ejecución en lugar de al estado de los objetos. Éste es el caso de la familia de editores Eliot (Ben-Ari et al., 2002). Su última versión, Jeliot 3[1] es una herramienta de visualización de programas diseñada para enseñar programación procedimental y orientada a objetos usando el lenguaje Java. Estas herramientas se caracterizan por definir lo que se conoce como la metáfora del teatro. En esta metáfora, el código fuente interpreta el papel del guión de una obra de teatro y cada una de las variables, objetos, procedimientos y métodos que en él aparecen son considerados como personajes o roles. En la visualización será necesario definir cuáles son los actores, es decir, los elementos visuales que interpretarán cada uno de los personajes. Por último, existe un director que elige los actores que van a ser empleados y que los coordina para generar la visualización completa.

La evolución de esta herramienta comienza en 1997 con Eliot, una herramienta de visualización de algoritmos para la creación de animaciones de manera semiautomática. Está implementada en C++ y muestra estructuras de datos y tipos en C. El código no es visualizado durante la ejecución y

[1]Disponible en `http://cs.joensuu.fi/jeliot/` (último acceso: 15-02-2008)

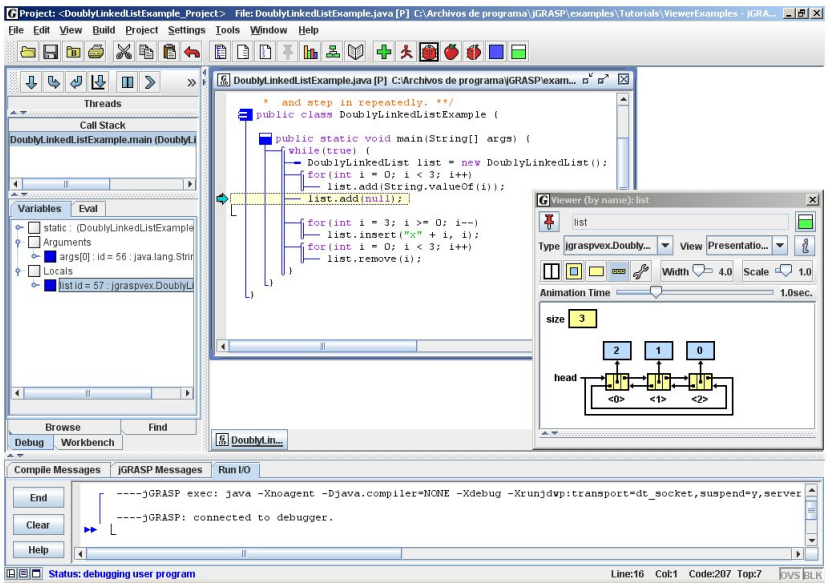


Figura 3.2: Aspecto del entorno jGRASP. A la derecha se puede ver la visualización del estado de una lista doblemente enlazada.

solamente funciona sobre entornos X Windows.

Poco tiempo después surge Jeliot (Java-Eliot), una plataforma cliente-servidor para la visualización de programas Java a través de web que soluciona el problema de la plataforma de Eliot. Es posible visualizar el flujo de datos pero no el de control. Esta versión dispone de un editor de código independiente de la visualización. En Jeliot se pueden visualizar la mayoría de los tipos de Java aunque se pueden añadir nuevos, codificando la correspondiente visualización de sus datos y sus operaciones. La evaluación de Jeliot sacó a la luz diversos problemas relacionados con la granularidad de la visualización y la necesidad de mostrar información del flujo de control.

Teniendo en cuenta los problemas hallados, en 2003 se crea Jeliot 2000, una aplicación *standalone* en Java con una interfaz de control de la animación muy sencillo —controles similares a los que tiene un reproductor de vídeo— y diseñado específicamente para alumnos novatos sin conocimientos en programación. Jeliot 2000 permite la visualización tanto de flujo de datos como de control. Incluye visualización de código a nivel de instrucciones y expresiones pero sólo es capaz de visualizar un pequeño subconjunto de Java y no soporta el paradigma de programación orientada a objetos.

En 2004 nace Jeliot 3 (Moreno et al., 2004; Myller, 2004), que extiende a su antecesor incluyendo la visualización del paradigma de la orientación a objetos, además de aumentar el subconjunto del lenguaje Java que es capaz de visualizar. Como puede verse en la Figura 3.3, este entorno carece de representación estática de clases, ya que se centra en la visualización del

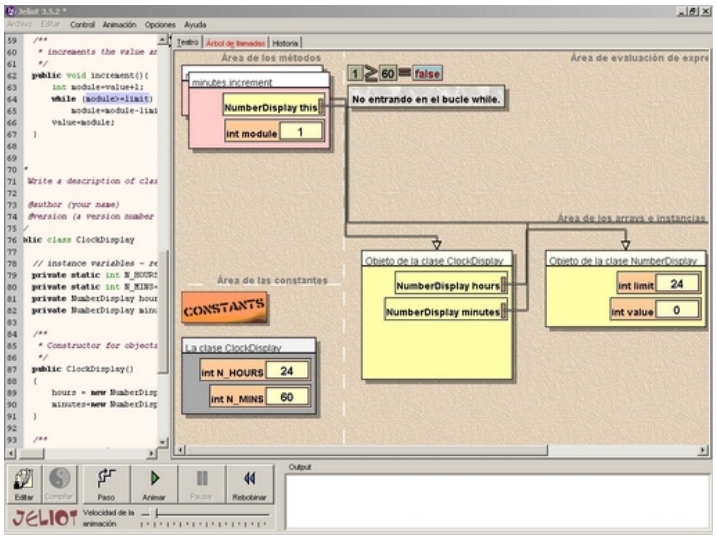


Figura 3.3: Aspecto de Jeliot3.

comportamiento de la aplicación. Durante la ejecución, en la zona de teatro, que aparece a la derecha, se irán visualizando animaciones que representarán la ejecución de métodos y de los cambios de estado que irán sufriendo los objetos que han sido creados, así como animaciones para la evaluación de las expresiones que se produzcan. Simultáneamente, en la parte izquierda aparece el código del programa que está siendo ejecutado. En él aparecen resaltadas la instrucción que actualmente está siendo simulada en el área del teatro.

Los anteriores entornos son un firme reflejo de la importancia que Java ha adquirido como lenguaje para la enseñanza de la programación orientada a objetos. En cambio, otros entornos dejan a un lado las distracciones que puede provocar la sintaxis de un lenguaje y usan otras formas de creación de programas. Como ejemplo tenemos el entorno JPie (Goldman, 2004b,c), un IDE de programación por manipulación directa que hace uso del concepto de cápsulas y regiones semánticas para la creación de programas (Goldman, 2004a). Una cápsula puede representar tipos, declaraciones y acceso a variables, atributos, métodos y constructores, llamadas a métodos y a constructores, expresiones y constantes. Estas cápsulas son manipuladas por el usuario de manera muy sencilla, arrastrando y soltando sobre una región semántica. Cada región semántica define el significado concreto de la cápsula que se ha arrastrado sobre ellas. En la Figura 3.4 se puede ver el aspecto visual de las cápsulas (a la izquierda) y regiones semánticas (a la derecha).

Este entorno promueve lo que se conoce como "programación en vivo" (del inglés, *live development*). Este tipo de programación se define como la capacidad de editar programas mientras se están ejecutando. Si durante la

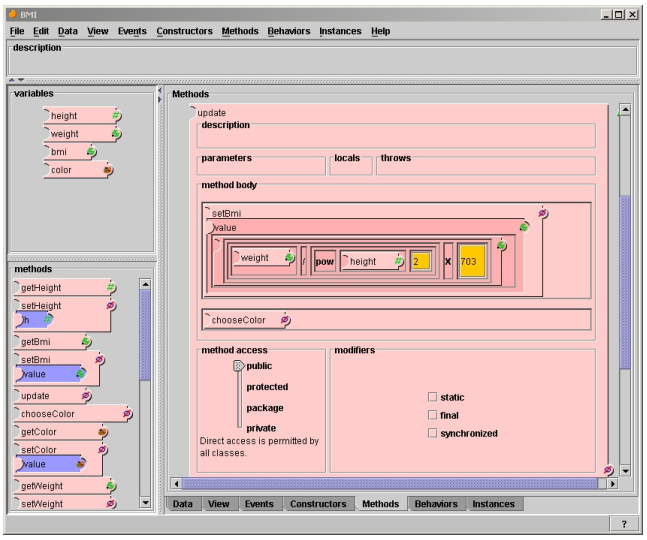


Figura 3.4: Aspecto de JPie.

ejecución se alcanza un punto en el que se encuentra un error, ésta se detiene hasta que la instrucción que contiene el error sea ejecutable. Posteriormente se puede continuar con la ejecución desde el punto en que se dejó (Birnbaum y Goldman, 2005). Las clases desarrolladas de esta forma se conocen como clases dinámicas y permiten modificar la definición de una clase en tiempo de ejecución, de modo que los cambios generados afectan inmediatamente a las instancias u objetos ya creados.

Los entornos vistos hasta el momento dan apoyo a la codificación de los programas. En cambio, existen otros entornos que ponen el énfasis especialmente en el diseño detallado de las clases, dejando inicialmente de lado la implementación. Fujaba (*From UML to Java And Back Again*)[2] es un editor para el desarrollo de sistemas en Java a partir de diagramas UML, como se puede ver en la Figura 3.5. El desarrollador se encarga de especificar la arquitectura del sistema mediante diagramas de clases y el comportamiento de las clases que lo conforman mediante diagramas de colaboración. Fujaba también se usa en ingeniería inversa ya que es capaz de generar los diagramas UML correspondientes a partir del código fuente en Java.

Fujaba no ha sido desarrollada como herramienta de enseñanza. Sin embargo, el proyecto Life3 (*"Lernwerkzeuge für den Informatikunterricht: Einsetzen, Evaluieren und (Weiter)Entwickeln"* —Herramientas de enseñanza para cursos de informática: uso, evaluación y desarrollo) [3] promueve el uso de Fujaba como herramienta pedagógica para enseñar programación orientada a objetos en escuelas de secundaria (16-17 años), teniendo en cuenta

[2]Disponible en `http://www.fujaba.de/` (último acceso: 15-02-2008)
[3]`http://life.upb.de/` (último acceso: 15-02-2008)

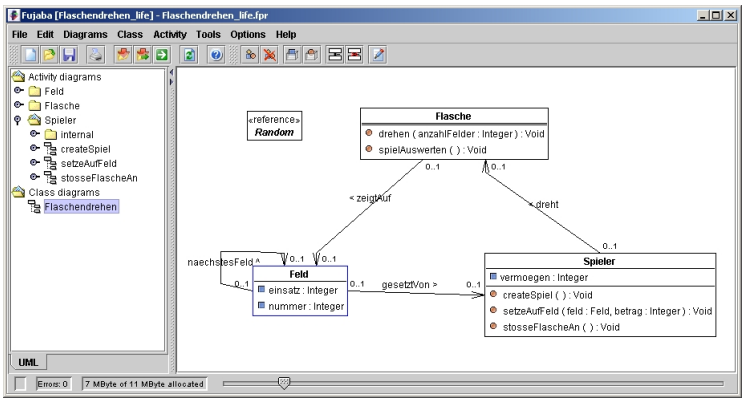


Figura 3.5: Aspecto del editor de clases de Fujaba.

la experiencia de los autores en la enseñanza de esta disciplina en la universidad. Life3 usa métodos de enseñanza activa para que el alumno aprenda tanto diseño como programación orientada a objetos. Propone al alumno un problema a resolver, de modo que éste se centre en hacer un buen diseño de la solución para dicho problema, dejando a un lado inicialmente la implementación. De hecho, los alumnos pueden especificar toda la aplicación usando Fujaba y dejar que ésta genere automáticamente el código en Java. De esta forma se abstrae al alumno de los conceptos propios del lenguaje de programación y se le centra en la estructura de clases y la colaboración entre objetos (Niere y Schulte, 2002; Schulte et al., 2003).

3.4. Micromundos

Otra forma de combinar la visualización con la participación activa del alumno es proporcionarle entornos en los que el alumno construya y programe el comportamiento de entidades en base a un repertorio de acciones y en los que pueda experimentar y descubrir a través de la observación del comportamiento programado. Este tipo de entornos es lo que se conoce como *micromundos*. Como se verá más adelante, algunos de los micromundos utilizados en la enseñanza de la orientación a objetos aumentan las capacidades de manipulación del alumno. Unos facilitan la inclusión de nuevas acciones y la modificación del comportamiento básico de las entidades predefinidas. Otros permiten que el alumno construya nuevas entidades, ya sea especializando las existentes, ya sea creándolas desde cero. Los más avanzados soportan incluso la creación de nuevos micromundos.

Estos entornos nacen siguiendo las teorías constructivistas de (Piaget, 1955), en las que se considera que el aprendizaje consiste en la construcción de las ideas internas del alumno en el mundo real. Cualquier cosa que el alumno percibe se contrasta con los conocimientos previos de los que dispone y,

si encaja con sus ideas internas, entonces pasará a formar parte de su conocimiento. Las teorías constructivistas, por tanto, se centran en proporcionar al alumno oportunidades que le permitan construir conocimiento.

Un micromundo proporciona un modelo de aprendizaje activo: el alumno juega con el entorno, experimenta, explora alternativas, prueba sus propias hipótesis y descubre. En estos entornos el alumno aprende construyendo y refinando su propio conocimiento. Seymour Papert proporcionó en la década de los 80 una de las primeras definiciones formales de lo que era un micromundo (Papert, 1980):

> *" ...a subset of reality or a constructed reality whose structure matches that of a given cognitive mechanism so as to provide an environment where the latter can operate effectively. The concept leads to the project of inventing microworlds so structured as to allow a human learner to exercise particular powerful ideas or intellectual skills."*

Según (Edwards, 1998) las características de un micromundo se pueden ver desde dos perspectivas distintas: desde el punto de vista estructural y desde el punto de vista funcional. De acuerdo a su estructura, un micromundo se compone de una serie de entidades "computacionales" que modelan ciertos aspectos del dominio que pretende enseñar. Un micromundo suele ir acompañado de una serie de actividades con un objetivo bien definido que el alumno ha de realizar usando las entidades y las operaciones proporcionadas. Para completar estas actividades, las entidades y las operaciones pueden combinarse para crear nuevas entidades u operaciones más complejas.

Desde el punto de vista funcional, se espera que el alumno haga uso del micromundo manipulando los objetos y ejecutando operaciones del entorno. Además, el alumno ha de interpretar la retroalimentación que el micromundo le proporciona en respuesta a las manipulaciones y operaciones realizadas, de modo que vaya refinando su conocimiento del dominio. Por último, se espera que complete las actividades que acompañan al micromundo utilizando las entidades y operaciones proporcionadas por el mismo.

Desde el punto de vista estructural, es importante destacar que los micromundos ofrecen una representación metafórica de los conceptos a aprender. Cada micromundo tiene sus propias metáforas y representaciones, dependiendo del dominio que se pretende enseñar y de las distintas representaciones que se pueden dar a los conceptos del mismo dominio. Sea cual sea la metáfora utilizada, es muy importante que sea definida de modo que el alumno sea capaz de interpretarla fácilmente. El principal fracaso en el diseño de un micromundo es que el diseñador cree una metáfora para unos determinados conceptos que luego el usuario no sea capaz interpretar, de modo que el usuario pierde la consciencia de qué es lo que el diseñador quiere enseñar con la metáfora usada.

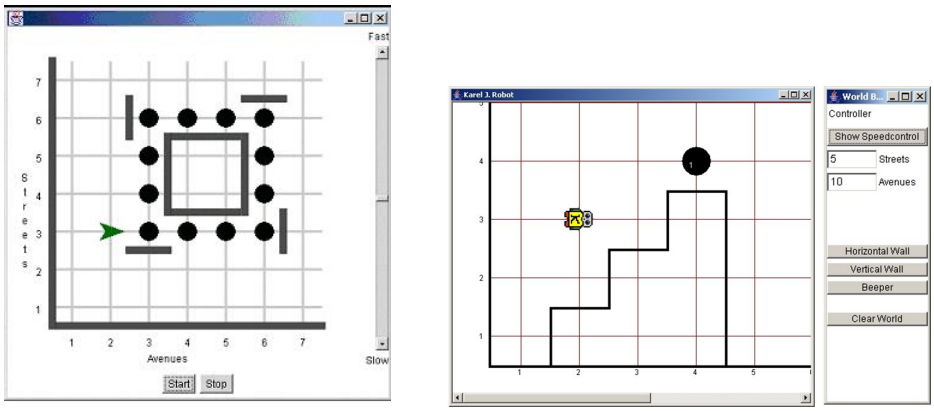
Figura 3.6: La tortuga mecánica del lenguaje Logo (Papert, 1980).

Tal vez los primeros ejemplos conocidos de micromundos fueron los creados usando el lenguaje Logo (Papert, 1980). Mediante este lenguaje el alumno era capaz de dirigir desde un computador a una tortuga mecánica (como la que aparece en la Figura 3.6) que disponía de un rotulador con el que dibujaba en el suelo figuras geométricas (lo que ha llegado hasta nuestros días con el nombre de *gráficos de tortuga*).

Este entorno fue usado para enseñar los conceptos fundamentales de programación a niños. El alumno programaba el comportamiento de la tortuga mediante instrucciones muy sencillas (mover, rotar, levantar y bajar el rotulador...) y comparaba sus expectativas de comportamiento de la tortuga con el comportamiento real de la misma, que se obtenía interpretando el dibujo que la misma realizaba sobre el suelo. Esta comparación ayudaba a refinar los conocimientos de programación del alumno.

Más tarde, la tortuga física migró a simuladores en los que los dibujos se plasmaban en la pantalla de un ordenador. El éxito de Logo fue tal que hasta la fecha se conocen más de 180 variantes del lenguaje (Boytchev, 2007) y se han diseñado todo tipo de entornos y herramientas (tanto académicas como comerciales) basadas en Logo. Además, Logo no sólo ha sido usado en la enseñanza de la programación sino que se ha empleado en otros campos como la enseñanza de geometría o de física (Edwards, 1998).

Volviendo al ámbito de la programación, el micromundo de Logo ha ampliado sus metáforas. Uno de los micromundos más conocidos para la enseñanza de la programación es Karel the Robot (Pattis, 1995). Su primera versión, creada por Richard Pattis, se ha empleado en la enseñanza de la programación estructurada. Usa un lenguaje similar a Pascal para programar el comportamiento del robot llamado Karel, que habita un mundo compuesto de calles —que van de este a oeste— y avenidas —que van de norte a sur. En el mundo también puede haber paredes que bloquean estas calles y avenidas.



(a) Karel the Robot (b) Karel J Robot

Figura 3.7: Distintas versiones de Karel the Robot: (a) La original de Richard Pattis y (b) Karel J Robot, una versión orientada a objetos en Java.

Karel se puede mover de intersección a intersección siempre que no haya una pared que le bloquee el paso. En las intersecciones puede haber zumbadores (*beepers*) que el robot puede coger o dejar. Karel puede realizar giros de 90 grados, conoce hacia qué punto cardinal está mirando, detectar si hay un zumbador u otro robot en la intersección en la que actualmente se encuentra y transportar, coger y dejar zumbadores. El mundo y las acciones del robot sobre el mismo son representadas visualmente, como se puede ver en la Figura 3.7(a).

Debido a su popularidad, Karel the Robot ha sido versionado para distintos lenguajes de programación. De especial interés para esta Tesis son aquellas versiones que han sido modificadas para la enseñanza de la programación orientada a objetos. La primera de ellas fue Karel ++ (Bergin et al., 1997), que usaba el lenguaje de programación C++. Posteriormente y debido a la popularidad del lenguaje Java, se crearon JKarelRobot (Buck y Stucki, 2001) y Karel J Robot (Bergin et al., 2005), este último en la Figura 3.7(b). Además de estas implementaciones cabe destacar la versión descrita en (Becker, 2001), que amplía el número de clases de objetos que pueden aparecer en el mundo (por ejemplo, añade semáforos), incluyendo nuevos conceptos de programación orientada a objetos que no estaban tratados con las anteriores versiones, como es el caso de la herencia y el polimorfismo.

Otros autores han desarrollado nuevas metáforas que, en el fondo, son evoluciones de Karel. Éste es el caso de Jeroo (Sanders y Dorn, 2003a,b), una evolución de un simulador y entorno de programación conocido como Jessica para enseñar programación usando como lenguaje Pascal. Esta herramienta fue creada en 1990 por Lai Kuan Tong y fue usada durante varios años en la Illinois State University. La metáfora del simulador Jeroo (en la Figura

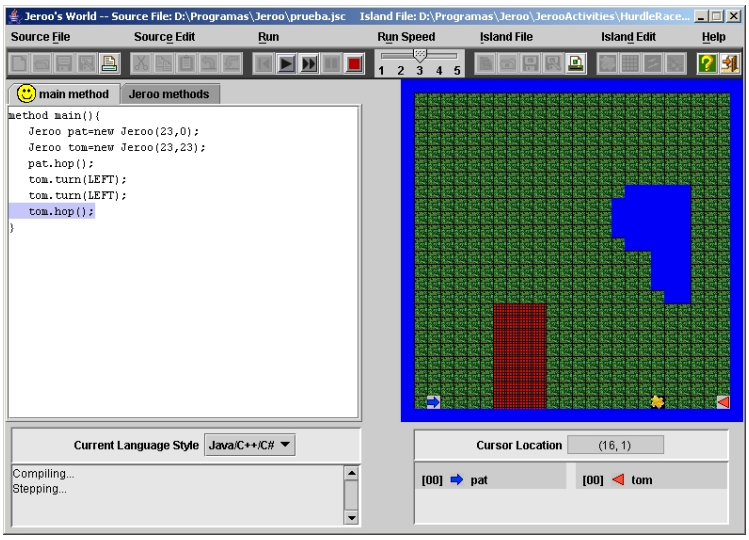


Figura 3.8: Aspecto del micromundo Jeroo.

3.8) es muy similar a la que hay tras Karel. Los robots han sido sustituidos por jeroos, una especie particular de canguro que vive en la isla Santong. Un jeroo se mueve saltando en una de las cuatro direcciones de la brújula (N, S, E, O). En la isla también hay flores Winsum, de las que el jeroo se alimenta. El jeroo puede coger flores, plantarlas, dárselas a otro jeroo o lanzarlas sobre las redes de los cazadores para neutralizarlas. En (Sanders y Dorn, 2004) la metáfora ha sido ampliada incluyendo en el mundo lagos que el jeroo ha de evitar, ya que no sabe nadar. Esta metáfora ha sido construida a lo largo de los años usando los comentarios de los alumnos, lo que hace de ella una metáfora no técnica, no violenta, cómoda de usar y muy fácil de aprender para los alumnos (Dorn y Sanders, 2003; Sanders y Dorn, 2004).

Los micromundos vistos hasta ahora tienen una metáfora fija. Pero también existen otro tipo de entornos que son usados para que el alumno pueda crear sus propios micromundos. Un ejemplo de este tipo de entornos es Alice, desarrollado por un grupo de investigación de la universidad de Carnegie Mellon. Alice está a caballo entre un micromundo y un entorno de programación por manipulación directa. Inicialmente, Alice nació como un sistema de prototipado rápido para entornos de realidad virtual (Pausch et al., 1995). Más tarde, se propuso usar este sistema como entorno para enseñar gráficos 3D a alumnos noveles (Conway et al., 2000). Por último, Alice fue usado para enseñar programación (Cooper et al., 2000; Dann et al., 2000, 2001). Actualmente se emplea como entorno virtual 3D con el que enseñar conceptos de programación orientada a objetos (Cooper et al., 2003; Dann et al., 2003).

Alice consiste en un mundo virtual que el alumno puede poblar con ob-

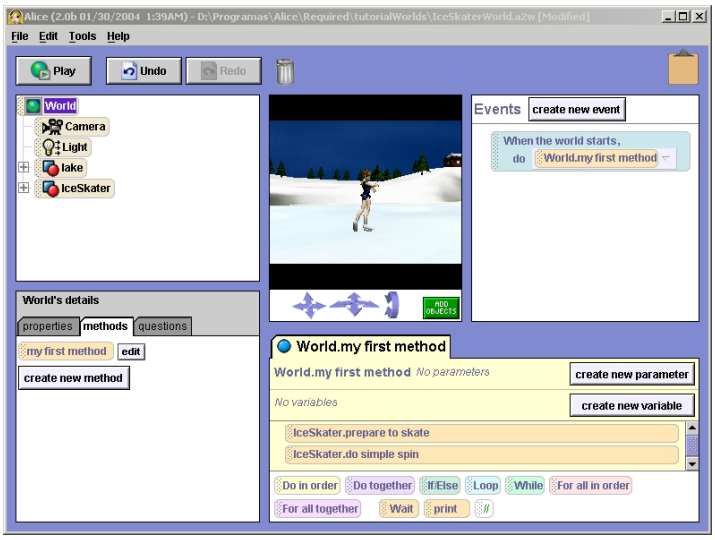


Figura 3.9: Aspecto del entorno Alice.

jetos tridimensionales animados. Un ejemplo de este tipo de mundo se puede ver en la Figura 3.9. Existe toda una biblioteca de objetos que el alumno puede situar en el mundo con tan solo pinchar y arrastrar. Cada uno de los objetos encapsula un conjunto de atributos privados que representan su estado. Además, el alumno puede programar los métodos que definen su comportamiento utilizando un conjunto de operaciones básicas y que modificarán dichos atributos haciendo uso de un lenguaje visual.

Este micromundo es especialmente motivante para los alumnos debido a que en él pueden sentirse como directores de cine que crean sus propias historias o películas: diseñan las escenas, seleccionan a los actores, crean los guiones de comportamiento de los actores y observan si la escena queda como ellos tenían en mente. Sus autores destacan que esta metáfora (*storytelling*) motiva especialmente a las estudiantes de informática, acostumbradas a convivir en un entorno en el que la tipología de las actividades realizadas suele ir dirigida especialmente para los alumnos de género masculino. Además, haciendo uso de Alice, el alumno puede crear mundos interactivos, similares a juegos de computador, ya que Alice soporta que el alumno programe el comportamiento de los objetos en respuesta a un conjunto reducido de eventos.

Existen otros entornos de creación de micromundos en los que la creación de los tipos de objetos que poblarán el mundo se hace de manera mucho más cercana a la programación tradicional. Un ejemplo es Greenfoot (Henriksen, 2004; Henriksen y Kölling, 2004; Kölling y Henriksen, 2005), un entorno para la creación de simulaciones interactivas en 2D. Está basado en la implementación de BlueJ y extiende la idea del banco de objetos de éste,

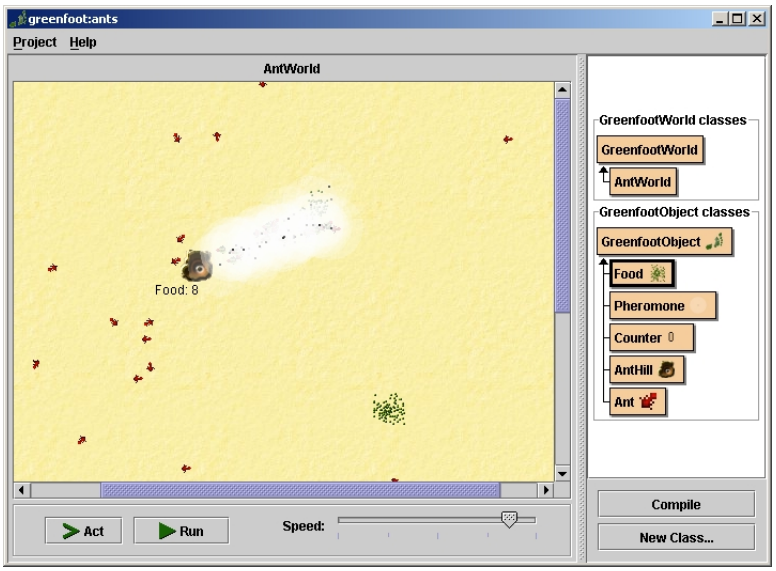


Figura 3.10: Ejemplo de un micromundo creado con Greenfoot.

convirtiéndolo en un mundo de objetos con una representación visual y un comportamiento observable mediante animaciones. Cada objeto tiene una apariencia, rotación y posición en el mundo y en él se puede observar su comportamiento. Un ejemplo de escenario en Greenfoot puede verse en la Figura 3.10.

Al igual que Alice, Greenfoot permite la creación de mundos interactivos que responden a determinados eventos de ratón y teclado. Pero, en lugar de soportar la metáfora del *storytelling*, Greenfoot se emplea principalmente para que el alumno cree simulaciones.

No podemos olvidar un ejemplo muy particular de micromundo: Lego Mindstorms. Esta herramienta nos devuelve a la época de Papert y de su tortuga mecánica programable en Logo ya que Lego Mindstorms es un conjunto de piezas de Lego al que se la ha añadido un micro-computador y un conjunto de sensores para crear robots de todo tipo (Ferrari et al., 2001; Boogaarts et al., 2007).

Las primeras versiones de Lego Mindstorms iban acompañadas de Lego RCX (Figura 3.11(a)), un computador de 8 bits (con un procesador Hitachi H(300L) con 22Kb de memoria (de los que 16Kb pertenecen al firmware, dejando libres sólo 6Kb para programas de usuario) montado sobre una pieza de Lego de 3x2x2.5 pulgadas. Incluye tres conectores para sensores (entrada), tres conectores para motores y una pequeña pantalla LCD (salida), y un puerto de infrarrojos para la comunicación con otros RCX y para la carga de los programas de usuario (Ferrari et al., 2001). Este computador está incluido en el Mindstorms Robotics Invention Kit junto con 717 piezas Lego, motores

(a) Lego RCX (b) Lego NXT

Figura 3.11: Los computadores de Lego Mindstorms: (a) Lego RCX y (b) Lego NXT.

y sensores de tacto y luz. Además, se incluye un manual para la construcción de un robot básico y un CD que contiene un entorno de programación visual (RCX Code) con el que programar nuestros robots.

Aunque ésta es la versión de Lego más usada en la literatura, actualmente Lego está comercializando el Lego NXT (Figura 3.11(b)), un computador mucho más avanzado, con un procesador ARM de 32 bits con 64Kb de memoria, cuatro entradas analógicas y digitales y 3 salidas para servomotores con sensores de rotación. La comunicación con otros NXT se realiza por Bluetooth y dispone de un altavoz y una pantalla LCD de 100x64 píxeles. El kit incluye sensores de presión, sonido, luz y ultrasónicos.

Lego Mindstorms ha sido empleado en colegios como introducción a la mecánica, robótica y programación. En el ámbito de enseñanzas superiores, Lego Mindstorms es usado en cursos de ciencia e ingeniería (Beer et al., 1999), inteligencia artificial y robótica (Klassner, 2002), introducción a la programación (Fagin y Merkle, 2002) y planificación y diseño de proyectos (Wolz, 2001).

La forma clásica de emplear Lego Mindstorms en la enseñanza de programación consiste en proporcionar al alumno un robot ya montado —el alumno no ha de preocuparse en montar desde cero el robot sino que es el instructor quien se lo proporciona ya ensamblado— y proponerle una serie de problemas que han de ser resueltos mediante la programación del robot. Los problemas más comúnmente empleados son laberintos, exploración de habitaciones o seguimiento de una línea, entre otros. El alumno ha de resolver estos problemas programando el comportamiento del robot. Aunque Lego Mindstorms viene con su propio firmware y lenguaje de programación, suele ser muy común cambiar este firmware para usar otros lenguajes más apropiados para la disciplina que se está enseñando.

La principal ventaja del empleo de Lego Mindstorms en la enseñanza de la programación es la motivación que proporciona al alumno observar cómo un programa codificado por él mismo se ejecuta en un objeto tangible del mundo real. El robot es el medio por el que el alumno visualiza el comportamiento de su programa. También se ha destacado su valía en la enseñanza de resolución de problemas usando una metodología basada en análisis, diseño, construcción y prueba (Flowers y Gossett, 2002).

Por último, hay que destacar otra herramienta para la generación de micromundos cuya principal característica es favorecer la creación colaborativa de los mismos. Nos referimos a MUPPETS (*Multi User Programming Pedagogy for Enhancing Traditional Study*) (Phelps et al., 2003; Phelps y Parks, 2004; Bierre y Phelps, 2004), un entorno multi-usuario virtual colaborativo para la enseñanza de la programación. En él, los estudiantes disponen de un avatar que se mueve por un mundo virtual similar al de un videojuego masivo multijugador y en el que puede programar y crear objetos en Java haciendo uso de un IDE integrado en el propio micromundo, como se puede apreciar en la Figura 3.12.

En este entorno el alumno no sólo puede crear objetos predefinidos y ver su comportamiento sino que también es capaz de compartir dicho código con otros alumnos o de crear nuevos objetos de manera colaborativa. Algo totalmente novedoso en MUPPETS es la capacidad de comunicación entre alumnos. En este entorno, los alumnos pueden comunicarse utilizando una herramienta de chat y sus avatares disponen de animaciones gestuales que favorecen la comunicación. Por tanto, a las capacidades pedagógicas de los micromundos (un medio de aprendizaje activo que sigue el modelo constructivista) hay que añadirle las ventajas del trabajo colaborativo, en el que cada alumno aprende no sólo de sí mismo sino también del resto de alumnos con los que comparte la actividad. Además, este tipo de entornos también fomenta otras habilidades sociales como el trabajo cooperativo en grupo.

3.5. Frameworks y casos de estudio

Los IDEs y los micromundos se han revelado como herramientas centradas principalmente en el alumno. En ellas, el alumno puede ser autónomo e interactúa directamente con ellas para el desarrollo de una serie de tareas. Pero existen otro tipo de recursos usados en la enseñanza de la orientación a objetos que también dan soporte al profesor. Destacamos los frameworks y los casos de estudio. Este tipo de recursos son usados por los profesores con dos objetivos fundamentales (Proulx et al., 1996):

- Servir de ayuda para la construcción sencilla de aplicaciones sobre las que el alumno pueda observar y modificar el comportamiento de las mismas. Esto se debe a que estos recursos se suelen caracterizar por facilitar el uso de los gráficos y de la programación de interfaces.

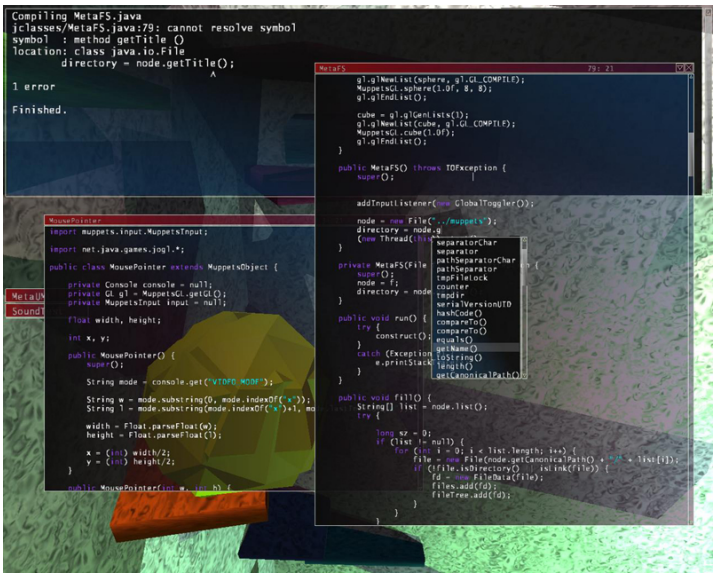


Figura 3.12: Aspecto del MUPPETS.

- Servir de ejemplo de buenos diseños orientados a objetos. Además, suelen ser usados para que los alumnos aprendan a leer y a comprender código de programas orientados a objetos.

El uso de recursos para el desarrollo de aplicaciones en cursos de iniciación suele despertar cierta controversia entre la comunidad de enseñanza de programación a objetos. En (Rasala, 2000) se citan algunas de las objeciones que se suelen realizar hacia estas herramientas y las respuestas a estas objeciones:

- La mayoría de estos frameworks no se adecúan a los que realmente se usan en el mercado laboral (como por ejemplo, Swing), por lo que no les servirá a los alumnos en el futuro. Los autores argumentan que no se pretende enseñar un framework. Éste es el medio usado como apoyo para la enseñanza y contiene ideas de diseño útiles para los alumnos. Además, estos recursos eliminan las deficiencias pedagógicas de las herramientas profesionales. No se pretende enseñar al alumno la tecnología que se encontrará en la industria, sino formarle en destrezas y habilidades que le servirán en el futuro independientemente de cuál sea la tecnología empleada.

- Los frameworks son complejos de usar, por lo que añaden al currículo la necesidad de aprender a usarlos. Aunque ésta es posiblemente una de las principales objeciones del uso de este recurso, los autores promueven su uso debido a que consideran que sus ventajas son mayores que la carga de aprendizaje que acarrean. Los frameworks facilitan mucho el

trabajo de desarrollo de alumnos y profesores, sobre todo en la creación de aplicaciones gráfica e interfaces de usuario, e insisten en las buenas ideas de diseño que hay tras ellas.

- El uso de aplicaciones gráficas distrae al alumno de los conceptos que son verdaderamente interesantes. Los autores se defienden esgrimiendo la motivación del alumno. Los alumnos prefieren un entorno visual a una consola en la que se observan una sucesión de mensajes como prueba del funcionamiento del sistema. Además, a medida que los tiempos avanzan, los alumnos se sienten más perdidos con estas consolas que con las interfaces gráficas, ya que las nuevas generaciones sólo han convivido con este tipo de interfaz de comunicación.

Muchos de estos frameworks han sido desarrollados como medio para facilitar la creación de interfaces gráficas de usuario (GUIs, del inglés *Graphical User Interface*). Un ejemplo de ello es Objectdraw (Bruce et al., 2001a), un framework para la construcción de GUIs muy sencillas: solamente permite eventos de ratón, dibujado de algunas figuras simples (rectángulos, arcos, óvalos, líneas, texto e imágenes) y animaciones simples. A grandes rasgos, ObjectDraw es una fachada que simplifica el uso de los frameworks de diseño de las GUIs de Java: AWT y Swing.

ObjectDraw se ha empleado para dar soporte a una metodología de enseñanza de programación orientada a objetos que introduce desde muy al principio la programación dirigida por eventos. Aunque se considera que este tipo de programación es demasiado complicada para alumnos iniciados, el autor rebate esta afirmación indicando que el problema se encuentra en el uso de los recursos usados para programadores profesionales, no en la complejidad de este tipo de programación en sí misma. En cambio, el diseño de Objectdraw simplifica el uso de este tipo de programación a los estudiantes (Bruce et al., 2001b). Además, el autor también destaca dos razones, descritas anteriormente, por las que emplear la programación basada en eventos en cursos introductorios: los programas modernos son desarrollados siguiendo este paradigma y el uso de GUIs es más motivante para los alumnos que la clásica entrada y salida textual (Bruce et al., 2004).

Siguiendo la misma línea de trabajo que ObjectDraw se encuentra Java Power Tools (JPT) (Rasala y Proulx, 2005), otro framework diseñado y usado en la Universidad de Northeastern para la creación de interfaces gráficas de usuario. JPT es el resultado de la evolución de múltiples herramientas y ejercicios creados en esta universidad desde principios de los 90 (Pascal (Brown et al., 1993), C++ (Fell et al., 1996; Rasala, 2000) y, finalmente, Java (Raab et al., 2000; Proulx et al., 2001; Rasala et al., 2001; Proulx et al., 2002)). Los dos principales objetivos de este framework son, al igual que el anterior, facilitar a los alumnos el desarrollo de GUIs y servir de ayuda para que los instructores creen sus propios ejemplos y ejercicios basados en

visualizaciones que ayuden a los alumnos a aprender los conceptos básicos de la programación orientada a objetos, algoritmos, estructuras de datos y diseño de interfaces.

La complejidad del diseño de JPT es mayor que la de ObjectDraw, por lo que sus autores destacan un tercer objetivo pedagógico de JPT: su uso como caso de estudio para cursos avanzados de orientación a objetos. Este framework ha sido cuidadosamente diseñado y documentado para ser objeto de estudio en cursos de diseño más avanzados. Además, se encuentra plagado de patrones de diseño, lo que permite que los alumnos puedan tener ejemplos de uso de estos artefactos.

Los frameworks tienen unas particularidades de diseño que los hacen especialmente interesantes para mostrar el uso de patrones de diseño. Por ejemplo, Erich Gamma, uno de los autores del libro más conocido sobre patrones de diseño (Gamma et al., 1995), desarrolló junto a Thomas Eggenschwiler un framework para la creación de editores gráficos de figuras en dos dimensiones que sirviera como medio para ver estos patrones de diseño en acción. Este framework se conoce como JHotDraw (JHotDraw; Kaiser, 2001) y está inspirado en HotDraw, su antecesor en SmallTalk, y en ET++, un framework más una librería de clases para el desarrollo de GUIs en C++. JHotDraw proporciona un conjunto de elementos básicos para la creación sencilla de editores gráficos (de aspecto similar al de la Figura 3.13), como la creación de distintos tipos de figuras y herramientas de manipulación de las mismas. Además del ámbito académico, este framework ha sido empleado en la construcción de editores gráficos especializados para el diseño de redes de Petri, redes neuronales, sistemas software orientados a objetos, sistemas distribuidos o sistemas de automoción.

Este framework posee una serie de características que lo hacen especialmente útil para la pedagogía: (1) Es de código libre, por lo que se puede revisar y modificar su código fuente; y (2) tiene gran cantidad de documentación. Esto se debe a que, además de la documentación desarrollada por los autores y que acompaña al framework, muchos investigadores han desarrollado, tanto para JHotDraw como para HotDraw, nuevas aproximaciones de documentación específica para frameworks.

El diseño de un editor con JHotDraw supone un esfuerzo menor, en comparación con la creación de una aplicación desde cero. Por ejemplo, el simulador que se observa en la Figura 3.13 fue desarrollado creando no más de 10 clases y aproximadamente 300 líneas de código. A pesar de ello, no hay que olvidar que esta facilidad de desarrollo se traduce en una alta complejidad de su diseño. Está compuesto por cerca de doscientas clases distribuidas en ocho paquetes y se ha hecho un uso intensivo de los patrones de diseño, lo que hacen de él un caso de estudio sólo apto para cursos avanzados de diseño orientado a objetos.

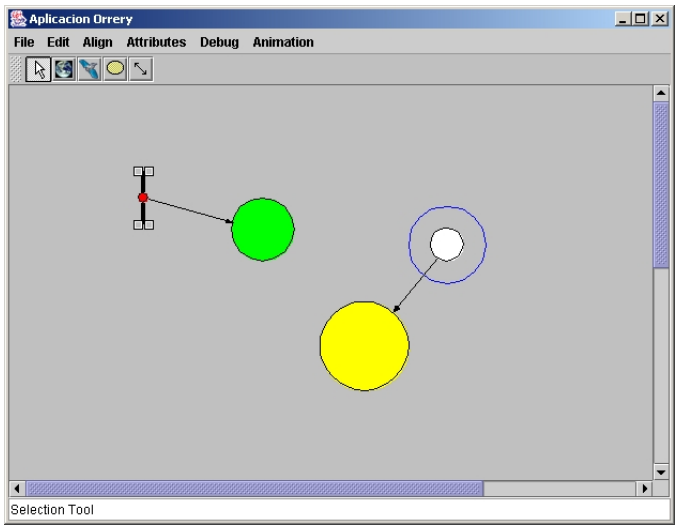


Figura 3.13: Ejemplo de un simulador de un sistema solar creado usando JHotDraw.

Volviendo a la enseñanza de la orientación a objetos a un nivel mucho más elemental hay que destacar un caso de estudio ya señalado en el capítulo anterior que es ampliamente usado y citado en numerosos trabajos: el simulador de biología marina (MBSCS). Este caso de estudio fue desarrollado como parte del temario relacionada con la enseñanza de programación en el Advanced Placement del College Board. Este caso de estudio describe el desarrollo de un entorno visual en forma de cuadrícula en el que se puede ver nadar a una colección de peces. El aspecto de la aplicación a desarrollar puede verse en la Figura 3.14.

Este caso de estudio incluye un conjunto de actividades para que el alumno se familiarice con la aplicación. Posteriormente propone realizar extensiones del mismo creando nuevos tipos de peces y modificando la funcionalidad y las capacidades del entorno. Durante el curso 2007-2008 se ha abandonado este caso de estudio para pasar a utilizar uno más genérico: GridWorld (GridWorld). Este caso de estudio también describe el desarrollo de un entorno visual que representa un mundo en forma de cuadrícula, esta vez poblado por insectos. A diferencia del simulador de biología marina en el que sólo había peces, los insectos no son los únicos habitantes del entorno sino que existen otros tipos de entidades y criaturas con distinto comportamiento, como flores y rocas. Este entorno ha sido generalizado de tal forma que permite abandonar el ámbito del mundo poblado por insectos para realizar todo tipo de aplicaciones que usen como base una cuadrícula, como un *Sudoku* o un *Conecta-4*, entre otros[4]. En la Figura 3.15 se puede ver el aspecto

[4]Este material está accesible desde la página de AP Central College Board (`http://apcentral.collegeboard.com/`), previo registro de usuario (último acceso: 15-02-2008)

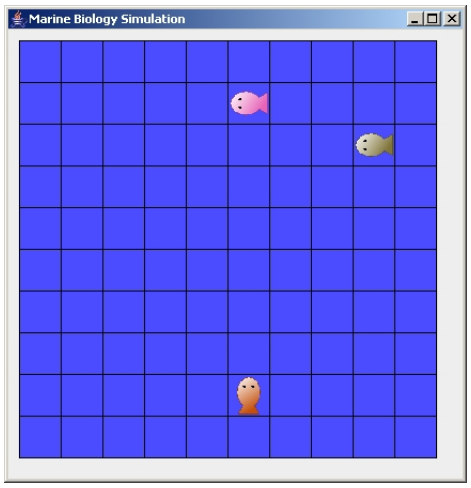


Figura 3.14: Aspecto del simulador de biología marina usado como caso de estudio.

de este entorno.

3.6. Tratamiento de las dificultades de la orientación a objetos

Como ya se ha descrito en el Capítulo 2, la enseñanza de la orientación a objetos conlleva una serie de dificultades que el instructor ha de tener en cuenta para hacer más efectivo el proceso de enseñanza de esta disciplina. Estas dificultades también deben ser consideradas, en el diseño de los distintos recursos de apoyo a la enseñanza de la orientación a objetos.

A continuación se analizan los medios empleados por los distintos recursos revisados para tratar las principales dificultades encontradas en la enseñanza de la orientación a objetos. Así mismo, también se destacan los inconvenientes de estos recursos, de modo que sean evitados en el desarrollo de futuros recursos de apoyo.

3.6.1. El lenguaje

Es un hecho que la orientación a objetos suele impartirse ligada a un lenguaje de programación. Este hecho queda claramente patente en los recursos mostrados. Esto se debe a que estos recursos se encuentran especialmente enfocados a la enseñanza de la *programación orientada a objetos*. Como se irá viendo, muy pocos dan especial interés a actividades de diseño.

Como se puede ver, las herramientas más antiguas usadas para la enseñanza de la programación orientada a objetos comenzaron utilizando C++,

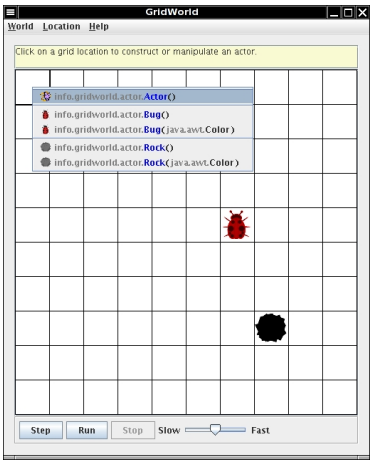


Figura 3.15: Aspecto del caso de estudio GridWorld.

lenguaje utilizado durante muchos años para la enseñanza de esta disciplina. Posteriormente, esta comunidad tomó la decisión de usar un lenguaje profesional pero que fuera más sencillo para los alumnos, decantándose por Java. De acuerdo a esto, los entornos que usaron C++ fueron migrando al lenguaje Java. Esta evolución se puede ver claramente en Karel the Robot, en el simulador de biología marina, en algunos *firmware* de Lego Mindstorms o en la evolución de Java Power Tools o de Jeliot. Además, los entornos más modernos han optado directamente por Java como lenguaje de programación.

A pesar de la predominancia de estos lenguajes, se ha podido observar que algunas herramientas optaron inicialmente por otros lenguajes orientados a objetos. Por ejemplo, Alice comenzó usando Python como lenguaje de programación, aunque rápidamente se recondujo a la senda de C++ y de Java. Kölling desarrolló el lenguaje Blue a partir de su experiencia en la enseñanza de la orientación a objetos con otros lenguajes (Kölling, 1999a). Incluso llegó a crear el editor para este lenguaje. Pero la predominancia de Java hizo que este editor fuese modificado para convertirse en el que actualmente se conoce como BlueJ. Otro ejemplo es Lego Mindstorms, que para el ámbito universitario proporciona un entorno de desarrollo basado en su propio lenguaje de guiones llamado Mindscripts, y otras utilidades para programar el robot usando Visual C++y Visual Basic. Además de los proporcionados por Lego, han surgido una gran variedad de firmwares y kits de programación asociados a los primeros y que pueden sustituir al original. Se conocen interfaces de programación para C (NQC -Not Quite C), FORTH (pbFORTH), SmallTalk (Bot-Kit), Tcl (TclRCX), C++ (legOS, actualmente conocido como BrickOS) y Java (leJOS), entre otros (Patterson-McNeill y Binkerd, 2001). Este último es uno de los más empleados en cursos relacionados con enseñanza de orientación a objetos.

Cabe destacar que los micromundos usados para la enseñanza de la orientación a objetos no suelen utilizar lenguajes propios. Generalmente, un micromundo posee un lenguaje específico muy sencillo con el que programar el comportamiento de los objetos que lo habitan. En cambio, los micromundos usados para la enseñanza de orientación a objetos usan lenguajes de más alto nivel (Java, mayoritariamente) para programar el comportamiento de dichas entidades, de igual forma que si se estuviera programando el comportamiento de una clase y siguiendo las reglas del paradigma orientado a objetos.

También es reseñable el esfuerzo de algunos entornos por no ligarse a ningún lenguaje concreto. Estas aproximaciones hacen que el alumno se libre de una sintaxis, de modo que se pueda centrar en los conceptos de la orientación a objetos sin distrarse por los errores sintácticos de programación. Por ejemplo, Lego Mindstorms viene acompañado de un lenguaje visual, RCX Code. Si lo que se desean son otros entornos más avanzados, se puede conseguir RoboLab, otro lenguaje visual con un entorno muy similar al anterior. Pero estos lenguajes no suelen ser demasiado útiles para la enseñanza de la orientación a objetos ya que han sido pensados para expresar el comportamiento del robot de manera procedimental.

Otro ejemplo de este tipo de lenguajes visuales es el usado en Alice. El editor de código de este entorno se emplea para que el alumno programe el comportamiento de los objetos de la escena. Este editor es visual, de modo que el alumno genera el código arrastrando paneles de métodos y propiedades de los objetos o construcciones del lenguaje (`if, while, for`...). Si son necesarios, el editor solicitará los parámetros actuales que un método espera y el alumno lo rellenará de nuevo arrastrando objetos o seleccionándolo de una lista de posibles valores. El lenguaje en el que se expresa el comportamiento es configurable por el usuario, pudiendo elegir entre un lenguaje más "narrativo" o una sintaxis más similar a Java/C++.

Otro ejemplo es JPie que, como ya se describió en la Sección 3.3, usa el mecanismo de incluir cápsulas en regiones semánticas para programar. La particularidad de JPie es que por debajo de todas estas cápsulas y regiones realmente está el lenguaje Java. El lenguaje visual que se enseña no es único para este entorno sino que solamente es un medio para ocultar la sintaxis de Java. De hecho, este entorno tiene cápsulas para cualquier clase y paquete de Java, así como una serie de enlaces directos a las clases más comunes de Java y a los tipos primitivos.

Aunque este tipo de lenguajes visuales elimina los errores sintácticos del alumno hay que tener en cuenta que, en la mayoría de los casos, solamente son útiles para programas muy sencillos. En cuanto las estructuras de control y las expresiones que los alumnos han de programar se hacen un poco más complejas, estos entornos se vuelven lentos, tediosos y frustrantes. Por ejemplo, en JPie las estructuras condicionales se construyen siguiendo su estructura arbórea y no de manera lineal, por lo que una expresión como `x>0`

`AND x<n` se ha de construir añadiendo una expresión `AND`, en cuyos extremos tiene las expresiones `>` y `<`, sobre las que habrá que añadir las variables `x` y `n` y la constante `0`. A pesar de que el entorno incluye una herramienta similar a una calculadora que puede simplificar un poco esta construcción, lo cierto es que, una vez adquiridas ciertas destrezas, el entorno se vuelve demasiado pesado para programar. Por esto, las últimas versiones de este entorno combinan la programación textual con la visual, de modo que la primera será la predominante a medida que el alumno vaya adquiriendo más destrezas (Birnbaum y Goldman, 2005).

Muy pocos son los casos en los que la herramienta o recurso se centra en enseñar diseño, dejando de lado un lenguaje de programación concreto. En Fujaba tanto el diseño de las clases como su comportamiento se realiza usando un lenguaje visual. El usuario crea la arquitectura de la aplicación mediante la construcción de su diagrama de clases UML. Para cada clase, se define sus métodos y atributos y sus relaciones con otras clases. Una vez especificada la estructura estática de las clases, el comportamiento de las mismas se describe mediante lo que se conoce como diagramas de historia (*story diagrams*), un lenguaje de grafos de reescritura para el modelo de la orientación a objetos (Fischer et al., 1998). Un diagrama de historia (como el de la Figura 3.16) consiste en un conjunto de actividades conectadas entre sí para representar la secuencia de ejecución de las mismas. Las transiciones pueden bifurcarse indicando las guardas con las que decidir por dónde seguir. Las actividades en los diagramas de historia pueden ser actividades de instrucciones o patrones de historia. Los primeros son fragmentos de código Java (permite añadir entrada-salida, instrucciones matemáticas o cualquier otra instrucción que no seamos capaces de representar con estos diagramas). Los patrones de historia representan reglas de reescritura usando diagramas de colaboración UML.

También es destacable que estos lenguajes visuales tengan una traducción a lenguajes profesionales como Java, de modo que a medida que el alumno tenga mayores conocimientos pueda ver cuál es la traducción de lo que ha hecho a un lenguaje profesional. Fujaba es capaz de evaluar los diagramas de historia y generar su código Java asociado. En cambio, aunque JPie proporciona la misma funcionalidad, el resultado no es bueno desde el punto de vista académico: aparecen clases nuevas que pueden desconcertar al alumno e incluye código con distintas hebras de ejecución.

3.6.2. El uso de ejemplos

Ya es conocida la necesidad de enseñar orientación a objetos haciendo uso de ejemplos útiles para los alumnos. La mayoría de los desarrolladores de las herramientas dedicadas a la enseñanza de este paradigma son conscientes de ello, por lo que suelen acompañarlos de ejemplos que faciliten la tarea del instructor. Pero la forma de los ejemplos y el número de ellos no es igual en

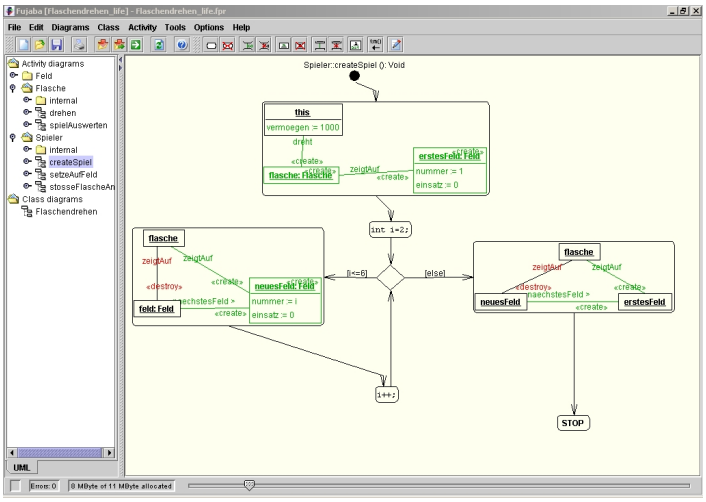


Figura 3.16: Ejemplo de un diagrama de historia de Fujaba.

las herramientas revisadas.

En el caso de los IDEs es donde más carencias de ejemplos se encuentran. Los IDEs son entornos de creación por lo que algunos desarrolladores sólo se han molestado en incluir los ejemplos básicos necesarios para que un instructor comprenda el funcionamiento de la herramienta. Este es el caso de Fujaba, JPie y Jeliot. En el primero los ejemplos existentes son muy escasos y de complejidad muy similar. Además, no dispone de una buena documentación y la poca disponible se encuentra en alemán. JPie incluye pocos ejercicios. En algunos no suele haber más de una clase y otros, sin embargo, contienen conceptos muy complejos para alumnos novatos, como redes y concurrencia. En el caso de Jeliot, no hay ejemplos o ejercicios que puedan ser usados en programación orientada a objetos. De hecho, los ejemplos que acompañan a la versión de distribución son meramente procedimentales.

Pero ésta no es la tónica general de los IDEs ya que algunos autores se han molestado en construir cursos guiados por ejemplos que han sido generados usando el entorno. Por ejemplo, los autores de BlueJ tienen una guía de enseñanza de programación orientada a objetos dirigida por proyectos y en la que se incluye el uso de su herramienta (Barnes y Kölling, 2005). Una de las ventajas de ejemplos dirigidos por proyectos es que al alumno se le presenta un problema que ha de analizar y entender y sobre el que se propone un diseño razonado, por lo que se le están introduciendo ideas de diseño desde el principio.

En el caso de los micromundos esa necesidad de ejemplos es aún mayor, ya que el instructor se encuentra con un mundo con unas reglas prefijadas y con una serie de entidades ya construidas. Por ejemplo, Jeroo es un micromundo en el que las acciones están muy limitadas: sólo se puede programar

el comportamiento de los jeroos haciendo uso de los métodos de los que dispone y modificando alguno de sus tres atributos (su posición en la isla, la dirección en la que mira y el número de flores que transporta). En estos micromundos el instructor necesita de gran ayuda para obtener el máximo beneficio de ellos. En caso contrario, se puede encontrar con que dispone de una herramienta que se supone que le va a ayudar pero que no sabe cómo emplear. Éste es uno de los grandes problemas de MUPPETS, ya que es un micromundo de grandes posibilidades pedagógicas pero que carece de ejemplos de uso en enseñanza.

Por otro lado, los micromundos suelen ser entornos muy útiles para mostrar conceptos elementales relacionados con el comportamiento de los objetos pero que caen en el error de tener ejemplos demasiados sencillos, en los que no hay más que una clase y una instancia de dicha clase. Por ejemplo, las distintas versiones de Karel tienen un gran número de ejercicios y una guía de empleo de Karel, pero que tan solo es útil para la impartición de cursos elementales (Pattis, 1995; Bergin et al., 2005). Para este tipo de entornos es de agradecer aportaciones como la descrita en (Becker, 2006), que ha aumentado el número de clases de su versión de Karel para que pueda ser usado en cursos más largos y en los que se traten todos los conceptos relacionados con la orientación a objetos.

La principal ventaja de los micromundos que sirven para la creación de nuevos micromundos es que los ejemplos que se pueden construir tienen toda la complejidad que el instructor desee. Como ejemplo, los autores de Alice han creado un libro con ejercicios para tratar los conceptos fundametales de la orientación a objetos y que facilita la transición de los alumnos a lenguajes profesionales como C++ o Java (Dann et al., 2005). En el caso de Greenfoot, existe una comunidad muy activa en la creación de nuevos micromundos, como se puede ver en su página web[5]. De hecho, se ha celebrado incluso una competición para la creación de nuevos micromundos usando Greenfoot[6].

La necesidad de este tipo de ejemplos es absoluta en herramientas como Lego Mindstorms cuyo uso en la enseñanza de orientación a objetos no es obvio. En (Lawhead et al., 2003) y (Barnes, 2002) se detalla el uso de Lego Mindstorms con el firmware leJOS para la enseñanza de la programación orientada a objetos. Aquí se detalla el diseño de algunos robots y el uso de los mismos para enseñar algunas de las características básicas de la programación orientada a objetos, como la invocación de métodos, el paso de parámetros, múltiples instancias de una clase o la herencia, así como otros conceptos de programación como secuencias de instrucciones, bucles e instrucciones condicionales.

En el caso de los frameworks, disponer de una batería de ejemplos de

[5] `http://www.greenfoot.org/` (último acceso: 15-02-2008)

[6] `http://www.greenfoot.org/javaone/competition-flyer.html` (último acceso: 15-02-2008)

uso es algo fundamental para hacer uso de los mismos. En el caso de usar estas herramientas para la creación de ejemplos de aplicaciones, es fundamental que el instructor disponga de ejemplos que le den idea del tipo de aplicaciones que es capaz de desarrollar y del uso que les puede dar en las clases. Por ejemplo, los autores de ObjectDraw han desarrollado un conjunto de ejercicios para ser usados en la enseñanza de programación orientada a objetos usando esta herramienta (Bruce et al., 2005). A esto hay que añadir aportaciones de terceros autores, que han mostrado ejemplos de uso de esta herramienta en cursos introductorios (Bærbak-Christensen, 2004). Otro caso similar es JPT, que tiene una gran cantidad de ejercicios y actividades desarrolladas por sus autores y empleadas en la Universidad de Northeastern. Además, existen distintos artículos en los que los autores proponen nuevos ejercicios con distintos objetivos de enseñanza que están o pueden ser adaptados para JPT (Proulx, 1998, 1999; Rasala et al., 2002). En el caso de JHotDraw, existen varios ejemplos disponibles desde su página web y hay trabajos en los que se describen algunas actividades originales, aunque todas ellas carecen de información sobre los objetivos pedagógicos cubiertos (Kirk, 2001, 2002).

En el caso de usar los frameworks como ejemplos de buen diseño o en los casos de estudio, es necesario que estas herramientas proporcionen información detallada de su diseño así como su código fuente bien documentado. La documentación de ObjectDraw no es suficiente para poder hacer uso de él como caso de estudio. En cambio, los autores de JPT han hecho un especial esfuerzo en proporcionar una documentación mucho más completa y en describir detalladamente las principales ideas de diseño que hay tras esta herramienta (Rasala et al., 2001). En el caso de JHotDraw, su complejidad es tal que no lo hace interesante para los primeros cursos. Pero este framework está especialmente documentado para servir de ejemplo de uso de patrones de diseño.

Uno de los casos de estudio más detallados y que, por tanto, le ha hecho de mayor interés entre los docentes es el simulador de biología marina de College Board. Este caso de estudio incluye una guía detallada para usarlo como herramienta de enseñanza. Esta guía trata los principales conceptos de la orientación a objetos y propone ejercicios de observación del estado de los objetos en tiempo de ejecución, lectura y comprensión del código fuente, modificación de clases ya existentes y creación de otras nuevas o extensión de algunas de las ya existentes por herencia. Como parte de las actividades también se incluyen ejercicios de role-play con los alumnos (Andrianoff y Levine, 2002).

3.6.3. Encapsulación y abstracción

Como ya se ha mencionado en el Capítulo 2, los procesos de enseñanza ligados a un lenguaje de programación afectan a la comprensión de la abs-

tracción ya que hacen que el alumno se preocupe más por la implementación del comportamiento que por el diseño de las clases que conforman el sistema. Desde este punto de vista, una de las limitaciones de la mayoría de los entornos estudiados es que dan mucho más soporte a la enseñanza de la implementación que a la del diseño. Además, como se ha podido ver, la mayoría de los entornos estudiados quedan ligados a un lenguaje de programación concreto.

Hay que señalar que los mejores entornos para mostrar los conceptos relacionados con la abstracción son los micromundos. En ellos, el alumno puede concentrarse en las clases de objetos que pueblan el micromundo, puede crear objetos de dichas clases, modificarlos, observar su comportamiento y no tiene la necesidad de conocer cómo están implementados. Por ejemplo, en Jeroo el alumno sabe que un jeroo dispone de un conjunto de métodos para manipular su estado y su comportamiento pero en ningún caso el alumno ha de preocuparse de cómo está implementada ninguna de estas operaciones.

Los frameworks también hacen que el alumno trabaje el concepto de la abstracción pero a un nivel más avanzado. Estos entornos suelen proporcionar interfaces para que el alumno pueda construir sus propias clases, las cuales generalmente tienen una representación visual, sin tener que preocuparse en la forma en la que van a ser visualizadas por el framework. Tan solo ha de conocer los mecanismos que le proporciona el mismo para poder visualizar aquello que desea que aparezca en la pantalla de su ordenador.

Lego Mindstorms es un caso especialmente interesante para enseñar conceptos de abstracción. En este entorno, el alumno define el comportamiento del robot a nivel de clases que representan sensores y actuadores del robot. Pero el alumno no necesita conocer absolutamente nada de la manera en la que el robot ha sido construido o en las especificaciones físicas del mismo.

La encapsulación es un concepto a la que la mayoría de las herramientas no le proporcionan especial atención. Sólo aquellos que se encuentran fuertemente ligados a un lenguaje concreto de programación hacen uso de los mecanismos del propio lenguaje para preservar la encapsulación. Este es el caso de los IDEs y de los algunos de los frameworks y casos de estudio revisados. En cambio, en los micromundos hay todo tipo de posibilidades. Los que están ligados al lenguaje, como algunas versiones de Karel o Greenfoot dejan al alumno que sea él quien decida sobre la encapsulación de las clases que manipula. Alice no da alternativa al alumno, de modo que todas las propiedades o atributos de un objeto son privadas al objeto mientras que sus métodos son siempre públicos. Por último, Jeroo no tiene ninguna preocupación por la encapsulación y permite que todos los atributos y comportamientos de los jeroos sean públicos.

Un problema relacionado con la encapsulación es la inspección del estado de los objetos. La inspección es necesaria en muchas ocasiones para hacer patente que la ejecución de ciertos métodos puede suponer el cambio de

estado de un objeto. Cuando se hace uso de la inspección es necesario hacer patente al alumno que observar durante la visualización el estado de un objeto no implica que estemos accediendo a sus detalles de implementación y, por tanto, rompiendo su encapsulación.

Muchos de los entornos se despreocupan de la inspección de objetos ya que los cambios de estado de los objetos se observan de manera intrínseca durante la ejecución de las visualizaciones. Por ejemplo, en el entorno Alice muchos de los cambios de estado referentes a propiedades físicas de los objetos (como posición, color o tamaño) van acompañados de animaciones que destacan dicho cambio de estado. En la mayoría de los micromundos como Karel, Jeroo, MUPPETS, Lego Mindstorms o Greenfoot, o en casos de estudio como el simulador de biología marina, los cambios de estado se ven explicitamente mediante las variaciones en la posición de los objetos y de la dirección a la que miran. Al igual que en Alice, algunos de estos entornos añaden animaciones de transición entre cambios de estado.

Otros entornos como Jeliot optan por mostrar en todo momento el estado de los objetos en ejecución. Esta opción puede hacer que el alumno tenga demasiada información que ver por lo que no es capaz de centrar su atención en los cambios de estado interesantes. Para solventar este problema hay muchos entornos que proporcionan inspectores de objetos que permiten que el alumno pueda acceder de manera selectiva a la información privada de los objetos. Los IDEs BlueJ, Fujaba y JPie proporcionan sendas herramientas de inspección. Este último permite incluso inspeccionar varios objetos que se encuentren en el heap simultáneamente.

Algunos entornos proporcionan distintas vistas del estado del objeto. Greenfoot aprovecha su implementación basada en BlueJ para poder hacer uso del inspector de objetos de éste y poder observar aquellas propiedades del estado que no son explícitamente visibles durante la ejecución del micromundo. En Jeroo, además de las características explícitas, aparece, para cada jeroo, su nombre, su representación en el mundo y las flores que transporta en cada momento.

Por último, algunos entornos permiten modificar deliberadamente el estado de los objetos. Al igual que antes, esta característica del entorno hay que usarla siempre con cuidado en pos de que el alumno no pierda de vista el concepto de la encapsulación. El inspector de objetos de BlueJ y de Greenfoot permite modificar el valor de los atributos de cualquier objeto. En el caso de Greenfoot podemos cambiar la posición de los objetos con tan solo arrastrarlos con el ratón a otra posición del mundo. Este mecanismo es análogo al que podemos realizar de manera física con los robots de Lego Mindstorms.

3.6.4. Clases y objetos

La distinción entre clases y objetos por parte del alumno es algo crucial en la orientación a objetos. A pesar de ello, hay entornos que no se preocupan por ello. Algunos micromundos como las primeras versiones de Karel no permitían nada más que tener un robot con un comportamiento predeterminado y no modificable. Aquí el alumno no puede ver diferencia alguna entre la clase del robot y la instancia que está usando. Esto mismo ocurre con Lego Mindstorms. En el caso de Karel, la mayoría de las versiones han solventado este problema permitiendo que se puedan crear múltiples instancias de los robots.

Alice tampoco hace una distinción real entre clases y objetos y, además, sus autores dan una solución para distinguirlos cuanto menos controvertida. Según los autores, las clases son los modelos 3D de los objetos que se pueden ubicar en el mundo. Cuando esta "clase" es situada en el mundo entonces el alumno puede programar su comportamiento. Pero lo que el alumno está haciendo realmente es modificar el comportamiento de un objeto, no de una clase. De hecho, si el alumno quisiera generar una nueva instancia de la "clase" cuyo comportamiento ha modificado entonces ha de clonar el objeto que aparece en el mundo virtual.

Estos problemas no son algo general sino que la mayoría de los entornos hacen distinciones entre estos conceptos. De todas formas, algunas no son del todo claras. Por ejemplo, muchos de los micromundos permiten la creación de escenarios a partir de un conjunto básico de entidades. Por ejemplo, el entorno Jeroo dispone de una "paleta de clases" con los elementos principales que se pueden colocar sobre el entorno (jeroos, lagos, redes, flores...). Al situarlos sobre el entorno estos elementos se convierten en instancias con distinto estado (en particular, su posición es distinta). Esto suele ser similar en otros micromundos como Karel o en Greenfoot.

En cambio, la mayoría de los IDEs tienen una distinción mucho más explícita. En el caso de BlueJ y de jGrasp se puede ver simultáneamente la estructura de clases y los objetos creados, siendo ambos perfectamente distinguibles. Otros entornos como JPie y Fujaba disponen de vistas distintas para visualizar la estructura de la aplicación en tiempo de compilación (a nivel de clases) o en tiempo de ejecución (a nivel de objetos). Otros entornos como Jeliot o MUPPETS no disponen de una representación visual de las clases sino que éstas son representadas por su código fuente. Durante la ejecución, en cambio, existe una representación visual para los objetos.

Relacionado con esta distinción está la forma en la que se crean las instancias a partir de las clases que el entorno proporciona o que han sido construidas por el alumno. En el caso de las librerías, los frameworks y los casos de estudio, así como en Jeliot, los objetos sólo pueden ser creados mediante la implementación de una aplicación. En cambio, algunas herramientas como BlueJ, jGrasp, JPie o MUPPETS permiten construir instancias de un

objeto sin necesidad de crear la aplicación completa. Sobre esta instancia, el alumno puede ejecutar métodos e inspeccionar su estado. En algunos de estos entornos, además, se puede elegir el constructor con el que crear dicha instancia.

En cambio, en la mayoría de los micromundos, la creación se realiza de manera no explícita: el alumno elige una clase y la arrastra sobre el mundo, creándose una instancia de esa clase inicializada con una determinada posición. Además, estos entornos no suelen permitir la creación de otras instancias durante la ejecución. Un caso especial es Greenfoot: aunque la instanciación se realiza de manera no explícita, el alumno ha de invocar un constructor de la clase para crear un objeto antes de arrastrarlo sobre el mundo. Además, este micromundo no impide que durante la ejecución un objeto pueda crear nuevos objetos.

3.6.5. Paso de mensajes y métodos

El paso de mensajes es una de las mecánicas fundamentales de la orientación a objetos. Por esto, algunas herramientas proporcionan funcionalidades que facilitan que el alumno observe y entienda esta mecánica. Como se ha mencionado en la sección anterior, la mayoría de los IDEs tienen la capacidad de crear instancias de una clase y de invocar individualmente los métodos de esta clase. Como se puede ver, estos entornos aíslan la mecánica de modo que pueda ser ejecutada fácilmente sin necesidad de distraer al alumno con la construcción de un programa principal. Además, esta funcionalidad permite que el alumno pueda ver fácilmente que el comportamiento de un objeto depende de su estado. Esta misma funcionalidad es soportada por algunos micromundos como Greenfoot o Alice.

En el resto de entornos, en general, no es posible mostrar tan atómicamente esta mecánica. Los micromundos y los casos de estudio tan solo muestran el resultado visual del comportamiento de los objetos, de modo que el alumno ha de inferir que dicho comportamiento se debe a la ejecución del conjunto de métodos que el alumno ha programado. De hecho, la mayoría de estos entornos ni siquiera dan facilidades para que el alumno pueda asociar las instrucciones que son ejecutadas con el comportamiento que está viendo. Sólo Jeroo muestra el código del alumno mientras que se ejecuta la simulación, marcando la instrucción que actualmente está siendo invocada.

También es interesante que el alumno sea consciente de que el paso de mensajes se usa para la delegación en otros objetos para completar una responsabilidad. Al igual que antes, los micromundos y los casos de estudio carecen de medios específicos para poder observar esta mecánica. Pero también hay IDEs como BlueJ en los que no existe forma de visualizar las interacciones entre objetos. La ejecución de un método puede generar llamadas a otros métodos de otros objetos y todo esto pasa de manera transparente al alumno, hecho que no parece coherente tratándose de un IDE pedagógico.

Otros IDEs crean animaciones detalladas para observar la interacción entre objetos. Este es el caso de Jeliot, en el que la ejecución de los métodos se acompaña de una serie de animaciones que van cambiando el estado de los objetos de la aplicación. En el código fuente aparece resaltada la instrucción que actualmente está siendo ejecutada. Uno de los principales defectos de Jeliot, también destacado por sus usuarios (Kannusmäki et al., 2004), es su lentitud en la visualización. Aún a máxima velocidad la ejecución de un método algo complejo puede durar varios minutos.

La mayoría de los IDEs se apoyan en el uso de un depurador para poder observar la interacción entre los objetos. La principal diferencia de estos depuradores con los que vienen con un IDE convencional es que los depuradores suelen ir acompañados de representaciones visuales que facilitan que el alumno observe la delegación entre los objetos y los cambios de estado que se pueden producir en los mismos.

Por último, cabe destacar que el principal inconveniente de Lego Mindstorms está relacionado con este concepto de orientación a objetos y se debe a la naturaleza física de la herramienta. Los robots se mueven en un entorno real donde el rozamiento del suelo y la luz, entre otros, afectan a la ejecución del robot. Esto puede hacer que programas correctos no alcancen el resultado esperado debido a estos parámetros físicos. Por ejemplo, un alumno puede haber programado un giro de noventa grados que, en la ejecución en el entorno real, no sea tal, dando al traste con la ejecución del programa. Para evitar este problema, en algunos centros se ha trabajado en la creación de entornos de simulación que ejecutan gráficamente en la pantalla del computador el código que posteriormente se va a ejecutar en el robot. Este es el caso del entorno de simulación Jago (Figura 3.17), usado en la Academia militar de West Point (USA) (Flowers y Gossett, 2002; Schumacher et al., 2001; Wolfe et al., 2003). Jago es un entorno que permite al alumno escribir programas en un lenguaje similar a Java, ejecutarlos en un entorno gráfico y descargarlos y ejecutarlos en el robot.

3.6.6. Herencia y polimorfismo

Estos mecanismos fundamentales en la orientación a objetos no son soportados por todos los recursos analizados. Por ejemplo, en Jeroo no se pueden crear nuevas clases, por lo que no es posible explicar estos conceptos. Algo similar pasa con Alice, aunque sus autores consideren que este concepto también puede ser explicado con su entorno. Resulta complicado que una herramientas en la que la distinción entre clases y objetos sea tan difusa permita explicar unos conceptos de esta complejidad.

Algunos entornos proporcionan representaciones visuales para que el alumno sea consciente de que una clase hereda de otra. Las clases en BlueJ aparecen representadas usando diagramas UML de clase, por lo que las relaciones de herencia aparecen expresamente en los diagramas. Algo similar ocurre con

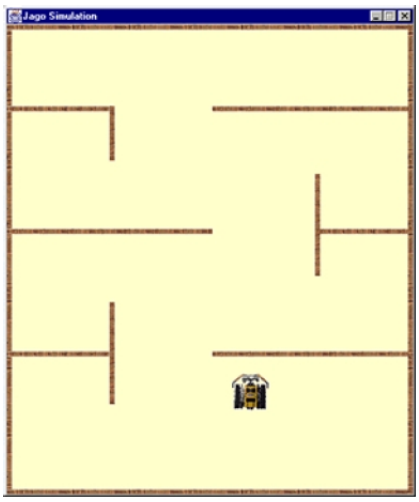


Figura 3.17: Jago, un simulador de Lego Mindstorms.

Fujaba, que también usa este tipo de diagramas para representar las clases.

Algunos micromundos necesitan que el alumno entienda el concepto de la herencia para poder hacer uso de ellos. Por ejemplo, Greenfoot dispone de dos clases principales —`greenfootObject` y `greenfootWorld`— desde las que el alumno ha de crear sus subclases para que éstas puedan ser visualizadas en el entorno. En MUPPETS existe una clase principal que proporciona la capacidad de visualización en el entorno virtual y de la que ha de heredar toda clase creada dentro de este entorno. Los robots creados en los distintos entornos de Karel han de extender una clase que proporciona el comportamiento básico —moverse una casilla, girar a la izquierda, coger y dejar zumbadores, desconectarse o establecer su visibilidad, entre otros.

En el caso de los frameworks los conceptos de la herencia y el polimorfismo son imprescindibles. Una de las características principales de los frameworks es la inversión de control, que implica que es el framework quien lleva el control del flujo de la aplicación y es quien hace llamadas a las clases creadas por el usuario del mismo. Para hacer esto, el usuario de un framework está obligado a especializar ciertas clases del mismo. Por tanto, los alumnos que hacen uso de frameworks de enseñanza necesitan desde el principio hacer uso de la herencia de clases.

Algunos recursos como el simulador de biología marina son especialmente útiles para mostrar los conceptos relacionados con el polimorfismo. En este caso de estudio se proponen ejemplos en los que se puebla el entorno con distintos tipos de peces. Aquí el alumno puede observar que, aunque el método invocado sobre todos los objetos que representan peces es el mismo, el comportamiento de cada uno de ellos es distinto.

3.6.7. Diseño orientado a objetos

La mayoría de los recursos estudiados están más preocupados por la enseñanza de la programación orientada a objetos que por el diseño orientado a objetos. En el caso de los IDEs esta preocupación es aún mayor. La mayoría de ellos solamente proporcionan al alumno la posibilidad de crear y modificar código (ya sea usando un lenguaje de programación concreto o lenguajes visuales), dejando totalmente de lado la descomposición de la aplicación clases.

Algunos IDEs como jGrasp o BlueJ aproximan al alumno ideas de diseño orientado a objetos proporcionando una vista de más alto nivel de la arquitectura de la aplicación que están implementando. Estos entornos usan diagramas UML de clases para mostrar la organización de clases de su aplicación. Sin embargo, estos entornos tan solo muestran una vista general de la arquitectura y no dejan definir las responsabilidades de cada clase a este nivel. En estos entornos, definir una responsabilidad supone implementar los métodos que resuelven dicha responsabilidad.

El único IDE que verdaderamente se preocupa por proporcionar al alumno un alto nivel de abtracción para que pueda preocuparse del diseño de la aplicación es Fujaba. En este IDE el alumno puede desarrollar una aplicación completa sin tener que haber escrito una sola línea de código, tan solo diseñando la arquitectura y definiendo el comportamiento de las clases.

Los micromundos son entornos que dificultan mucho la enseñanza de conceptos de diseño orientado a objetos. Estos entornos vienen con una estructura predeterminada por lo que es muy complicado generar nuevas aplicaciones, incluso en aquellos como Alice y Greenfoot pensados para crear nuevos micromundos. Estos entornos tampoco sirven como ejemplos de buen diseño ya que no proporcionan ninguna información sobre su arquitectura. Solamente Greenfoot da alguna ligera idea sobre su diseño, ya que el alumno ha de conocer de la existencia de las dos clases fundamentales de las que sus clases han de heredar. Pero el resto del diseño del micromundo queda oculto no siendo útil para el propósito descrito en esta sección.

En cambio, los casos de estudio están especialmente pensados para ser usados como modelo de un buen diseño. Por ejemplo, el simulador de biología marina va acompañado de mucho material en el que se detalla el diseño del mismo y en el que se explican algunas de las decisiones tomadas.

Los frameworks también son ejemplos de buenos diseños pero su utilidad en este sentido depende del propósito con el que han sido desarrollados. ObjectDraw es un framework sencillo pero cuya documentación de diseño no es muy extensa. Esto se debe a que su principal objetivo era facilitar la creación de interfaces gráficas, no servir de ejemplo de buen diseño. En cambio, uno de los objetivos de JPT es servir como caso de estudio, por lo que sus desarrolladores han puesto especial interés en detallar su diseño y la documentación que acompaña a las clases.

Los frameworks no son aplicaciones convencionales ya que han sido pensadas para servir como herramienta de reutilización de diseño y código. Esto hace que su arquitectura sea excesivamente compleja, siendo de poca utilidad para los cursos más básicos. Pero en cursos más avanzados son especialmente interesantes, sobre todo en la enseñanza de patrones de diseño. Por ejemplo, la arquitectura de JPT se basa en el patrón MVC (modelo-vista-controlador) y luego hace uso de otros patrones de diseño para resolver problemas de reutilización. Por otro lado, JHotDraw fue diseñado como modelo de aplicación de patrones de diseño. Su diseño está plagado de ellos y gran parte de su documentación se basa en el estudio de su arquitectura desde el punto de vista de los patrones. No es menos cierto que es un framework muy complejo, por lo que muchos de los patrones se encuentran mezclados en el código, lo que en ocasiones dificulta su comprensión.

Como anécdota cabe destacar la utilidad de Lego Mindstorms a la enseñanza de diseño. Wolz (2001) destaca que el elevado coste de los kits de Lego Midstorms (cada kit RCX cuesta aproximadamente unos 200$, elevándose a los 250$ en el caso del kt NXT) hace que los alumnos pongan un especial cuidado en el diseño. El elevado coste hace que generalmente sólo se pueda disponer de un número reducido de robots para un número elevado de alumnos. Esto implica que los alumnos han de compartir los robots y, por tanto, limita el número de pruebas que cada alumno puede realizar sobre el robot, lo que conduce a que el alumno se vuelva más cuidadoso en la planificación, diseño y codificación de los programas.

3.6.8. Patrones de diseño

De acuerdo a otros conceptos más avanzados como son los patrones de diseño, vemos que las herramientas de visualización interactiva no proponen medios específicos para enseñarlos. Esto se puede deber fundamentalmente a que la mayoría de estas herramientas se centran en la enseñanza de la programación, por lo que no están interesadas en mostrar estos conceptos de diseño.

Sin embargo, se puede ver que recursos como los frameworks son especialmente interesantes para su enseñanza. El diseño de frameworks se suele realizar haciendo un uso intensivo de los mismos por lo que son un buen ejemplo para ver su aplicación en sistemas reales. Los desarrolladores de algunos de los frameworks vistos en este estudio, como JHotDraw o JPT, han dado especial valor pedagógico a estas herramientas, realizando un esfuerzo extra para documentar la aparición y el uso de estos patrones de diseño dentro del framework. De esta forma se facilita su uso para ejemplificar la utilización de patrones de diseño.

Aunque JHotDraw ha sido usado en la enseñanza de patrones de diseño (Bærbak-Christensen, 2004), el uso de un framework profesional como éste no es demasiado adecuado para un primer contacto con los patrones de diseño.

Al usar JHotDraw, se está forzando a que el alumno se tenga que enfrentar a una aplicación de gran tamaño (JHotDraw se compone de, aproximadamente, 180 clases) y de gran complejidad. No se puede olvidar que un framework ha sido pensado para la reutilización, por lo que no sigue unas ideas de diseño convencionales —véase, por ejemplo la inversión de control, en la que las clases desarrolladas fuera del framework son llamadas por el framework, en lugar de que dichas clases tengan el control de la ejecución.

Por otro lado, aunque la documentación de este framework ayuda a la localización de los patrones dentro del código y añade comentarios sobre los problemas que motivaron la utilización del patrón aplicado, los patrones siguen sin ser fácilmente identificables. La mayoría de los patrones de diseño que aparecen en JHotDraw se hallan entremezclados, de modo que algunas clases pueden interpretar varios roles, ya sea en distintos patrones o en distintas instanciaciones del mismo patrón. Aunque esto ha de servir para que el alumno comprenda que los patrones pueden convivir juntos dentro de una misma aplicación, complica en gran medida la enseñanza de los mismos.

3.7. Estrategias de interactividad

Como ya hemos señalado con anterioridad, conviene que en el proceso de enseñanza/aprendizaje el alumno no sea un mero observador sino que participe en el desarrollo de actividades relacionadas con el propio proceso.

En este momento vamos a dejar de lado los frameworks y casos de estudio. La interacción con estos recursos se reduce a emplearlos para construir aplicaciones o a modificar su código y observar el nuevo comportamiento de la aplicación. Sin embargo, estamos especialmente interesados en las estrategias seguidas para involucrar al alumno en las que hemos catalogado como herramientas de visualización interactiva. Aunque todas disponen de mecánicas que fomentan la interactividad, el nivel de interacción con el alumno es distinto para cada una de las herramientas.

Se han detectado distintas estrategias de fomento de la interactividad con el alumno. Para hablar de ellas, se ha hecho un estudio de las herramientas de visualización interactiva de acuerdo a la forma de participación propuesta al alumno. Estas formas de participación son las vistas en la Sección 3.1.

3.7.1. Visionado

Como es obvio, todos los entornos presentan al menos el primer grado de interactividad, conocido como visionado. Pero cada entorno implementa unas estrategias distintas de interacción. Todos los entornos presentan al menos dos vistas distintas de los mismos conceptos. Generalmente una de ellas es el código fuente mientras que la otra es la metáfora concreta utilizada para la visualización de la aplicación. Además, se suelen tener distintas vistas para

la estructura estática y para la observación del comportamiento y del estado de los objetos (ya se destacó la función de la inspección de objetos en la Sección 3.6).

Dependiendo del entorno, el número de vistas se puede multiplicar. Por ejemplo, los IDEs son los entornos que mayor número de vistas proporcionan y sobre las que el alumno puede cambiar más. En particular, BlueJ es un entorno en el que las posibilidades de visualización son prácticamente infinitas ya que ha sido desarrollado de modo que desarrolladores externos puedan crear nuevos plugins para este entorno. Como ejemplo cabe destacar que Jeliot puede ser integrado dentro del entorno BlueJ, dando a éste una mayor riqueza en la visualización de las interacciones entre los objetos.

La mayoría de las herramientas, sobre todo las que más posibilidades de visualización proporcionan, permiten que el alumno tenga control sobre las vistas que desea visualizar. Esto permite que el alumno no se sienta saturado por la cantidad de información recibida. Un ejemplo de este tipo de problema es el Jeliot y el alto nivel de detalle de las animaciones que genera. Para este entorno sería interesante añadir granularidad a la información presentada, de modo que se pudiese ocultar o presentar información de manera interactiva

También relacionado con este grado de interactividad está el control de la ejecución de las visualizaciones. Los micromundos presentan interfaces muy intuitivas, similares a las de un reproductor de vídeo, que controlan la ejecución de la simulación. Todos los micromundos permiten congelar y reanudar la ejecución, reiniciarla y finalizarla. También muchos de ellos permiten realizar una ejecución paso a paso para poder observar detenidamente el comportamiento de los objetos. Algunos de ellos permiten incluso modificar la velocidad de la ejecución, pudiendo pasar más rápido determinadas partes de la ejecución que no tienen especial interés.

La mayoría de los IDEs delegan en un depurador para controlar la ejecución de la aplicación. Los depuradores posibilitan realizar ejecuciones paso a paso de una aplicación o añadir puntos de ruptura en lugares de especial interés. A través del depurador se suele poder observar el estado de los objetos, de modo que se puedan apreciar los cambios de estado de los mismos. También suelen incluir información sobre la pila de llamadas, lo que es de especial interés para observar la delegación entre objetos. Tan solo Jeliot proporciona una interfaz similar a la descrita en los micromundos para controlar la ejecución del programa. Este entorno incluye además un historial de la ejecución, de modo que se puede ir directamente a puntos concretos de la visualización.

3.7.2. Respuesta

Ninguna de las herramientas estudiadas integra el grado de interactividad relacionado con la realización de preguntas sobre las visualizaciones

presentadas. Pero muchos de los desarrolladores de estos entornos saben de la importancia de que el alumno comprenda lo que está viendo y lo que ha de hacer con el entorno, por lo que incluyen documentación, ejercicios propuestos y libros de apoyo a la enseñanza de la orientación a objetos usando su entorno. BlueJ (Barnes y Kölling, 2005), Karel the Robot (Pattis, 1995; Bergin et al., 2005; Becker, 2006), Alice (Dann et al., 2005) u ObjectDraw (Bruce et al., 2005) son ejemplos de entornos que disponen de guías de enseñanza de orientación a objetos. En ellas se detallan todo tipo de ejercicios que el alumno ha de resolver usando estas herramientas.

3.7.3. Cambio y construcción

Estos dos niveles de interactividad son los más usados por todos los entornos. Además, se han tomado como uno solo ya que en muchos de los entornos la línea que separa ambos tipos de interacción es muy fina.

La mayoría de los entornos hacen que el alumno cambie o cree nuevas visualizaciones mediante la implementación de código fuente. Ya que éste es uno de los principales medios de interacción, muchos de los entornos, además de los IDEs, integran sus propios editores de código, ya sean textuales (como en Jeroo, la mayoría de las versiones de Karel o Greenfoot) o visuales (como en Alice o en algunas versiones de Karel (Bergin, 2006)).

Dentro de este tipo de interacción se incluye también los inspectores de objetos interactivos de algunos IDEs. Estos inspectores no se limitan a mostrar la información del estado sino que además permiten modificar dicho estado y ejecutar métodos del objeto. Esta interacción permite hacer cambios sobre el objeto para que el alumno observe que el comportamiento del mismo puede depender del estado actual del objeto. Otros entornos como JPie incluso permiten modificar las clases durante la ejecución, de modo que los objetos ya creados de esa clase son capaces de reconfigurarse para comportarse de acuerdo a las modificaciones realizadas.

Todos los micromundos permiten cambiar los escenarios donde se va a ejecutar la simulación y crear el comportamiento de algunos de los objetos que aparecerán en el mundo. Algunos de ellos como Alice y Greenfoot permiten incluso crear nuevos mundos y nuevas clases de objetos que lo poblarán. Como ya se ha visto con anterioridad, la forma de crear objetos y mundos es muy variada, haciendo uso de lenguajes de programación textuales o visuales o mediante un simple "arrastrar y soltar" de entidades sobre el mundo.

3.7.4. Presentación

Cualquiera de los entornos presentados es susceptible de ser usado como herramienta para que instructor y alumnos creen visualizaciones y discutan sobre conceptos relacionados con la orientación a objetos. Sin embargo, ninguna de las herramientas da especial interés a este uso. Pero hay que

destacar iniciativas como MUPPETS, en las que el propio entorno es el medio en el que los implicados comparten y discuten los diseños y clases creados. Este IDE dispone de servidores para el almacenamiento de código fuente de distintos alumnos y servidores de compilación, responsables de la creación y almacenamiento de objetos precompilados. De este modo, los alumnos pueden recuperar tanto su trabajo como el de otros alumnos. Además, las herramientas de comunicación proporcionan el medio para que los alumnos puedan discutir sobre las decisiones tomadas y los cambios realizados.

3.8. Conclusiones

A lo largo de este capítulo se ha descrito el estudio realizado sobre los recursos de apoyo a la enseñanza de la orientación a objetos. Como se ha podido ver, gran parte de ellos se basan en el uso de la visualización como medio fundamental para presentar al alumno los conceptos existentes en el paradigma orientado a objetos. Las metáforas empleadas para estos conceptos son muy variadas, desde el uso de representaciones formales como los diagramas de UML hasta robots físicos, pasando por todo tipo de entidades con estado y comportamiento observables que habitan mundos en dos y tres dimensiones.

Los frameworks y los casos de estudio se han mostrado como un recurso muy valioso de cara al estudio de aplicaciones bien diseñadas. Con ellos el alumno se empapa de las experiencias de diseño de los desarrolladores de estos recursos. También se ha visto que muchos de ellos son muy interesantes para la enseñanza de patrones de diseño. Sin embargo, la interactividad del alumno con estos recursos se ve reducida a tareas de desarrollo y modificación de aplicaciones haciendo uso de estos recursos y observación de la ejecución de las mismas.

De entre todos los medios de apoyo a la enseñanza estudiados nos hemos interesado especialmente por las herramientas de visualización interactivas, que fomentan la participación del alumno involucrándolo activamente en su proceso de aprendizaje. Las mecánicas por las que se promueve la participación del alumno son muy variadas y todos los entornos suelen tener más de una. Además, estas mecánicas promueven distintas formas de participación, desde el control de la información y de la ejecución de la visualización hasta la creación y modificación de las propias visualizaciones.

De acuerdo a las formas de interacción descritas en la Sección 3.1, una de las menos explotadas es la relacionada con la integración de preguntas que el alumno ha de resolver haciendo uso de la información proporcionada por la visualización. El único soporte que se suele dar a este tipo de interacción es mediante la elaboración de actividades, ejercicios y prácticas que el alumno ha de resolver haciendo uso de las herramientas, o mediante cuestionarios que han de ser resueltos observando una simulación. Sería interesante estudiar

cómo integrar parte de estas cuestiones dentro de la propia actividad con la herramienta, de modo que el alumno pudiera ser evaluado *in situ*. La evaluación de las cuestiones informa al alumno de los errores que comete, lo que proporciona oportunidades especialmente interesantes de aprendizaje, ya que en este momento el alumno se hace consciente de que tiene carencias de conocimiento y está preparado para aprender.

Tampoco está muy explotada la participación mediante presentación, en el que el alumno expone sus conocimientos y genera discusión con otros alumnos haciendo uso de la herramienta. En esta línea es especialmente interesante el enfoque del entorno MUPPETS, que fomenta el trabajo en grupo de los alumnos para aprender orientación a objetos. Los entornos de aprendizaje colaborativo por computador (CSCL, del inglés *Computer Supported Collaborative Learning*) como MUPPETS son herramientas que, aunque han sido estudiadas desde hace tiempo, están adquiriendo especial interés en los últimos años. Estas herramientas permiten que los alumnos trabajen en la realización de una tarea sin tener que estar físicamente en el mismo lugar. Estas colaboraciones no sólo son interesantes por la actividad desarrollada sino que también promueven la reflexión del alumno, ya que el trabajo en grupo hace necesario que los alumnos sean capaces de comunicar sus ideas, y argumentar o evaluar las opiniones de otros alumnos (Guzdial et al., 1996).

Como se ha podido constatar, hay una predominancia de las herramientas que enseñan programación orientada a objetos. En cambio, la gran mayoría de las herramientas presentadas dan poco o nulo soporte a la enseñanza del diseño orientado a objetos. El diseño es tanto o más importante que la implementación en este paradigma pero no es fácil enseñar diseño estando tan ligado a un lenguaje de programación. El diseño supone abstracción mientras que la programación obliga a ligarse a las restricciones que impone un lenguaje. Esto no significa que haya que dejar de lado la implementación sino que ésta podría tener una posición menos predominante en la enseñanza de la orientación a objetos. Para poder mejorar estas herramientas de ayuda necesitamos estudiar la forma en la que el diseño orientado a objetos es enseñado en las clases presenciales. De aquí se pueden extraer estrategias, técnicas e ideas que puedan ser trasladadas a herramientas de visualización y que ayuden a aliviar las carencias descritas a lo largo de este capítulo.

Capítulo 4

Uso de técnicas de aprendizaje activo para la enseñanza de la orientación a objetos

Las cosas no son porque existan, son porque se
sienten, porque alguien las retiene,
las recuerda, les da vida.

Princesas (2005)

La gran mayoría del conocimiento que una persona adquiere lo hace a través de lo que oye y de lo que ve. Pero existen gran cantidad de conceptos lo suficientemente complejos como para necesitar algo más que una imagen o un discurso para ser comprendidos. Por ejemplo, habilidades sociales como el liderazgo o la diplomacia no pueden ser aprendidas tan solo con una serie de clases teóricas que proporcionen los conceptos básicos que hay tras ellas. La asimilación de estos conceptos tan abstractos pasa por la puesta en práctica de los mismos, para que el alumno se involucre activamente en su propio aprendizaje. Las técnicas que promueven este tipo de aprendizaje se conocen como técnicas de aprendizaje activo.

La enseñanza de la orientación a objetos concierne determinados conceptos abstractos de difícil comprensión por parte de los alumnos. Como se ha concluido en el capítulo anterior, la visualización de la estructura y del comportamiento de una aplicación software es una tarea pasiva, de efectividad limitada si no se ve reforzada por la intervención del alumno. Es conveniente, pues, que el alumno se vea involucrado en las tareas de aprendizaje para garantizar el máximo aprovechamiento de las mismas. Estas tareas le han de proporcionar la experiencia que le permita ir adquiriendo mayores destrezas y habilidades en el diseño y desarrollo de aplicaciones en este tipo de paradigma.

Este capítulo hace un estudio de la aplicación de técnicas de aprendizaje activo en la enseñanza del paradigma de la orientación a objetos. La Sección 4.1 hace un somero estudio de lo que estas técnicas, en abstracto, proporcionan al alumno. Posteriormente se describe una técnica de aprendizaje activo empleada en el diseño orientado a objetos: el role-play.

En la Sección 4.2 se hace una revisión de la literatura en busca de la aplicación de estas técnicas en la enseñanza de la orientación a objetos. Se verá que los trabajos revisados hacen uso de un tipo concreto de role-play muy dirigido que, aunque efectivo, no deja margen a la exploración del alumno, algo que también le favorecería de cara a su aprendizaje.

Tomando en cuenta el uso que han dado otros autores a estas técnicas para la enseñanza de diversos conceptos relacionados con la orientación a objetos, la Sección 4.3 describe la propuesta del autor para la enseñanza de conceptos más avanzados, como son los patrones de diseño. Inicialmente se realiza una descripción en abstracto de la metodología propuesta. Ésta se basa en el diseño iterativo de una aplicación sencilla y completa, partiendo de un diseño propuesto por el instructor, en el que se combinan episodios de refactorización con el role-play para la evaluación del diseño.

La propuesta descrita en este capítulo no sólo ha quedado en el ámbito más teórico y de manera abstracta sino que también ha sido puesta en práctica con éxito en la impartición de dos seminarios sobre patrones de diseño en la Facultad de Informática de la Universidad Complutense de Madrid. En la Sección 4.4 se detalla cómo se hizo efectiva la propuesta descrita en la anterior sección. Se define el caso de estudio concreto que se ha empleado en estos seminarios y se hace un breve resumen de las sesiones desarrolladas y de los objetivos pedagógicos que han perseguido cada una de las sesiones.

Para terminar, se incluyen los resultados de la evaluación que se ha realizado a lo largo de los dos cursos impartidos. Por un lado, se ha evaluado el grado de aceptación obtenido por parte de los alumnos ante una técnica totalmente novedosa para ellos y que les hace tan partícipes en su propio aprendizaje. Por otro lado, se han desarrollado una serie de tests que han sido empleados para medir la efectividad de la propuesta de enseñanza realizada. Se ha hecho un estudio sobre las diferencias obtenidas entre alumnos que han seguido una metodología tradicional para la enseñanza de patrones de diseño frente a aquéllos que han participado activamente en las sesiones en las que se ha hecho uso de las técnicas de aprendizaje activo. Así mismo, se han comparado los resultados obtenidos entre los participantes activos y los espectadores de estas sesiones, para así probar las capacidades del aprendizaje activo para mejorar la efectividad del proceso de enseñanza.

4.1. Técnicas de aprendizaje activo

La fórmula tradicional de enseñanza en el ámbito universitario ha sido la clase magistral. En ella, el instructor es el centro de atención de la clase y en la que su función es transmitir conocimiento mientras que los alumnos actúan como simples receptores de esa información. En la literatura se ha demostrado que el uso exclusivo de este tipo de clases no es demasiado efectivo ya que los alumnos tan solo actúan como receptores pasivos de información. Esto se debe a que la mayoría de las personas aprenden sólo el 50 % de lo que ve y escucha, mientras que aprenden un 80 % de lo que usa en la vida real (Warren, 2005).

Las técnicas en las que el alumno pasivo es transformado en parte activa del aprendizaje se conocen con el nombre de técnicas de aprendizaje activo. Estas técnicas incluyen el uso de preguntas abiertas a los alumnos, la lectura, la escritura, la resolución de problemas o que los propios alumnos sean los encargados de dar una clase, entre otras. Las técnicas de aprendizaje activo ponen el énfasis en el desarrollo de las habilidades del alumno, sobre todo en las de más alto nivel, como son el análisis, la síntesis y la evaluación. Además, incrementan la motivación del alumno (Schank y Cleary, 1995).

La enseñanza de Informática, en general, y de la orientación a objetos, en particular, es una disciplina en la que el aprendizaje activo encaja perfectamente. Muchas de las asignaturas que se cursan incluyen prácticas de diseño y de programación en parejas, así como la realización de pequeños ejercicios en los que el alumno pone en práctica los conceptos aprendidos durante las clases magistrales.

En el caso de la orientación a objetos existen una técnica que ha sido usada en tareas de diseño orientado a objetos y que posteriormente ha sido trasladada al ámbito de la enseñanza: la interpretación de roles o role-play. A continuación pasamos a describir esta técnica.

4.1.1. Role-play

La interpretación de roles o role-play no es una técnica propia de la orientación a objetos sino que es una estrategia de psicología interpersonal empleada para la resolución de conflictos. En el ámbito de la enseñanza, es una evolución del método socrático que se basa en el examen y el razonamiento de conceptos mediante un diálogo de preguntas y respuestas. En éste, el alumno aprende por descubrimiento mediante el uso de preguntas. En el role-play, el alumno también aprende por descubrimiento pero mediante la interpretación de un determinado rol o papel en una situación hipotética. De esta forma, el alumno comprende cómo se puede llegar a sentir y cómo se comportará en dichas situaciones. Se basa en que el alumno aprende activamente metiéndose en ese papel, lo que es mucho más efectivo que el aprendizaje pasivo en el que el alumno recibe la información de terceras

personas.

En el campo que nos concierne, la orientación a objetos, el role-play nació como una técnica para el diseño orientado a objetos (Wirfs-Brock y McKean, 2003; Bellin y Suchman-Simone, 1997). Posteriormente, se ha trasladado al ámbito de la enseñanza (Biddle et al., 2001; Börstler, 2005) y proporciona una técnica de aprendizaje activo muy útil para la comprensión, análisis y la evaluación de un sistema. En lugar de que los alumnos reciban de manera pasiva un análisis sobre el diseño de un determinado sistema, el role-play pone a los alumnos en la piel de los objetos para que piensen y analicen el sistema completo. Mediante la interacción de los alumnos durante la simulación de la ejecución de un determinado escenario de uso, el alumno es capaz de comprender cómo se organiza el sistema, dónde se concentra determinado comportamiento del mismo y cómo colaboran las clases para llevar a cabo dicho comportamiento. Además, una sesión de role-play es mucho más motivante y entretenida que escuchar una descripción sobre el diseño de una aplicación, aunque esté acompañada de información visual como diagramas UML de clases y de colaboración.

El role-play es especialmente idóneo en la orientación a objetos debido a que se puede establecer una metáfora muy clara entre un objeto y una persona: un objeto es una entidad autónoma, con unos atributos propios y que no suele ser muy útil de manera aislada, por lo que necesita de la interacción con otros objetos para llevar a cabo tareas (Bergin et al., 2001). Ésta hace que sea sencillo que el alumno se meta en la piel de un objeto.

Además, el role-play es útil para evitar falsas expectativas de diseño. En ocasiones, un nombre de clase confuso puede hacer que un alumno suponga erróneamente cuáles son las responsabilidades de una determinada clase. En cambio, cuando esta persona observa un objeto de esa clase en acción (incluso él mismo puede interpretar el papel de este objeto) se da cuenta de cuáles son las responsabilidades reales, ya que estas falsas expectativas emergen durante la simulación.

El role-play suele emplear las tarjetas CRC como herramienta para representar la información de las clases del diseño a evaluar. Éstas se describen a continuación.

4.1.1.1. Tarjetas CRC

Las tarjetas CRC nacieron como una herramienta empleada en el diseño dirigido por responsabilidades –del inglés *Responsibility-driven design* (Wirfs-Brock y McKean, 2003). Las tarjetas CRC se usan como ayuda en la tarea de definición de las clases básicas que componen un sistema software, qué es lo que cada clase hace y cómo colaboran entre sí. También son empleadas para la evaluación de software ya desarrollado.

Su nombre proviene de las palabras **C**lase (según Beck y Cunningham

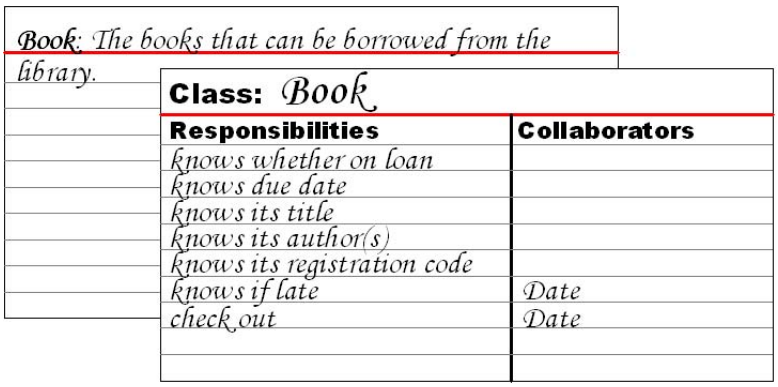


Figura 4.1: Ejemplo de una tarjeta CRC (extraída de Börstler (2005)).

(1989), o **C**andidato según Wirfs-Brock y McKean (2003)), **R**esponsabilidades y **C**olaboradores. Una tarjeta CRC representa a una clase de una aplicación software orientada a objetos y en ella se detalla qué es lo que esta clase sabe hacer, qué información conoce y con qué clases colabora para completar una tarea. Las tarjetas CRC se usan tradicionalmente en papel, con un formato o estructura similares a las de la Figura 4.1:

- El nombre de la clase queda descrito en la parte superior de la tarjeta. Generalmente es un sustantivo y ha de ser único y lo suficientemente explicativo como para que cualquiera pueda entender la abstracción que representa. La parte trasera de la tarjeta también se usa para guardar una breve descripción de la clase o cualquier clase de comentario que se quiera incluir al respecto.

- Una responsabilidad define qué es lo que un objeto de la clase definida por la tarjeta sabe hacer o qué tipo de información guarda. A nivel de tarjetas CRC, lo importante es que una responsabilidad defina lo que la clase sabe hacer pero que no se preocupe en cómo lo ha de hacer (su implementación). También es importante destacar que las responsabilidades son las que permiten a una clase delegar en otras para completar una tarea. A nivel de representación las responsabilidades aparecen en la parte izquierda de la tarjeta CRC.

- Un colaborador es un objeto de otra clase que ayuda a la consecución de una responsabilidad. Un objeto puede delegar la ejecución de una parte de alguna de sus responsabilidades en uno o varios de sus colaboradores. En la parte derecha de la tarjeta CRC aparecen las clases de los objetos que pueden participar como colaboradores. Generalmente, los colaboradores quedan alineados con la responsabilidad en la que intervienen.

En el ámbito de la enseñanza las tarjetas CRC se emplean para el desarrollo, discusión y evaluación de un sistema orientado a objetos (Börstler, 2005). Durante una primera fase, los alumnos son los responsables de definir los candidatos potenciales a ser clases para un sistema propuesto. Haciendo uso de las tarjetas CRC decidirán el nombre de estas clases, cuáles son las responsabilidades de cada clase y cuáles creen que serán las colaboraciones que se van a producir entre ellas. Durante esta fase, las clases son tratadas de manera aislada y pueden ser desarrolladas por distintos alumnos. Posteriormente, los alumnos harán uso de estas tarjetas CRC para evaluar el diseño que han alcanzado y para refinarlo en caso de que sea necesario. Esta segunda fase de diseño se realiza usando todas las tarjetas CRC y el role-play.

4.1.2. Sesiones de role-play

Las sesiones de role-play con tarjetas CRC son un proceso iterativo que se compone de cinco fases (Bellin y Suchman-Simone, 1997):

1. Desarrollo de una lista de escenarios. Para verificar que el diseño de clases desarrollado es correcto, es necesario comprobar que el sistema se comporta correctamente durante la ejecución haciendo uso de los objetos de las clases creadas. Por tanto, es necesario elaborar una lista con los escenarios típicos de ejecución del sistema, así como escenarios que representan situaciones excepcionales. Estos escenarios también se conocen como casos de uso del sistema y definen el guión de ejecución de la simulación.

2. Asignación de roles. Antes de la simulación de cada uno de los escenarios es necesario decidir qué alumno será el responsable de la interpretación de cada uno de los roles participantes. En este momento es cuando a cada alumno se le proporciona la tarjeta CRC de la clase del objeto que va a representar durante la simulación.

3. Simulación del escenario. Haciendo uso del guión del escenario, los alumnos interactuan entre ellos de acuerdo a las responsabilidades y colaboraciones descritas en las tarjetas CRC. Cada escenario es iniciado por un determinado actor, encargado de realizar la primera petición a uno de los objetos participantes. Cada uno de los alumnos puede hacer preguntas al resto de los objetos para comprender cuáles son las interacciones que puede realizar. Cada vez que un alumno realiza una acción relativa a la simulación ha de anunciarla al resto de los participantes.

4. Corrección de las clases. Ya que las clases han sido desarrolladas de manera aislada no siempre suelen representar la mejor solución. Generalmente, las simulaciones hacen que alguno de los participantes descubra

errores o mejoras en el diseño inicial, por lo que es necesario revisar las clases iniciales y con ellas, las tarjetas CRC iniciales. Esta revisión puede implicar varios tipos de acciones, desde la redistribución de las responsabilidades hasta la creación de nuevos colaboradores. Una vez que las correcciones han sido hechas es necesario revisar el escenario que se está simulando. Una pequeña revisión puede permitirnos continuar con la simulación desde el punto en el que el fallo fue detectado. Sin embargo, una revisión mayor puede obligarnos a comenzar de nuevo la simulación.

5. Simulación y revisión del escenario final. Una vez que los escenarios típicos han sido ejecutados sin necesidad de hacer modificaciones en las tarjetas CRC, se pasará a comprobar que el diseño también soporta aquellos escenarios atípicos y que suelen ser excepciones al comportamiento habitual del sistema. Este proceso también puede conducir a la realización de nuevas modificaciones en las tarjetas CRC de cara a que el diseño también soporte estos escenarios.

En la simulación de los escenarios suele ser útil hacer uso de algún elemento que sirva para representar al objeto activo. Sólo éste está capacitado para dar el siguiente paso de ejecución, interactuando con otro objeto y haciendo que éste pase a ser el nuevo objeto activo. Generalmente, los participantes suelen ir pasándose una pelota o algún otro objeto que sea el que marque a dicho objeto como el activo.

Cuando el role-play se aplica a la enseñanza de la orientación a objetos se rige por las reglas que impone el paradigma. Además de que el único objeto responsable de dar el siguiente paso de ejecución será el que actualmente se encuentre activo, también hay que tener en cuenta que un objeto sólo responde a las responsabilidades que describe su tarjeta CRC y que un objeto sólo puede pasar un mensaje a otro si la clase del segundo es colaboradora de la clase del primero y si existe una vinculación entre ambos objetos. Además, será necesario recordar las modificaciones de los atributos de un objeto para actuar de acuerdo a los mismos en el presente escenario.

Por último, cabe destacar que el role-play también aumenta su efecto beneficioso en el aprendizaje del alumno debido a su aspecto colaborativo. El alumno no sólo aprende de sí mismo sino que la simulación del escenario hace que la tarea de análisis quede distribuida entre los participantes de la simulación. De este modo, cada alumno se beneficia de los conocimientos y de las opiniones del resto de los participantes.

4.1.3. Análisis del role-play

Las tarjetas CRC, conjuntamente con el role-play, han sido ampliamente utilizadas y analizadas en la literatura, tanto en el ámbito de la enseñan-

za como fuera de este contexto. A continuación detallamos las principales ventajas y problemas que se han extraído de estos análisis.

Algunos autores destacan la importancia de estas herramientas en la enseñanza de la orientación a objetos (Börstler y Schulte, 2005). Esta disciplina no consiste en aprender cómo es la sintaxis de un lenguaje orientado a objetos, como Java o C++, sino en interiorizar un modelo conceptual que permita razonar sobre un sistema orientado a objetos sin ligarse a un lenguaje de programación concreto. Las tarjetas CRC y el role-play emplean los conceptos básicos de la programación orientada a objetos (clases, objetos y paso de mensajes) independientemente del lenguaje usado.

Otros autores (Biddle et al., 2001) también destacan que las tarjetas CRC facilitan la discusión tanto de la estructura estática de clases como de la relación entre objetos y del comportamiento dinámico de un sistema software. Destacan principalmente la importancia del role-play como herramienta de exploración del software. Los participantes se meten en el papel del objeto y estudian detenidamente el comportamiento y las interacciones de los objetos a base de cuestionarse qué es lo que ocurre si acontecen unas determinadas circunstancias en el sistema. Esto ayuda no sólo a su entendimiento sino también a la consecución de mejoras en el sistema.

Durante el estudio de estos trabajos también nos hemos hecho eco de algunos de los problemas (y de las soluciones que los autores proponen) a los que se enfrenta aquél que usa estas técnicas en la enseñanza de orientación a objetos. Uno de las más destacadas es la dificultad de los alumnos en distinguir entre las clases y los objetos (Biddle et al., 2001; Börstler y Schulte, 2005; Beck y Cunningham, 1989). Es importante que los alumnos se den cuenta de que una tarjeta CRC representa una clase, mientras que en el role-play participan instancias de las clases definidas por las tarjetas CRC, esto es, objetos. La solución más comúnmente usada y que coincide con las recomendaciones dadas para la enseñanza de los conceptos básicos de orientación a objetos es que se usen escenarios en los que participen varias instancias de la misma clase, de modo que varios de los participantes deberán compartir la misma tarjeta CRC.

Otro de los principales problemas encontrados es el nivel de abstracción en el que los alumnos han de trabajar. El role-play se usa para analizar las responsabilidades a nivel de diseño, pero no se suele emplear para verificar si la implementación es correcta. El problema es bastante comprometido ya que los alumnos suelen cambiar de un nivel a otro casi sin darse cuenta. Como se detalla en (Börstler y Schulte, 2005), suele ser más fácil trabajar con aquellos alumnos que no tienen conocimientos de programación, ya que "conceptualizan los objetos como 'seres inteligentes', por lo que asumen que no es necesario especificar explícitamente ciertos detalles".

Otros autores destacan la confusión del término "colaborador" (Biddle et al., 2001). Algunos alumnos no se dan cuenta que este término no es

bidireccional (si una clase A es colaboradora de la clase B, esta última no tiene por qué ser colaboradora de A). Estos mismos autores prefieren hacer uso de la palabra "ayudante" (*helper*), que elimina esta confusión en los alumnos.

También se apunta la facilidad con la que es posible perderse durante las simulaciones de los escenarios. Este problema se ve acrecentado sobre todo cuando el número de objetos que intervienen en el escenario es grande y cuando cada objeto dispone de un gran número de responsabilidades. A nivel de enseñanza se propone que los sistemas empleados sean lo suficientemente complejos como para que se pueda hacer palpable la diferencia entre clases y objetos y de la necesidad de interacción entre objetos, pero lo suficientemente simples como para no saturar al alumno (Börstler, 2005). Además, se ha de intentar que en los escenarios empleados no intervengan un número excesivo de objetos.

También destacan la importancia de "grabar" dichas interacciones para que los escenarios puedan ser analizados fuera de la simulación. Mientras que algunos optan por documentar las sesiones de role-play mediante su grabación en vídeo (Biddle et al., 2001), otros optan por medios más elementales como crear diagramas UML de colaboración (Wirfs-Brock y McKean, 2003). De entre los dos posibles tipos de diagramas de colaboración (interacción y secuencia) los autores no se decantan por ninguno aunque destacan que, mientras que los diagramas de interacción ponen el énfasis en las relaciones existentes entre los objetos, los diagramas de secuencia lo ponen en la secuencia de los pasos de mensajes.

Especialmente interesante resulta la propuesta que se realiza en (Börstler, 2005). Aquí se hace uso de lo que se definen como diagramas de role-play o RPDs (del inglés *Role-play Diagrams*). Estos diagramas incluyen elementos de los diagramas de objetos y de colaboración. El autor destaca que los diagramas de secuencia no tienen información sobre las relaciones entre los objetos. Además, aunque los diagramas de interacción sí disponen de esa información, ninguno de los dos captura información sobre el estado de los objetos, algo que se considera necesario para la comprensión del escenario.

Los objetos se representan en un RPD mediante tarjetas que guardan el nombre del objeto, la clase a la que pertenece y el estado en el que se encuentra. Dos objetos se pueden comunicar sólo si están conectados por una línea, ya que significa que ambos objetos "se conocen". Además, un objeto sólo puede conectarse a las instancias de sus clases colaboradoras. El paso de mensajes entre objetos se representa mediante una flecha y la responsabilidad que ha sido invocada. Para facilitar el seguimiento de este paso de mensajes estas comunicaciones van numeradas. Cuando una responsabilidad ha de devolver un valor, se añade una flecha con un círculo, acompañada del valor retornado. Por último, los cambios de estado de un objeto se representan explícitamente, tachando el valor antiguo de la propiedad cambiada

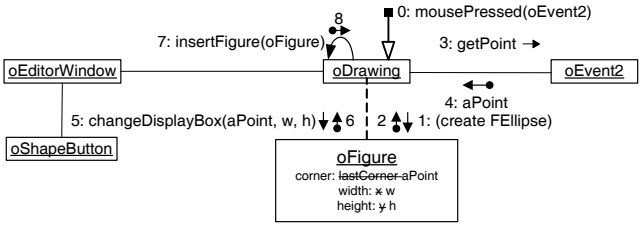


Figura 4.2: Ejemplo de un diagrama de role-play (RPD).

y añadiendo el nuevo valor. Si durante el role-play se crea una "unión" entre dos objetos —un objeto crea a otro objeto o recibe como resultado de una invocación una referencia a otro objeto— entonces se dibuja una línea punteada entre ambos objetos.

Un ejemplo de RPD se puede ver en la Figura 4.2. Representa un escenario de role-play en el que intervienen cinco objetos. El mensaje inicial es el marcado por un extremo cuadrado y va acompañado del valor 0. El siguiente mensaje es el de creación del objeto *oFigure*, lo que crea la línea punteada entre éste y *oDrawing*. Siguiendo la numeración, tras el retorno de la construcción del objeto el siguiente mensaje va dirigido al objeto *oEvent2*. El retorno de este mensaje, representado por la flecha con extremo redondo, devuelve como valor *aPoint*. El paso del siguiente mensaje produce el cambio de los atributos de *oFigure*. Este cambio se representa tachando el valor antiguo y sustituyéndolo por el nuevo valor. Por último, *oDrawing* se autoinvoca un método, representado por una flecha hacia sí mismo, y se realiza el retorno de dicho mensaje.

Desgraciadamente, una de las principales faltas que tiene esta técnica es que no define una metáfora ni contemplan un concepto fundamental de la orientación a objetos: la herencia. Revisando la literatura relacionada con esta técnica no se ha encontrado ninguna referencia sobre cómo solventa la simulación de sistemas en los que algunas de sus clases se encuentren relacionadas por herencia ni de cómo hacen uso del polimorfismo. Es posible que esto se deba a que el role-play fue creado con el propósito de ayudar en la tarea de análisis. Esta tarea no contempla las relaciones entre clases sino entre objetos, dejando para la tarea de diseño la definición de las relaciones de herencia entre clases.

4.2. Role-play en la enseñanza de la orientación a objetos

Desde hace más de una década la enseñanza de la orientación a objetos ha sido un tema de extenso debate en la comunidad docente informática (ACM, 2001; ACM, 2005). A lo largo de todos estos años no sólo se ha

debatido cómo enseñar de manera más efectiva los conceptos concernientes a la orientación a objetos sino también cuál es el mejor momento en el que enseñar esta disciplina. Tras todo este tiempo no ha habido ningún tipo de consenso sobre el cuándo ni el cómo. Como ya se pudo ver en el Capítulo 2, ni siquiera hay un consenso real en lo que se considera como los conceptos básicos. En la literatura se han podido ver una gran variedad de alternativas, unas que tan sólo han quedado en propuestas, otras que han sido puestas en práctica con buenos resultados. En lo que hay unanimidad es en la necesidad de que el alumno vea y experimente con ejemplos.

Los casos de estudio fomentan la actividad del alumno para la comprensión de los conceptos de orientación a objetos, como se destacó en la Sección 2.4.2. Pero existen propuestas que van un paso más allá y que combinan las experiencias de estos entornos con las técnicas de aprendizaje activo descritas en la Sección 4.1.

Existen una serie de propuestas concretas para el uso del role-play en la enseñanza de orientación a objetos. En (Bergin, 2006) se describe una propuesta que hace uso del role-play para que los alumnos vean qué es un objeto y cómo los objetos interaccionan entre sí para llevar a cabo una responsabilidad. Más interesante es el trabajo de (Andrianoff y Levine, 2002), en el que se describe con un poco más de detalle su uso a lo largo de distintas sesiones. Este trabajo describe tres sesiones de role-play, a lo largo de las cuales se introducirán los siguientes conceptos:

- Encapsulación y ocultación de información. Se destaca el hecho de que los objetos son unidades autónomas con estado propio.

- Protocolo de paso de mensajes. Se describe la manera en la que se hace el paso de mensaje de un objeto a otro, destacando el paso de parámetros, los valores devueltos y la sobrecarga de métodos. También se destaca el hecho de que un objeto puede invocar uno de sus propios métodos o que puede pasarse a sí mismo como parámetro.

- Clases y objetos. Se hace patente la diferencia que hay entre unos y otros. Relacionado con esto también se presentan distintas instancias (objetos) de la misma clase.

- Delegación. Se observa cómo, mediante el paso de mensajes, unos objetos delegan en otros para llevar a cabo una tarea. También esto ayuda a los alumnos a que vean que existe un flujo de control, en el que el control de la ejecución y la responsabilidad de decidir cuál es el siguiente paso de ejecución pasa de un objeto a otro.

- Herencia y polimorfismo. Se hace patente el comportamiento polimórfico de los objetos. También se muestra cómo la invocación de un método puede recaer en parte o totalmente sobre sus ascendientes y cómo la

construcción de un objeto recae en parte en la construcción de las clases ascendientes.

En este trabajo, las sesiones de role-play se encuentran totalmente guiadas (los autores lo definen como *scripted role playing*). Para cada rol se detalla exactamente qué es lo que ha de hacer y, en ocasiones, cuándo lo ha de hacer. Además, estos guiones detallan de manera informal el comportamiento, haciendo mucho énfasis en la acción que se ha de realizar durante el role-play –por ejemplo, los guiones muestran frases como "pide a Vecino que haga Select, pasando el entero que recibiste en el paso 4 (como respuesta recibirás una posición)". Estos mismos autores han desarrollado también este tipo de guiones para los casos de estudio del College Board, tanto para el simulador de biología marina como para GridWorld[1]. Como destaca Andrianoff, el uso del role-play hace que los alumnos puedan pensar más a nivel de diseño que a nivel de código, aunque los guiones sigan ligándose en gran medida al mismo.

Aunque estos trabajos son muy ilustrativos y han servido para sentar las bases de la investigación descrita en esta Tesis, no por eso se encuentran exentos de crítica. La principal es que este tipo de role-play no deja lugar a la iniciativa del alumno. Los guiones hacen que los alumnos se involucren en la ejecución de los escenarios pero tratándolos como meras marionetas. No dan opción alguna al error del alumno (salvo los que estén descritos a propósito en el guión) ni a que tenga que revisar la documentación que le ha sido aportada para deducir cuál es el siguiente paso de role-play que ha de dar. Es cierto, como destaca Andrianoff, que en una aplicación concreta la secuencia de acciones que se ha de realizar está perfectamente definida, pero no por ello hay que especificarlo tal cual para que el alumno lo simule. Este tipo de guiones no da opción a la proactividad del alumno. Es mucho más rico para el alumno, y tal vez menos frustrante, darle la oportunidad de decidir cuál es el siguiente paso de ejecución a dar. Esta libertad también permite que el alumno pueda ver que, dependiendo del ejemplo concreto que se esté realizando, el orden de ejecución de determinadas acciones puede ser o no relevante. Además, es mucho más pedagógico dejar que se equivoque, aunque posteriormente sea reconducido al camino correcto. De esta forma, el alumno se sentirá mucho más involucrado en su tarea de aprendizaje.

Por otro lado, en estos trabajos tampoco se ha dado una solución al problema de cómo representar la herencia y el polimorfismo en un escenario de role-play. Según estos trabajos, cuando se crea una nueva instancia de una subclase, junto a ésta aparece un objeto "genérico" de la superclase que le ayuda a la ejecución de determinados métodos, así como a almacenar determinada información. Esta metáfora parece algo confusa ya que da la sensación que el objeto se compone de otros objetos sobre los que delega, que no es el caso, o que la creación de un objeto de la subclase supone la creación

[1]Estos guiones están disponibles en la página sobre role-play de Kevin D. Levine: `http://www.cs.sbu.edu/dlevine/RolePlay/roleplay.html` (último acceso: 15-02-2008)

de un objeto de la superclase, lo que puede conducir a la confusión, sobre todo a la hora de la utilización de clases abstractas (que no pueden ser instanciadas aunque pueden contener información del estado y responsabilidades comunes a todas sus subclases).

Aunque en los trabajos anteriores se dejan a un lado los conceptos de diseño, cabe destacar el trabajo de Jutta Eckstein en el uso de role-play para la comprensión de arquitecturas complejas (Eckstein, 1998). Esta autora propone un método iterativo en el que los alumnos van asimilando las particularidades de la arquitectura de una aplicación compleja (en este caso, Eckstein propone un framework) mediante un ciclo formado por clases teóricas, role-play, tareas de desarrollo y discusión. En este caso, el role-play es usado para ver cómo se comportan durante la ejecución las distintas partes vistas en la parte teórica. Posteriormente, estos conocimientos son reforzados mediante el desarrollo de pequeños ejercicios de laboratorio y mediante la evaluación. Esta propuesta hace uso del role-play al más puro estilo de herramienta de diseño, para la comprensión de una arquitectura, y puede ser muy útil para la enseñanza de conocimientos más avanzados de diseño.

Como se describió en la Sección 2.4.8 la utilización de frameworks y casos de estudio es una forma bastante común de introducir los patrones de diseño. Algunos autores, sin embargo, han propuesto alternativas más participativas de cara a los alumnos. En (Schmolitzky, 2005) se propone que el alumno participe en la extensión de un caso de estudio. Primeramente se le da el caso de estudio y su código fuente para que comprenda su funcionamiento. Posteriormente, se propone que se añada una funcionalidad al sistema. Generalmente, esta extensión hace que sea necesaria la aplicación de un patrón de diseño. Una vez que se ha desarrollado la nueva funcionalidad, el instructor estudia la solución adoptada, sus consecuencias y posibles diseños alternativos. Aunque más participativa, esta propuesta no llega a utilizar el role-play. De todas formas, esta técnica podría encajar perfectamente como medio para que el alumno comprenda el funcionamiento del caso de estudio y evalúe la solución obtenida.

4.3. Una propuesta de role-play para la enseñanza de patrones de diseño

La técnica de aprendizaje activo vista en las primeras secciones de este capítulo no sólo puede ser empleada para la enseñanza de los conceptos elementales y para diseño orientado a objetos. El role-play tiene suficiente valor pedagógico en cuanto al fomento e intercambio de experiencias de diseño como para ser utilizado en la comprensión de conceptos más avanzados, como son los patrones de diseño. Por esto, el autor de esta Tesis ha decidido realizar una aproximación del role-play a la enseñanza de patrones de diseño (Jiménez-Díaz et al., 2006, 2008).

Al igual que la propuesta descrita en (Schmolitzky, 2005), vista al final de la anterior sección, nuestra aproximación se basa en el desarrollo participativo de un caso de estudio. Sin embargo nuestra propuesta sí usa role-play para comprender el funcionamiento de la aplicación. En esta propuesta se hace uso de las sesiones de role-play y de las tarjetas CRC para la comprensión del funcionamiento de la aplicación usada como caso de estudio. Esta técnica sustituye a la ejecución de la aplicación o a la revisión del código fuente, centrando al alumno en la comprensión del diseño de la aplicación. Aunque no hay que obviar los detalles de implementación, la aplicación de los patrones de diseño se realiza, como su propio nombre indica, principalmente a nivel de diseño. Bajar a nivel de código hace que, aunque se comprenda el funcionamiento de la aplicación y la interacción entre los objetos, se pierda de vista la arquitectura de clases que conforman el sistema.

La propuesta que aquí se realiza se basa en el desarrollo iterativo de un caso de estudio mediante la aplicación sucesiva de patrones de diseño. Pero en lugar de *imponer* la aplicación de dichos patrones, lo que se pretende es que el alumno se haga consciente de la necesidad de aplicación de los mismos. Valiéndonos de nuestra propia experiencia se ha detectado que, en muchas ocasiones, cuando los alumnos aprenden los patrones de diseño realizan un uso indiscriminado de los mismos, complicando excesivamente el diseño de la aplicación, en ocasiones de manera innecesaria. La propuesta descrita a continuación pretende que el alumno se haga consciente de que los patrones de diseño se emplean para resolver problemas de diseño o para anticiparse a futuras extensiones de la aplicación pero que no se han de usar sin una motivación concreta.

La aplicación de los patrones de diseño se realiza mediante la técnica de refactorización o *refactoring*. Esta técnica es ampliamente usada en la comunidad de desarrollo de software y se emplea en la limpieza y mejora de la estructura del software de manera eficiente y controlada. La aplicación de la refactorización cambia la estructura de un diseño software para hacerla más fácil de entender y modificar sin que se produzcan cambios observables en el comportamiento de dicha aplicación (Fowler, 1999).

La tarea de refactorización no es algo que esté planeado dentro del ciclo de desarrollo de software sino que es una técnica que se aplica sólo cuando se necesita. En (Fowler, 1999) se describe la noción de "cuándo" es necesario refactorizar en términos de los "malos olores", descritos en la Sección 2.3.2. Estos "malos olores" suscitan la aplicación de la refactorización para eliminarlos. Una vez detectados estos "malos olores" la refactorización propone aplicar una serie de estrategias que los eliminen. Estas estrategias, que en esta memoria de Tesis se denominan "primitivas", en contraposición a las estrategias de refactorización basadas en patrones (Kerievsky, 2004), han sido catalogadas en (Fowler, 1999) y proponen la aplicación de modificaciones sencillas como renombrar un método o mover un método de una clase a otra.

Aunque estas estrategias han sido introducidas y aplicadas en cursos iniciales de la carrera de Informática, como se puede ver en (Smith et al., 2006) y (Stoecklin et al., 2007), nuestra propuesta de enseñanza también hace uso de las estrategias de refactorización basadas en patrones para eliminar esos "malos olores" y obtener un mejor diseño del caso de estudio propuesto.

La propuesta para enseñar patrones de diseño se basa en la participación del alumno en el desarrollo iterativo de un caso de estudio elegido por el instructor. Los alumnos inicialmente reciben unas breves nociones teóricas sobre los patrones de diseño, usando las clases magistrales tradicionales. Posteriormente, el instructor es el responsable de introducir el caso de estudio que se va a emplear a continuación. Este caso de estudio ha de ser sencillo aunque con la suficiente complejidad como para que pueda soportar la aplicación de patrones de diseño. Además, el instructor propone un diseño inicial para esta aplicación. Este diseño ha de ser sencillo y ha de parecer coherente con la aplicación que se quiere desarrollar. Hay que evitar diseños demasiado complicados o que hayan sido forzados para la aplicación posterior de los patrones de diseño. A continuación se desarrollarán una serie de iteraciones, cada una de ellas guiada por un determinado requisito funcional del caso de estudio y cuyo objetivo sea la aplicación de un determinado patrón de diseño.

En cada una de estas iteraciones se aplica una técnica que en (Astrachan et al., 1998) se conoce como *"Before and After"*: el alumno se encuentra con un diseño inicial (o diseño *Before*) sobre el que ha de incluir la funcionalidad solicitada; aunque el diseño parece aceptable, el alumno se encuentra con problemas a la hora de incluir dicha funcionalidad o para poder extenderla en un futuro, por lo que se le propone que los alumnos seleccionen y apliquen un patrón de diseño para mejorarla; la aplicación del patrón genera un diseño mejorado (diseño *After*), que el alumno valorará y comparará con el anterior para observar los beneficios y las consecuencias de aplicar el patrón de diseño.

Cada una de las iteraciones se divide en los siguientes pasos (Jiménez-Díaz et al., 2006):

1. Selección de un escenario de diseño. El instructor proporciona el diseño inicial del caso de estudio para esta iteración. Este diseño se describe mediante un sencillo diagrama de clases y cada clase vendrá representada como una tarjeta CRC. Generalmente, este diseño suele ser el que se ha obtenido en la anterior iteración aunque el instructor puede añadir nuevas clases y métodos. El instructor también ha de seleccionar la funcionalidad de la aplicación que va a ser añadida en el caso de estudio. En lugar de esto, el instructor también puede decidir mejorar alguna de las funcionalidades ya revisadas en anteriores interacciones. Asociado a este requisito funcional el instructor dispone de un conjunto de escenarios de ejecución de la aplicación que los alumnos simularán haciendo uso del role-play.

2. Asignación de roles. Cada escenario dispone de una serie de objetos o roles que los alumnos han de interpretar durante la sesión de role-play. El instructor selecciona qué alumno va a interpretar cada uno de los roles del escenario. Esta selección se puede realizar de acuerdo a determinados criterios: al azar, de acuerdo a los conocimientos y habilidades de los alumnos, por la familiaridad de un alumno con su rol, etc. A cada alumno se le proporciona la tarjeta CRC correspondiente a la clase del objeto que va a interpretar. El instructor también muestra el RPD inicial, donde aparecen los objetos que van a intervenir, las relaciones existentes entre dichos objetos y el estado actual de cada uno de ellos. Es muy importante que los alumnos tengan presente el estado actual del objeto ya que se puede simular un mismo escenario varias veces, modificando el estado de algunos de los objetos, para observar los distintos comportamientos de la aplicación de acuerdo a dichos estados.

3. Simulación. En función del escenario de ejecución seleccionado, el instructor envía el primer mensaje al objeto (alumno) apropiado. El resto de la simulación está dirigida por los alumnos, que se comunicarán entre ellos para conocer las responsabilidades de cada una de las clases y saber los mensajes que se han de enviar para completar el escenario propuesto. A medida que se producen los pasos de mensaje, el instructor los va registrando en el RPD, que será como el historial de la ejecución. Cuando un mensaje provoca el cambio en el estado de un objeto, el alumno responsable de la interpretación de este rol informará al instructor, que lo registrará en el RPD. El instructor también puede intervenir cuando los alumnos se encuentran bloqueados, para ayudarles a continuar con la ejecución, dándoles pistas de lo que han de hacer o decidiendo cuál es el siguiente mensaje que se ha de enviar.

4. Modificación. Las sesiones de role-play han servido para que el alumno se haya familiarizado con la aplicación y con el diseño propuesto y para que se haya hecho consciente de que, aunque el diseño inicial parecía bueno, es mejorable. Se pasa entonces a la fase de modificación en la que se realizará la refactorización. Los alumnos son responsables de seleccionar el patrón de diseño que se va a usar y de su instanciación. La instanciación de un patrón consiste en decidir qué clases y métodos del diseño inicial van a interpretar cada uno de los roles asociados a un patrón de diseño concreto. Si es necesario, los alumnos pueden añadir nuevas clases e interfaces o modificar las ya existentes. Eventualmente puede ocurrir que el nuevo diseño necesite nuevas modificaciones menores suscitadas por la aplicación del patrón, como la inclusión o eliminación de métodos y atributos. Tras esto, los alumnos volverán a la etapa anterior (Simulación) para repetir los mismos

escenarios de ejecución con el nuevo diseño. El instructor será el responsable de decidir si, según la variación introducida en el diseño, la simulación se ha de reiniciar o si, por el contrario, el cambio no afecta a lo anterior y se puede continuar desde donde estaba.

5. Evaluación. Una vez concluidos todos los ciclos necesarios de Simulación-Modificación se pasa a la etapa de la evaluación del nuevo diseño obtenido. En esta fase, el instructor hace ver a los alumnos las mejoras obtenidas en la aplicación de acuerdo al patrón instanciado. También se hacen ver las consecuencias y los inconvenientes del patrón seleccionado en este ejemplo. Si en la etapa de Modificación los alumnos seleccionaron distintos patrones, el instructor y los alumnos discuten sobre la aplicación de las diferentes alternativas al caso de estudio dado. En esta etapa incluso se puede decidir instanciar otro de los patrones seleccionados y retornar a la etapa de Simulación, para así hacer más patente las diferencias entre los patrones seleccionados y la imposibilidad o idoneidad de aplicar otros en lugar del seleccionado.

Como se ha podido ver, en gran parte de esta propuesta el proceso de enseñanza está centrado en el alumno. Éste se convierte en parte activa del proceso de enseñanza mediante la intervención en las sesiones de role-play. También hay que destacar que el alumno se involucra en un proceso de desarrollo de software a nivel de diseño, lo que le hace adquirir nuevas experiencias en este campo, algo imprescindible para la formación del alumno en el diseño del software orientado a objetos, y en el uso de patrones de diseño, en particular.

4.4. Experiencias en la enseñanza de patrones de diseño usando role-play

El enfoque pedagógico para la enseñanza de patrones de diseño descrito en el apartado anterior no ha quedado sólo plasmado de manera teórica. Durante los cursos académicos 2005-2006 y 2006-2007, en la Facultad de Informática de la Universidad Complutense de Madrid se ha venido impartiendo un seminario llamado "Patrones de diseño aplicados al desarrollo de editores gráficos en Java con JHotDraw". Los principales objetivos de este seminario son la familiarización de los alumnos con los conceptos fundamentales de los patrones de diseño "Gang of Four" (GoF (Gamma et al., 1995)) y observar su aplicación en el desarrollo del framework JHotDraw, para la creación de editores gráficos (Jiménez-Díaz et al., 2006, 2008). De cara a esta Tesis, estos seminarios han servido como medio para poner en práctica el enfoque pedagógico descrito en el apartado anterior y para evaluar su impacto en el alumnado (Jiménez-Díaz et al., 2008), como se verá en la Sección 4.5.

La duración de cada una de las ediciones de este seminario ha sido de 20 horas. Debido a la limitación de tiempo, se ha realizado una selección de los patrones enseñados tanto de manera teórica como mediante su aplicación en un caso de estudio. De entre todo el catálogo descrito en (Gamma et al., 1995), se seleccionaron los siguientes patrones para su presentación teórica: *Método de Factoría*, *Estrategia*, *Prototipo*, *Estado*, *Decorador*, *Composición*, *Factoría Abstracta*, *Singleton, Adaptador, Puente, Observador, Mediador, Método de Plantilla, Proxy, Fachada, Comando* e *Iterador*. De entre todos estos, cinco de ellos fueron usados mayoritariamente durante las sesiones prácticas basadas en role-play y refactorización basada en patrones: *Prototipo, Estado, Composición, Observador* y *Adaptador*.

El caso de estudio utilizado durante ambas ediciones ha sido un editor gráfico sencillo. Este caso de estudio ha sido seleccionado por la familiaridad que los alumnos tienen con este tipo de herramientas y por la experiencia del autor en el framework JHotDraw. En líneas generales, el editor desarrollado es una simplificación del editor gráfico que se puede crear usando dicho framework y que aparece como ejemplo de aplicación en la distribución de JHotDraw.

El editor gráfico (como se puede ver en la Figura 4.3) se compone de un lienzo donde se irán dibujando las figuras creadas, una región donde aparece información sobre el tamaño de la seleccionada, y una barra con las siguientes herramientas:

- Dos herramientas de creación de figuras (una por cada tipo). Sirven para crear rectángulos y elipses en el lienzo.
- Una herramienta de selección. Permite seleccionar figuras, moverlas y cambiar su tamaño.
- Una herramienta de agrupación. Sirve para crear un grupo de figuras. Dicho grupo hace que todas las figuras que se encuentren en su interior se comporten como una sola.

En la Figura 4.4 se muestra el diseño inicial del editor gráfico, compuesto por las clases que se detallan a continuación:

VentanaEditor Es la clase responsable de coordinar la aplicación. Contiene los botones de la barra de herramientas y es responsable de responder a los eventos de pulsación de dichos botones.

BotonHerramienta Es la clase que representa cada uno de los botones de creación figuras que hay en la barra de herramientas. En ejecución existe una instancia para cada uno de los tipos de figuras que se pueden crear.

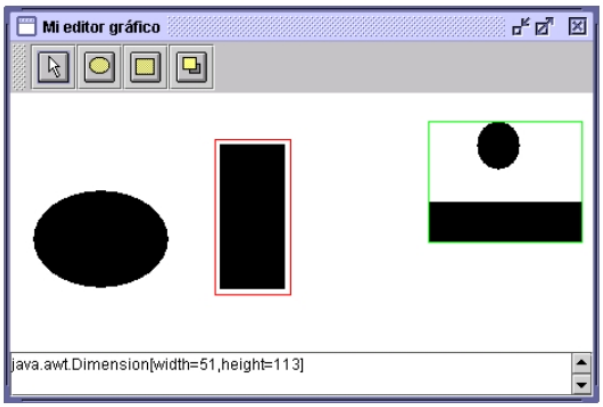


Figura 4.3: Captura del editor gráfico usado como caso de estudio.

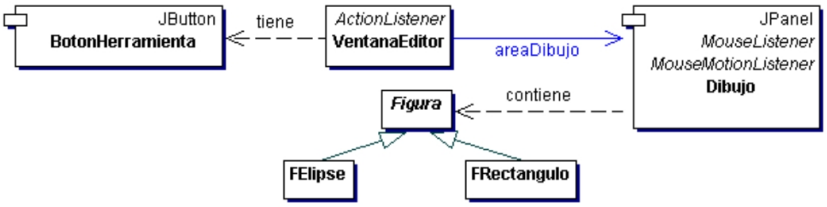


Figura 4.4: Diagrama de clases del diseño inicial del caso de estudio proporcionado por el instructor.

Dibujo Es la clase que representa el lienzo del editor y se encarga de almacenar las figuras que son creadas. Es responsable de dibujarlas y de responder a los eventos que se producen cuando el usuario pulsa sobre el lienzo.

Figura Es la superclase que representa a las figuras que se pueden dibujar usando el editor. Cada figura es responsable de dibujarse y de saber cómo modificar el rectángulo que la circunscribe (llamado área de presentación). Bajo esta clase aparecen las subclases FElipse y FRectangulo, que son las particularizaciones de la clase Figura para las elipses y los rectángulos, respectivamente.

En la Figura 4.5 se pueden ver las tarjetas CRC que describen las clases del diseño de partida. Como se puede observar, las tarjetas CRC no tienen una especificación informal de las responsabilidades, como se usan generalmente en el diseño orientado a objetos, sino que siguen una sintaxis similar a la de Java. Esta decisión se ha tomado teniendo en cuenta que los patrones de diseño tienen una importante componente de implementación. Además, esta sintaxis es familiar a los alumnos, por lo que facilita la comprensión de las tarjetas CRC.

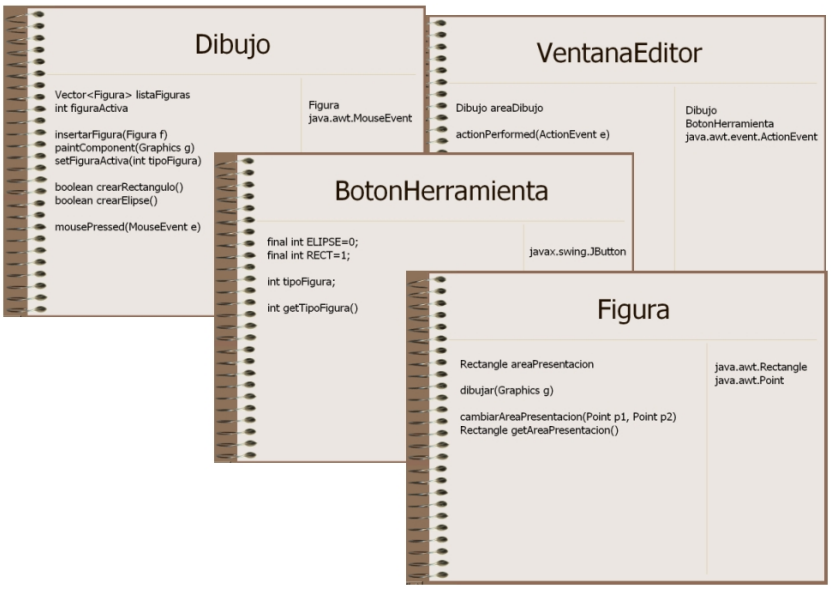


Figura 4.5: Tarjetas CRC de las clases proporcionadas inicialmente para el caso de estudio.

A partir de este diseño inicial se realizaron cinco sesiones prácticas[2]. A continuación se realiza un breve resumen sobre el objetivo perseguido con cada una de ellas y el método seguido para alcanzarlo:

Sesión 1 El objetivo de esta sesión es la aplicación del patrón *Prototipo*. Para ello se simulan varios escenarios de ejecución de creación de figuras de manera simplificada, es decir, las figuras son creadas con un tamaño fijo al pulsar sobre el lienzo. Durante la simulación, los alumnos observan que el diseño propuesto obliga a consultar qué tipo de figura es el seleccionado para decidir la clase de figura que se ha de crear. Estas preguntas se convertirán en estructuras condicionales que se hacen cada vez más complejas a medida que se quieren añadir nuevos tipos de figuras. El patrón *Prototipo* elimina estas estructuras condicionales y facilita la extensión del editor de cara a la creación de nuevos tipos de figuras. Durante la evaluación se realizan comparaciones con el patrón *Método de Factoría* y *Factoría Abstracta*, los cuales suelen ser propuestos por los alumnos durante la etapa de Modificación. La evaluación conduce a la conclusión de que tanto *Método de Factoría* como *Prototipo* son los más correctos de aplicar y que se opta por *Prototipo* debido a que el primero obliga a la creación de nuevas clases. Por último también se hace constar que este patrón obliga a la creación de

[2]Todo el material de estas sesiones se encuentra disponible en `http://gaia.fdi.ucm.es/grupo/projects/virplay/`

un método de clonación que, dependiendo de la estructura de la clase a clonar, puede ser complejo de implementar.

Sesión 2 En esta segunda sesión se pretende que los alumnos hagan uso del patrón *Estado*. Partiendo del diseño obtenido en la anterior sesión, los alumnos simulan una serie de escenarios de ejecución en los que la creación de las figuras se realiza a la manera en la que se suele hacer en cualquier editor gráfico (pinchar-arrastrar-soltar). Así mismo, simulan las acciones de seleccionar y mover, las cuales se realizan haciendo uso de la herramienta de selección. Una de las conclusiones que motivan la necesidad de refactorizar este diseño es que la clase Dibujo se convierte en una clase muy grande y con gran número de atributos, algunos de los cuales se usan sólo en acciones muy particulares. Además, observan que las sentencias condicionales se repiten a lo largo de distintos métodos de esta clase y que la adición de nuevas funcionalidades con herramientas se convierte en una tarea muy compleja. La aplicación del patrón *Estado* obliga a la creación de nuevas clases sobre las que Dibujo delega su ejecución y que simplifica enormemente la extensión de la funcionalidad. Durante la evaluación se observa la diferencia conceptual entre el patrón *Estado* y el patrón *Estrategia*, cuyas estructuras son similares, y se observa que la herramienta de selección es susceptible de una nueva refactorización usando el patrón *Estado*, ya que sigue teniendo estructuras condicionales complejas.

Sesión 3 Esta sesión es empleada para poner en práctica el patrón *Observador*. Se parte de un diseño de clases modificado sobre el que se extiende la funcionalidad del caso de estudio añadiendo una línea de edición de texto en la que aparecen las dimensiones de la figura actualmente seleccionada o de la última creada. Además, esta información se actualiza durante la creación, a medida que se arrastra la herramienta de creación. Tras simular el escenario de ejecución en el que se ve cómo se actualiza la información que aparece en dicha línea de edición, se propone cambiar la especificación, haciendo que en lugar de usar una línea de edición se use un cuadro de texto, cuya interfaz para incluir información es distinta a la de la línea de edición. Esta nueva simulación hace ver a los alumnos que el diseño actual acopla enormemente la clase Figura con el medio empleado para presentar la información. Además, la actualización se encuentra dispersa por el código de Figura, lo que puede suponer una fuente de errores a la hora de modificar el medio de presentación de la información. El patrón *Observador* elimina el acoplamiento entre estas clases y facilita la actualización de la información. Además, en la etapa de Evaluación se indica que mediante este patrón se pueden incluir múltiples observadores, aunque también se advierte de la sobrecarga de actualizaciones innecesarias en caso de

que distintos observadores sólo necesiten de una información muy concreta de la figura, lo que obligaría a usar versiones más complejas de este patrón.

Sesión 4 El objetivo de esta sesión es la aplicación del patrón *Composición*. Para ello se pide a los alumnos que incluyan en el editor la funcionalidad de agrupar figuras partiendo del diseño de clases de la sesión anterior. Los alumnos han de simular la ejecución de añadir nuevas figuras a un grupo, cómo se dibuja, cómo se selecciona y cómo se mueve el grupo de figuras. Estas simulaciones ayudan a comprobar que es necesario hacer distinciones en función de si se quieren manipular figuras o grupos de figuras, a pesar de que ambas se comportan de igual manera y tienen una interfaz similar. Esto conduce a que durante la etapa de Modificación se use el patrón *Composición*. La aplicación de este patrón hace que no sean necesarias estas distinciones y que se puedan añadir nuevas figuras compuestas al editor gráfico.

Sesión 5 Esta última sesión tiene por objetivo la aplicación del patrón *Adaptador*. Para ello se añade un nuevo requisito funcional que consiste en redimensionar una figura usando la herramienta de selección. Para ello, al seleccionar una figura aparecen dos manipuladores: uno en la esquina superior izquierda que sirve para redimensionar proporcionalmente la figura; y uno en la esquina inferior derecha que sirve para estirar o encoger la figura dejando fija la esquina superior izquierda. Los alumnos parten de un diseño inicial en el que se incluyen las clases para representar a estos manipuladores. La simulación de los escenarios de cada uno de estos manipuladores, junto con el de mover, hacen ver al alumno que, de nuevo, aparecerán estructuras condicionales complejas que no ayudan a la extensibilidad de la aplicación. Para eliminar esto, se realiza la refactorización haciendo uso del patrón *Adaptador*, de modo que los manipuladores de la figura se comportan como adaptadores del comportamiento para la herramienta de selección, dependiendo del tipo de manipulador que se esté usando. En la etapa de Evaluación se compara la aplicación de este patrón con el del patrón *Estado* y se hace ver que este patrón puede implicar la revisión de la interfaz de la clase adaptada, así como la necesidad de buscar la interfaz más estricta posible con la que se puedan pasar todos los datos necesarios para los distintos tipos de manipulaciones, lo que puede generar métodos con largas listas de parámetros.

Tras concluir estas sesiones, en la Figura 4.6 se puede observar la arquitectura final alcanzada para el caso de estudio.

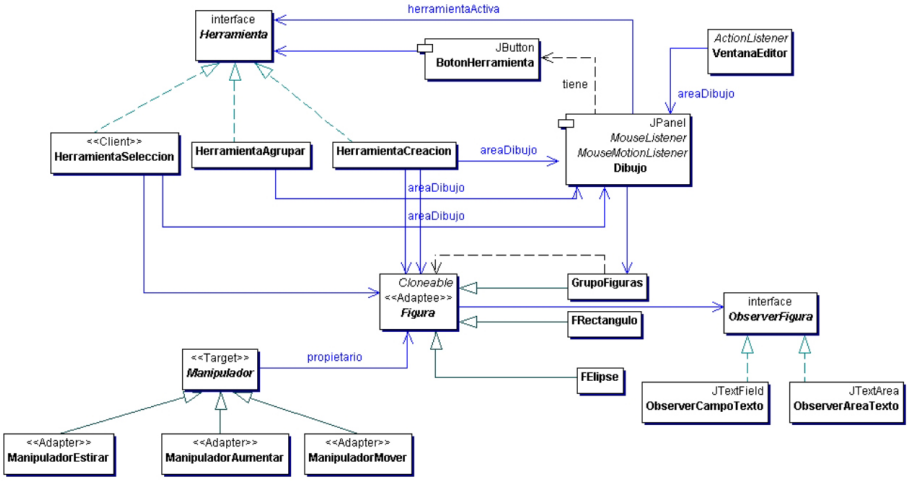


Figura 4.6: Diagrama de clases del diseño final del caso de estudio.

4.5. Evaluación de las experiencias

Tras cada una de las ediciones del seminario "Patrones de diseño aplicados al desarrollo de editores gráficos en Java con JHotDraw" se ha valorado la opinión de los alumnos acerca de la aplicación de las técnicas de aprendizaje activo y se ha medido el impacto de su aplicación de cara al aprendizaje del alumno [3].

En estos seminarios participaron un total de 61 alumnos pertenecientes a la Facultad de Informática, tanto de las titulaciones técnicas como de la Ingeniería en Informática. Para garantizar unos mínimos conocimientos de los alumnos en conceptos de orientación a objetos se seleccionaron aquellos alumnos que habían completado al menos el 50 % de los créditos necesarios para completar su titulación. Para valorar los conocimientos previos de los alumnos se realizó un Pretest que mide los conocimientos de los mecanismos básicos de orientación a objetos en Java. Dicho Pretest también se empleó para conocer cuáles eran los conocimientos previos de los alumnos en patrones de diseño.

Los resultados del Pretest concluyeron que los alumnos que han participado en ambas experiencias tenían un conocimiento medio-alto en conceptos de orientación a objetos usando el lenguaje Java, ya que los alumnos de ambas ediciones obtuvieron una puntuación media de 6,4 puntos sobre 10. En cambio, los conocimientos de los alumnos en cuanto a patrones de diseño eran escasos o nulos, siendo la puntuación media obtenida de 3,4 puntos sobre 10.

A continuación vamos a pasar a analizar las evaluaciones realizadas y los

[3] Los cuestionarios empleados en estas evaluaciones se encuentran disponibles en `http://gaia.fdi.ucm.es/grupo/projects/virplay/`

resultados obtenidos en cada una de las ediciones. Un resumen de las mismas se ha incluido en (Jiménez-Díaz et al., 2008).

4.5.1. Primera edición: puesta en marcha y valoración subjetiva

La primera edición fue una experiencia piloto, tanto para los alumnos como para los docentes, en la que se comenzó a hacer uso de las técnicas de aprendizaje activo para la enseñanza de orientación a objetos. Esta edición se celebró durante el mes de julio de 2006 y en ella participaron un total de 24 alumnos.

Este primer seminario se organizó de la siguiente manera:

- Durante el primer 20 % del seminario, se impartieron de manera teórica los conceptos fundamentales sobre los patrones de diseño GoF, sus ventajas e inconvenientes. Esta parte del seminario fue impartida a la manera tradicional, siguiendo las explicaciones y los ejemplos descritos en (Gamma et al., 1995) y sin emplear ninguna aplicación software concreta como fuente de aplicación de los patrones explicados.

- A continuación, el 70 % del tiempo del seminario se empleó en la implicación del alumno en tareas de refactorización orientada a patrones y en la interpretación de escenarios de role-play. Por supuesto, los alumnos habían recibido previamente las pautas básicas sobre las técnicas empleadas —tarjetas CRC y role-play. También se les explicaron las especificaciones y el diseño inicial del editor gráfico que se empleó para la impartición del seminario.

- El 10 % final del seminario fue empleado en realizar una introducción al diseño del framework JHotDraw. Durante estas sesiones, se pretendió que los alumnos observaran y valoraran la aplicación de algunos de los patrones de diseño vistos en clase en una aplicación real y más extensa que la desarrollada por ellos.

El objetivo principal de esta primera edición era observar cómo era aceptada por los alumnos la aproximación de enseñanza empleada, ya que hasta el momento no se habían impartido clases en las que el alumno formara parte de manera tan activa en el desarrollo de las mismas. Por tanto, durante este seminario se pretendió evaluar la reacción de los estudiantes ante las sesiones de role-play y valorar sus opiniones sobre la utilización de tarjetas CRC y RPDs en lugar de código fuente para comprender el funcionamiento interno de una aplicación.

Tras la celebración del seminario, los alumnos completaron un cuestionario en el que se evaluaban distintas partes de la aproximación y las técnicas usadas en el desarrollo del seminario. La valoración global de la actividad fue positiva y los alumnos consideraron, en general, que este tipo de

aproximaciones resultan muy motivadoras. Los alumnos destacaron la importancia del conocimiento y de la comprensión de los patrones de diseño como parte del diseño orientado a objetos y valoraron muy positivamente su enseñanza mediante la puesta en práctica de los mismos durante el desarrollo del editor gráfico.

Los alumnos valoraron muy positivamente la calidad y la utilidad del curso, ya que la puntuación media obtenida en estos parámetros fue superior a 4 puntos sobre 5. Por otro lado, también se les preguntó sobre la utilidad del empleo del editor gráfico para la enseñanza de patrones de diseño, frente a los ejemplos de (Gamma et al., 1995). En este caso, aunque consideraron que ambos eran útiles, valoraron mucho más la utilización del editor gráfico, que obtuvo un 4,6 sobre 5, frente a los 3,3 puntos sobre 5 que obtuvo la utilidad de los ejemplos usados durante la parte más teórica del seminario.

Los alumnos también consideraron de mucha utilidad las técnicas que se emplearon durante la impartición del seminario: las tarjetas CRC obtuvieron una valoración de 4.4 sobre 5, las simulaciones de role-play recibieron un 4 sobre 5 y los RPDs, un 4,2 sobre 5. Por último, los alumnos afirmaron encontrarse cómodos con el uso de las tarjetas CRC y los RPDs para seguir el desarrollo del editor gráfico y no consideraron necesario usar el código fuente para comprender el funcionamiento de la aplicación en lugar de las representaciones anteriores. De todas formas, algunos alumnos sí que apuntaron que les gustaría tener el código fuente al final del seminario para poder revisar la implementación de los patrones de diseño estudiados.

4.5.2. Segunda edición: ajustes y análisis de la efectividad

El segundo seminario se celebró en invierno de 2006. A esta edición asistieron un total de 37 alumnos. Gracias a la experiencia del anterior seminario, se hicieron pequeñas modificaciones con el fin de mejorar el funcionamiento del seminario y obtener una valoración más objetiva de la efectividad de la aproximación basada en aprendizaje activo frente a las clases pasivas tradicionales.

La organización de este segundo seminario mantuvo una primera parte en la que se impartían las nociones básicas sobre patrones de diseño. Posteriormente, los alumnos se dividieron en dos grupos:

- El grupo de control (GC) completó las dos primeras iteraciones de diseño siguiendo una aproximación en la que el instructor no promovía la participación de los alumnos. Durante estas dos iteraciones, el instructor se limitaba a mostrar el diseño inicial, explicar los problemas encontrados en este primer diseño, seleccionar el patrón a aplicar, instanciar dicho patrón en el diseño inicial y demostrar que el patrón soluciona el problema encontrado.

- El grupo experimental (GE) completó las mismas iteraciones siguiendo

la aproximación basada en las sesiones de refactorización y role-play.

Posteriormente, todos los estudiantes fueron reagrupados y completaron las iteraciones que faltaron utilizando las sesiones de role-play. Dado el gran número de alumnos, durante estas sesiones sólo unos cuantos participaban. Mientras, el resto de alumnos observaban la simulación interpretada por sus compañeros. De nuevo, el seminario concluyó con el análisis del framework JHotDraw.

Durante este segundo seminario se pretendió ser más ambicioso a la hora de evaluar la aproximación propuesta. Los alumnos tuvieron que completar, además del Pretest adicional dos tests más: uno tras las dos primeras iteraciones de diseño (Test1) y otro tras finalizar el seminario (Test2). Los resultados de estos tests se han usado para:

- Comparar los conocimientos adquiridos por los alumnos que pertenecen al GE frente a los alumnos pertenecientes al GC.
- Medir si los alumnos del GC mejoraron su aprendizaje tras participar en las sesiones de role-play frente a las clases no participativas.

Además, los alumnos realizaron un cuestionario final con el fin de evaluar de nuevo la aproximación propuesta desde un punto de vista más subjetivo. Los resultados de estos cuestionarios se han usado para:

- Corroborar la buena aceptación obtenida por parte de los estudiantes de la primera edición.
- Conocer si los alumnos prefieren participar en el role-play en lugar de ser meros observadores del mismo.
- Conocer si los alumnos incluidos en el GC prefieren las iteraciones de diseño que siguen una aproximación menos participativa o si, por el contrario, prefieren aquella aproximación en las que se les hace partícipes de las decisiones de diseño.
- Valorar la utilidad subjetiva de las tareas de refactorización para la enseñanza de patrones de diseño.

Mientras que la evaluación de los cuestionarios ha sido meramente descriptiva, los resultados de los tests han sido analizados estadísticamente para determinar si las diferencias entre las muestras tomadas en los grupos GE y GC son lo suficientemente significativas como para concluir que ambos grupos se comportan realmente de manera diferente. Para realizar esto, se ha aplicado el test t, en el caso de que ambas muestras se distribuyan normalmente, o el test de Wilcoxon, en caso contrario. La normalidad de la distribución de las muestras se ha determinado mediante el test de Shapiro-Wilk.

Tabla 4.1: Notas medias obtenidas por los alumnos tras completar el Test1 (izquierda) y el Test2 (derecha).

		Test1		Test2	
Grupo		Total	Preguntas prácticas	Total	Preguntas Prácticas
GE	Media	**7,6316**	**7,0526**	**7.7895**	**7,3026**
	N	19	19	19	19
	Desviación estándar	1,46099	1,80966	1,08418	1,19774
GC	Media	**7,3889**	**6,8889**	**8,000**	**7,7083**
	N	18	18	18	18
	Desviación estándar	1,19503	1,84355	0,90749	1,07187
Total	Media	7,5135	6,9730	7,8919	7,5000
	N	37	37	37	37
	Desviación estándar	1,32543	1,80257	0,99398	1,14109

Inicialmente se realizó una división aleatoria de los alumnos en los grupos GC y GE. El análisis de los resultados del Pretest reveló que no había una diferencia significativa entre ambos grupos, por lo que se puede considerar que ambos provienen de la misma población, un requisito fundamental para valorar las comparaciones realizadas entre ambos grupos.

El Test1 se compone de 10 preguntas, 5 sobre conceptos más teóricos de patrones de diseño y 5 más prácticas, sobre su aplicación y su reconocimiento en un diseño software dado. Se realizó una comparación de los resultados obtenidos por ambos grupos que mostró que el grupo GE obtuvo mejores resultados (si bien la diferencia no se considera estadísticamente significativa). Además, se compararon los resultados obtenidos en las preguntas prácticas. De nuevo, los resultados obtenidos por el GE fueron mejores (aunque las diferencias de resultados entre ambos grupos no se consideran estadísticamente significativas). Los resultados obtenidos se pueden ver en la Tabla 4.1.

Tras reagrupar a los alumnos y completar el diseño del editor gráfico se realizó el Test2. Este test era meramente práctico, de modo que las 8 preguntas consistían en la identificación y en la aplicación de patrones de diseño. De nuevo se compararon las diferencias entre los resultados obtenidos por ambos grupos, como se pueden ver en la Tabla 4.2. Sorprendentemente, el grupo GC experimentó una gran mejoría en sus calificaciones, mucho mayor que la obtenida por el grupo GE, aunque el test t determinó que la diferencia entre los resultados medios de uno y otro grupo no son estadísticamente significativos.

Tabla 4.2: Diferencias medias entre los resultados obtenidos en el Test1 y en el Test2.

	Grupo	N	Media	Desviación estándar	Error estándar
Diferencia total	GE	19	**0,1579**	1,2139	0,2785
	GC	18	**0,6111**	1,5392	0,3628
Diferencia preguntas prácticas	GE	19	**0,2500**	1,3307	0,3052
	GC	18	**0,8194**	1,9399	0,45724

Por último, los resultados de estos tests se han usado para medir el impacto de participar en las sesiones de role-play en lugar de actuar como un mero observador de estas simulaciones. Para realizar esto se hizo un estudio de los resultados obtenidos por cada uno de los alumnos comparando las calificaciones obtenidas en aquellos patrones en los que el alumno había intervenido en la simulación de role-play asociada a dicho patrón, y las calificaciones en los patrones en los que el alumno fue únicamente observador pasivo de la simulación. Mientras que todos los alumnos obtuvieron la mayor calificación posible (10 sobre 10) en todas las preguntas asociadas a los patrones en los que habían intervenido, sólo el 50 % de ellos completaron correctamente todas las preguntas sobre los patrones en los que habían participado pasivamente, obteniendo en media un resultado de 8,85 sobre 10. El test t permite concluir que existe una diferencia realmente significativa en los resultado obtenidos como fruto de participar activamente frente a ser un espectador pasivo de la simulación.

La evaluación de los cuestionarios arrojó unos resultados similares a los de la primera edición en cuanto a la aceptación de los alumnos de la aproximación pedagógica propuesta. Los alumnos se sintieron muy motivados y apreciaron la aplicación de los patrones de diseño en un ejemplo práctico. Los alumnos valoraron positivamente la calidad del curso para tener una vista global de los patrones de diseño, así como para comprender la intención, la estructura y las consecuencias de aplicación de cada uno de los patrones de diseño estudiados.

Al igual que en la anterior edición, los alumnos consideraron que el caso de estudio empleado fue de mayor utilidad que los pequeños ejemplos usados durante las clases de introducción de los patrones de diseño. El uso del caso de estudio recibió una valoración media de 4,5 sobre 5, mientras que la utilidad de los ejemplos fue menos valorada, un 3,9 sobre 5. Los alumnos también evaluaron positivamente el uso de las tarjetas CRC (3,9 sobre 5), de las sesiones de role-play (3,7 sobre 5), y de los RPDs (4,2 sobre 5). En esta

edición, los alumnos tampoco echaron en falta el código fuente de la aplicación para comprender su comportamiento y la gran mayoría consideraron que les era suficiente con las tarjetas CRC (70 %) y los RPDs (81 %).

A diferencia de la anterior edición, los cuestionarios también fueron empleados para conocer las preferencias de los alumnos en cuanto a la participación en las clases. El 66 % de los alumnos que asistieron a las sesiones de refactorización no participativas afirmaron preferir las sesiones de refactorización en las que se empleaban las simulaciones de role-play. Además, los alumnos consideraron que la participación en las sesiones de role-play era claramente más útil para la comprensión de los patrones de diseño que el mero hecho de observar a sus compañeros durante dichas simulaciones. En una escala entre 0 y 5, en la que el 5 significa "Participar en las sesiones de role-play es mucho más útil que observar", los alumnos valoraron esta afirmación con una nota media de 3,7.

4.6. Conclusiones

A lo largo de este capítulo se ha estudiado la aplicación de técnicas de aprendizaje activo para la enseñanza de la orientación a objetos. En particular, se ha analizado el uso de las sesiones de role-play soportadas por la utilización de las tarjetas CRC como medio para enseñar distintos conceptos relacionados con la orientación a objetos.

Este capítulo no ha sido una mera revisión de la aplicación de estas técnicas sino que se ha realizado una propuesta de metodología que las incluye para la enseñanza de patrones de diseño, fundamentales para la creación de aplicaciones con una arquitectura robusta y fácilmente extensible. Esta propuesta pretende hacer partícipe al alumno en el proceso de diseño de una aplicación. Las técnicas de aprendizaje activo son empleadas para involucrar al alumno en la evaluación de la arquitectura de la aplicación desarrollada. Esta propuesta también fomenta el uso de los patrones de diseño como soluciones a problemas concretos de diseño, en lugar de explicarlos de manera aislada, lo que provoca su uso indiscriminado. Para ello, se proponen tareas de refactorización, en las que los alumnos aplican patrones de diseño para resolver aquellos problemas encontrados durante la evaluación de la aplicación mediante sesiones de role-play.

Esta propuesta ha sido puesta en práctica durante los cursos académicos 2005-2006 y 2006-2007. A su vez, la experiencia ha sido evaluada tanto desde el punto de vista subjetivo de los alumnos como desde el punto de vista más objetivo de los resultados académicos obtenidos por los participantes de la actividad. Los resultados obtenidos han sido satisfactorios y la aceptación de la propuesta por parte de los alumnos ha sido buena.

Desde el punto de vista subjetivo, los alumnos han expresado que se sienten mucho más motivados con este tipo de propuestas. La gran mayoría valo-

ran muy positivamente ser partícipes de su propio aprendizaje. Los alumnos son conscientes de que necesitan experiencia para aumentar sus conocimientos en esta disciplina y valoran que los docentes proporcionen un contexto de enseñanza en el que puedan ser algo más que oyentes y observadores pasivos de documentación de diseño.

Esta valoración subjetiva también ha servido para conocer que los alumnos solicitan detalles de implementación. La propuesta aquí descrita hace hincapié en el uso de los patrones a nivel de diseño pero no es menos cierto que estos artefactos presentan unas particularidades de implementación que los alumnos esperan conocer. En compensación, los alumnos reciben el código final de la aplicación desarrollada. Pero es cierto que en cursos más largos se debería dedicar un tiempo a la implementación de los patrones de diseño en un lenguaje de programación concreto, de igual forma que Eckstein propone sesiones de laboratorio en su metodología de role-play incremental (Eckstein, 1998).

Desde el punto de vista objetivo, se ha podido observar que esta propuesta de aprendizaje activo no puede carecer de unas sesiones teóricas pasivas previas en las que los alumnos tengan una primera toma de contacto con los patrones de diseño. Así mismo, los resultados de la evaluación han puesto de manifiesto que también es más efectivo que el instructor desarrolle algunos episodios de refactorización antes de que los alumnos participen activamente en los mismos. Es probable que la falta de familiaridad con esta técnica haga que los alumnos que no han tenido estas sesiones guiadas no hayan adquirido mayores conocimientos en la medida en la que lo han hecho aquéllos que sí las disfrutaron.

Por último, se ha corroborado que la participación de los alumnos en estas tareas ha estimulado su proceso de aprendizaje, por lo que han obtenido mejores resultados que los meros observadores. La aplicación de este tipo de técnicas demuestra que no sólo el alumno se siente más motivado, sino que aprovecha mejor el tiempo que a ellas dedica.

A pesar de todas las bondades descritas hasta ahora, hemos detectado un problema fundamental en el desarrollo de este tipo de actividades: los recursos que es necesario dedicar. La infraestructura necesaria para organizar este tipo de sesiones y para garantizar que todos los alumnos intervengan en las sesiones de role-play es muy costosa. Desarrollar cada una de las sesiones descritas en la Sección 4.4 supone un gran esfuerzo por parte del instructor y una gran dedicación de horas ya que no se trata de una sesión tradicional. Por otro lado, estas sesiones suponen grupos de trabajo reducidos, por lo que, para los grupos con los que se ha experimentado, el número de instructores debería haber sido mucho mayor. El verdadero valor de esta experiencia es la participación de los alumnos en todas y cada una de las sesiones, algo que no se puede garantizar con los recursos existentes en la actualidad y con el número de alumnos que solicitan este tipo de cursos.

Como se vio en el capítulo anterior, en los últimos años se viene realizando un gran esfuerzo en el desarrollo de herramientas y entornos que sirven de apoyo al docente y al alumno durante el proceso de enseñanza. Pero la mayoría de ellas no cubren tareas de diseño. Además, no fomentan la cooperación entre alumnos para compartir experiencias. Como se ha podido ver, el role-play cubre perfectamente estas dos deficiencias en las clases presenciales. La pregunta que se plantea ahora es si sería viable desarrollar herramientas que simulasen esta técnica de aprendizaje activo para apoyar la enseñanza de la orientación a objetos. Los próximos capítulos intentarán dar respuesta a esta pregunta.

Capítulo 5

Traslado de las técnicas de aprendizaje activo a entornos virtuales de enseñanza

Voy a hacerle una oferta que no podrá rechazar.
El Padrino (1972)

Las técnicas de aprendizaje activo y en particular el role-play, han demostrado ser de utilidad en la enseñanza de la orientación a objetos. Además, las sesiones presenciales de role-play han tenido una buena aceptación por parte de los alumnos, que se sienten especialmente motivados al ser parte activa de las clases. Los alumnos valoran positivamente este tipo de actividades en la enseñanza de diseño de software ya que ellos mismos son conscientes de que las habilidades para el desarrollo de software de calidad sólo son alcanzables poniéndolas en práctica.

Todo ello hace interesante el desarrollo de herramientas de ayuda a la enseñanza de la orientación a objetos que reflejen el uso de esas técnicas de aprendizaje activo. Estas herramientas, por su parte, incorporan una serie de ventajas en términos de flexibilidad en tiempo y espacio para el aprendizaje del alumno, variedad de contenidos con los que practicar y aprender o la reducción del coste en términos de clases e instructores, entre otras (Strother, 2002).

En el Capítulo 3 se realizó un análisis de distintas herramientas que se emplean en la enseñanza de la orientación a objetos. Muy pocas de ellas se han concentrado en poner en práctica el diseño orientado a objetos. Además, casi ninguna favorece la colaboración entre los alumnos, algo muy importante para adquirir experiencias de diseño. El trabajo en grupo permite que los alumnos no sólo aprendan de sus propios diseños sino también de los conocimientos adquiridos por el resto de alumnos.

La falta de herramientas pedagógicas para la práctica de diseño orientado a objetos y los buenos resultados obtenidos en la experiencia descrita en el anterior capítulo nos han conducido a buscar soluciones que combinen lo mejor de ambas alternativas. En este capítulo se desarrollará la solución sencilla y novedosa por la que se ha optado: el traslado de las sesiones presenciales de role-play a entornos virtuales de enseñanza.

Para el desarrollo de esta solución primero se ha realizado un análisis más en profundidad de la experiencia conseguida con las sesiones de role-play desarrolladas en las clases presenciales. En la Sección 5.1 se ha sintetizado un patrón más general de la metodología empleada en el uso de las sesiones de role-play para la enseñanza de diseño orientado a objetos. Sobre este patrón se han ido detectando una serie de aspectos que proporcionan flexibilidad al patrón general. Estos aspectos, descritos en la Sección 5.2, permiten poner en práctica distintos tipos de sesiones, con distintos objetivos pedagógicos pero en las que se siguen aprovechando los beneficios que el role-play proporciona a la enseñanza de la orientación a objetos.

La Sección 5.3 proporciona una visión general de cómo llevar a cabo el traslado de este patrón de sesiones a herramientas software basadas en visualizaciones interactivas. En esta sección se han descrito las características generales del tipo de herramientas propuesto, para las que se ha adoptado la denominación de *Entornos Virtuales de Role-play*. La definición de estos entornos es la principal aportación realizada a lo largo de este trabajo de Tesis. En esta sección también se motiva por qué el tipo de herramientas propuestas se basa en entornos virtuales.

Las siguientes secciones detallan un análisis más en profundidad de las implicaciones de trasladar las actividades de role-play a un entorno virtual. Estas implicaciones han sido divididas en tres tipos de necesidades: las necesidades de representación, las de interacción y las de autoría. Las primeras se detallan en la Sección 5.4 y se centran en los elementos de las sesiones de role-play que han de necesitar de una representación visual dentro del entorno virtual. Las segundas aparecen en la Sección 5.5 y analizan las mecánicas que se han de incluir dentro del entorno virtual de role-play para que el alumno pueda realizar las mismas actividades que se desarrollan durante las sesiones presenciales. Las últimas se describen en la Sección 5.6 y estudian el material que ha de ser desarrollado por parte de un instructor para poder hacer uso de los entornos virtuales de role-play.

La propuesta realizada no consiste únicamente en una herramienta sino en la definición de una familia de entornos virtuales que sean empleados para distintos tipos de sesiones de role-play. En la Sección 5.7 se estudian las distintas decisiones de diseño que hay que tomar a la hora de desarrollar una herramienta concreta de estas características de acuerdo a los ejes de variabilidad analizados en las primeras secciones.

Para dar soporte al desarrollo de este tipo de herramientas se propone una

arquitectura de alto nivel para entornos virtuales de role-play. Ésta aparece en la Sección 5.8 y se ha desarrollado teniendo en cuenta, por un lado, los módulos comunes que han de compartir todos los entornos de esta familia, y, por otro lado, los módulos variables que definirán el distinto comportamiento de las herramientas de acuerdo a las decisiones tomadas según los ejes de variabilidad definidos en la sección anterior.

5.1. Patrón general de una sesión presencial de diseño usando role-play

En el Capítulo 4 se describió nuestra experiencia en el uso de sesiones de role-play para la enseñanza de patrones de diseño. Como ya se destacó en ese mismo capítulo, se consiguió una gran aceptación por parte de los alumnos y unos buenos resultados pedagógicos. Las buenas expectativas creadas con este tipo de sesiones nos ha hecho plantearnos la posibilidad de usarlas para otro tipo de tareas relacionadas con la enseñanza de la orientación a objetos, en particular con el diseño orientado a objetos.

Para ello, se ha decidido revisar el formato de las sesiones de diseño realizadas y plantear un patrón más general con el que se puedan realizar tareas relacionadas con el diseño orientado a objetos, no sólo a nivel de patrones de diseño. Este patrón se puede ver de manera gráfica en la Figura 5.1 y se divide en las siguientes fases:

1. *Selección del escenario de diseño.* El *agente selector* escoge el escenario de diseño sobre el que va a girar la sesión de role-play. Este escenario se compone del caso de estudio o problema de diseño a analizar. También incluye una solución inicial y un conjunto de escenarios de ejecución que serán simulados durante la sesión para comprender la solución que se ha propuesto. Esta información será creada por un *agente generador del escenario.*

2. *Asignación de roles.* Cada escenario de ejecución se compone de una serie de roles —objetos del caso de estudio— que serán interpretados por los alumnos. El *agente asignador de roles* decide qué rol va a ser interpretado por cada alumno y le entrega información sobre la clase asociada al objeto que va a representar. En caso de que haya objetos de poco interés para el alumno, el *agente moderador* interpretará estos roles.

3. *Ciclo de simulación-modificación.* El *agente moderador* inicia uno de los escenarios de ejecución enviando el primer mensaje al alumno que interpreta el rol del objeto correspondiente. A partir de este momento, la simulación corre a cargo de los alumnos: decidir los mensajes que se han de pasar, las modificaciones que se han de realizar en el diseño de

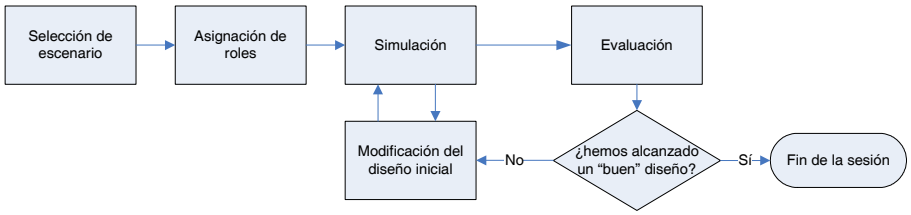


Figura 5.1: Patrón general de una sesión de diseño usando role-play.

clases, y, si es necesario, cuándo concluye la simulación. Los cambios pueden ser de lo más variados, pudiendo obligar en ocasiones a que el *agente moderador* sea el único responsable de realizarlos e, incluso, a que tenga que iniciar de nuevo la simulación. Los alumnos pueden comunicarse entre ellos para obtener más información sobre las responsabilidades que cada uno de ellos es capaz de llevar a cabo. Durante la simulación, el *agente moderador* puede ayudar a los alumnos si observa que se encuentran bloqueados.

4. *Evaluación.* Tras la finalización de las distintas simulaciones, el *agente evaluador* es responsable de evaluar el diseño resultante. Si el diseño es correcto, la sesión finaliza. En cambio, la sesión puede continuar si el *agente evaluador* detecta errores en el diseño o si decide que los alumnos sean conscientes de otras alternativas de diseño. En este caso, la sesión vuelve al paso 3, donde se volverá a realizar la simulación y las modificaciones necesarias.

Como se puede observar, este patrón es mucho más generalista que el empleado en las sesiones descritas en el Capítulo 4. En él se han incluido una serie de *agentes* que, dependiendo de la instanciación concreta del patrón, podrán ser interpretados por el instructor o por los alumnos. Siguiendo este patrón se pueden crear sesiones de role-play con distintos objetivos pedagógicos. Por ejemplo, se pueden crear sesiones de diseño de una aplicación propuesta por el instructor, en la que los alumnos son responsables de decidir las clases que se van a usar y los escenarios de ejecución que van a ser simulados. Por otro lado, este tipo de sesiones también pueden ser empleadas para la modificación de un diseño aplicando distintas técnicas de refactorización. En este patrón también entran las sesiones que sirven exclusivamente para el análisis y la comprensión de un sistema, de modo que los alumnos observan interactivamente cómo se comportan los objetos que lo componen, no permitiendo que se realice ningún tipo de modificación.

5.2. Ejes de variabilidad del patrón general

Como se ha destacado, este patrón general da cabida a distintos tipos de sesiones. El análisis de estas sesiones y del patrón general nos ha conducido a la detección de una serie de aspectos que son los que añaden la flexibilidad a este patrón para el desarrollo de los distintos tipos de sesiones. Cada uno de estos aspectos funciona como un eje de variabilidad del patrón general propuesto. De esta forma, dependiendo de las decisiones tomadas para cada uno de estos aspectos se definirán sesiones de role-play que siguen la misma metodología pero que usan diferentes aproximaciones de enseñanza —mayor o menor intervención del instructor, mayor interactividad de los alumnos, etc.— y que tienen distintos objetivos pedagógicos.

Los aspectos detectados han sido clasificados de acuerdo a la fase que afectan en el patrón general de la sesión. Las tres fases son: a) presimulación, que incluye las fases de selección y asignación de roles; b) simulación, que incluye el ciclo de simulación y modificación de las clases; y c) evaluación, que hace referencia a la fase de evaluación del diseño resultante. Para cada uno de los aspectos se ha determinado el grado de flexibilidad que tiene. Por último, se ha detectado que existen interrelaciones entre algunos de los aspectos, de modo que las decisiones tomadas en uno de ellos pueden afectar al grado de flexibilidad del que dispone otro aspecto. A continuación pasamos a describir cada uno de los mismos.

5.2.1. Aspectos de presimulación

El primero de los aspectos identificado es la **Selección de escenarios**. Este aspecto define el modo en el que el escenario a simular es elegido. En las sesiones presenciales, lo más normal es que el papel del *agente selector* lo interprete el instructor. En muchas ocasiones existe incluso un orden en el que los escenarios han de ser simulados, ya que la comprensión de alguno de los escenarios pueden depender de la simulación previa de otros escenarios o de las modificaciones producidas durante dichas simulaciones. Otra opción, aunque menos común, sería que el alumno fuese el responsable de decidir qué escenario simular de entre todo un catálogo de los mismos.

Este aspecto también define quién es el responsable de generar la solución inicial y el escenario de ejecución a simular. Si el rol del *agente generador del escenario* lo interpreta el instructor entonces se dará lugar a sesiones de modificación de un diseño y a sesiones de análisis de una aplicación. En cambio, la interpretación del rol del *agente generador del escenario* por parte de los alumnos permite sesiones de diseño mucho más libres, en los que los alumnos parten de un boceto de clases que irán refinando mientras completan los escenarios que ellos mismos decidan simular.

El siguiente aspecto es el de **Asignación de roles**. Este aspecto incumbe la toma de dos decisiones: quién es el *agente asignador de roles* en el patrón

general y qué es lo que ocurre si no hay suficientes alumnos para cubrir todos los roles para el escenario seleccionado.

De acuerdo a la primera decisión, lo más sencillo es que el alumno o el instructor sean responsables de seleccionar el rol que van a interpretar. Si el instructor participa activamente durante esta fase, éste puede decidir los roles que tomarán cada uno de los alumnos de acuerdo a las habilidades que han demostrado durante la simulación de otros escenarios.

La segunda decisión puede ser acometida de manera sencilla no dejando que se pueda simular un escenario mientras que no haya suficiente número de alumnos. Realmente, esta decisión es raramente aplicable en las sesiones presenciales, ya que lo más lógico es que el instructor o los alumnos más capacitados sean los responsables de interpretar los roles que faltan por asignar.

5.2.2. Aspectos de simulación

En primer lugar abordaremos dos aspectos relacionados con el *agente moderador*, el cual podría ser interpretado por el instructor o por algunos de los alumnos más capacitados. El primer aspecto a tratar dentro de la fase de simulación es el **Grado de libertad del alumno**. Este aspecto determina cuándo *agente moderador* toma el control de la sesión para ayudar o corregir los errores del estudiante. Una primera posibilidad son las sesiones libres, en las que no hay ningún *agente moderador* que controle las actividades del alumno. La opción opuesta a ésta es que el *agente moderador* corrija inmediatamente cualquier error que surja durante la sesión de simulación. También existen otro tipo de decisiones intermedias, como permitir que los alumnos cometan errores e intervenir solamente cuando el *agente moderador* detecte que los alumnos no son conscientes de que están siguiendo un camino erróneo.

Otro aspecto que se ve afectado por el anterior es el que se ha llamado **Provisión de ayuda**. Este aspecto define qué es lo que el *agente moderador* hace cuando toma el control de la sesión de simulación. Generalmente, el *agente moderador* proporcionará ayuda a los alumnos cuando se encuentran perdidos o para resolver un error. Este tipo de ayuda puede ser desde decidir cuál es el siguiente paso de simulación hasta pequeñas pistas que hagan pensar al alumno sobre cuál debería ser el paso de simulación correcto. Existen posibilidades más elaboradas, como proporcionar al alumno un diseño similar al que se está resolviendo y su solución. Mediante analogía, el alumno puede inferir cuál ha de ser la solución correcta, adaptando la que se le ha proporcionado al diseño actual.

En segundo lugar, se ha detectado un último aspecto relacionado con esta fase de las sesiones de role-play: **Modificaciones del diseño inicial**. Este aspecto refleja qué tipo de modificaciones se pueden realizar sobre el diseño

inicial. De acuerdo a este aspecto se pueden definir sesiones que sirvan para que el alumno se familiarice y analice el diseño inicial, por lo que no se le permite realizar ningún tipo de modificación. En contraposición a éste, se pueden crear sesiones en las que se permita a los alumnos la realización de modificaciones sobre las clases que conforman el diseño inicial para mejorar y extender dicho diseño. Otro caso sería la definición de sesiones para la creación desde cero una aplicación completa, por lo que los alumnos deberán decidir qué clases van a intervenir en la aplicación y, a medida que simulen escenarios de ejecución del mismo, irán definiendo las responsabilidades y colaboraciones asociadas a dichas clases.

Este aspecto también define los mecanismos y técnicas empleadas para realizar modificaciones del diseño. Las modificaciones pueden ser de lo más variado, desde pequeñas modificaciones que cambien las responsabilidades y colaboradores de una clase, añadan nuevas o eliminen los ya existentes, hasta la aplicación de todo tipo de técnicas de refactorización (Fowler, 1999) que sirvan para modificar el diseño. Por último, al igual que se ha realizado en las sesiones presenciales, esta refactorización puede llegar al nivel de aplicar patrones de diseño para la mejora del diseño propuesto. La aplicación de este tipo de refactorización se puede realizar incrementalmente, como se ha descrito en (Kerievsky, 2004).

Otra característica más relacionada con este aspecto es quién es el responsable de realizar una modificación. La más sencilla sería que el *agente moderador* fuese el responsable final de realizarla. Una decisión más pedagógica sería que ya que los alumnos colaboran en la realización de un diseño, tuviesen que ser capaces de llegar a un consenso antes de la realización de cualquier modificación.

Por último, este aspecto también contempla la decisión sobre cuándo se pueden realizar las modificaciones y los efectos que tienen sobre la ejecución de la simulación. En muchas ocasiones, las modificaiones se pueden realizar durante la simulación, teniendo en cuenta que algunas de las mismas pueden suponer tener que reiniciar la simulación del escenario de ejecución. Otras modificaciones pueden no afectar a los pasos previos de la simulación, por lo que se puede decidir continuar sin necesidad de reiniciar. Finalmente, algunas modificaciones pueden quedar pospuestas hasta la fase de evaluación, como se ha hecho durante las sesiones presenciales descritas en el capítulo anterior, en las que la refactorización basada en patrones sólo se realiza tras la simulación de los escenarios de ejecución, durante la evaluación del diseño inicial.

5.2.3. Aspectos de evaluación

El único aspecto detectado y relacionado con esta fase es el que se ha denominado como **Evaluador**. Este aspecto determina quién es el *agente evaluador* y cómo y cuándo interviene al detectar una mala solución.

En cuanto a la primera decisión, típicamente será el instructor el responsable de la evaluación. Sin embargo, este aspecto incluye la posibilidad de sesiones libres de instructor, en las que los alumnos participantes más aventajados sean los responsables de evaluar su propio diseño. Otra posibilidad sería que alumnos ajenos a la simulación se responsabilizarán de esta evaluación, a modo de revisiones entre iguales o *peer-review* similares a las descritas en (Hsiao y Brusilovsky, 2007). Aunque esta alternativa es completamente válida, consideramos que sería necesaria al menos una mínima intervención del instructor, al menos para valorar las evaluaciones realizadas y para proporcionar consejo y ayuda sobre las mismas.

La segunda decisión puede afectar a algunos de los aspectos anteriormente citados, dependiendo de si la evaluación se realiza durante o al final de la simulación. En el primer caso, este aspecto está relacionado con las decisiones tomadas en los aspectos **Grado de Libertad del alumno** y **Provisión de ayuda** y en la forma en la que el *agente moderador* interviene para corregir errores. Si la evaluación se realiza al finalizar la sesión, el *agente evaluador* expondrá entonces cuáles son los errores detectados, pudiendo acompañarlo de la aportación de una solución apta.

5.3. Transferencia de sesiones de role-play a entornos virtuales

Se ha visto en el Capítulo 3 que las herramientas basadas en visualización interactiva son de gran utilidad en la enseñanza de conceptos relacionados con la orientación a objetos. Estas herramientas simplifican la enseñanza de conceptos abstractos mediante el uso de visualizaciones y mantienen la atención del alumno involucrándolo en mayor o menor medida en el desarrollo de actividades. Sin embargo, la mayoría de estas herramientas dejan a un lado el diseño orientado a objetos y no fomentan la colaboración entre alumnos, algo que es fundamental en el diseño de aplicaciones de gran tamaño.

A pesar de la importancia de las sesiones de role-play, no se ha encontrado trabajo alguno de herramientas que den soporte a estas actividades en el campo de la orientación a objetos. Existen herramientas profesionales que dan soporte al desarrollo de tarjetas CRC, como QuickCRC[1], o que hacen uso de las mismas para buscar componentes software en base a tarjetas CRC, como `merobase` [2]. Pero ninguno de estos entornos da soporte a actividades de role-play y menos a nivel académico. En cambio, existen entornos web que dan soporte a este tipo de actividades para la enseñanza de Historia, como Fablusi (Ip et al., 2001) y TimeScope (Sharf et al., 1999), o Política (Naidu et al., 2000). En estos entornos, los alumnos y el instructor interactúan en

[1] Desarrollado por Excel Software y disponible en `www.excelsoftware.com/quickcrcwin.html` (último acceso: 15-02-2008)

[2] Accesible en `http://www.merobase.com/` (último acceso: 15-02-2008)

el desarrollo de un escenario histórico o político mediante herramientas de chat y foros.

Esta falta de herramientas y el interés despertado por las sesiones de role-play en la enseñanza de orientación a objetos nos ha conducido a plantear el desarrollo de herramientas que den soporte a este tipo de sesiones. Teniendo en cuenta el éxito de las herramientas basadas en la visualización proponemos el desarrollo de entornos que combinen el role-play con facilidades de inmersión del alumno para facilitar el aprendizaje de diseño orientado a objetos.

Para el desarrollo de estas herramientas se propone el traslado de las sesiones presenciales de role-play a un entorno virtual, de modo que los alumnos puedan realizar, desde un ordenador, las mismas actividades que se realizan durante una simulación de role-play. Este tipo de entornos son los que llamaremos *entornos virtuales de role-play*. En estos entornos, el alumno intervendrá, solo o en compañía de otros alumnos, en una sesión de role-play, pudiendo interpretar el papel de uno de los objetos que participan en la simulación. Al igual que en las sesiones presenciales, el entorno proporciona al alumno toda la información necesaria relacionada con el escenario que está siendo simulado. Además, el entorno ha de proporcionar los medios para que los alumnos puedan trabajar colaborativamente en tareas de diseño. Para ello deberá disponer de herramientas de comunicación con otros alumnos y de modificación de las clases que conforman el diseño, en caso de que esto fuera necesario.

5.3.1. ¿Por qué un entorno virtual?

Un entorno virtual de enseñanza es un espacio de información en el que se produce una interacción educativa entre alumnos y el entorno. Este espacio suele tener una representación en 3D aunque no se limitan a este tipo de representación. Es un espacio social, que soporta múltiples aproximaciones pedagógicas y que integra todo tipo de herramientas que fomentan la interactividad de los alumnos (Dillenbourg, 2000).

De entre las herramientas estudiadas en el Capítulo 3, sólo MUPPETS traslada a los alumnos a este tipo de entornos virtuales. Un entorno virtual da soporte a las interacciones sociales entre alumnos, al aprendizaje colaborativo y permite acceder a gran cantidad de información. Los entornos virtuales de enseñanza suponen una mejora *potencial* de la efectividad de la enseñanza debido a los novedosos e interesantes tipos de interacción que proporcionan. Sin embargo esto *no garantiza su efectividad* ya que ésta depende de la audiencia que hace uso de estos entornos y el contexto pedagógico en el que son empleados (Dillenbourg, 2000).

Los entornos virtuales tienen una serie de características que hemos considerado de especial interés para el traslado de las sesiones de role-play a ellos. Estas características son las siguientes (Dillenbourg, 2000):

Espacio de información Los entornos virtuales de enseñanza proporcionan gran cantidad de información de manera estructurada. Generalmente es el propio alumno el responsable de decidir qué información desea ver y cuándo la necesita. Esta información está a disposición del alumno mediante interacciones con elementos del entorno.

Espacio social El usuario tiene una representación de sí mismo dentro del mundo —que se suele denominar avatar— y que le permite interactuar con otras entidades del mundo. Estas entidades pueden ser, a su vez, avatares de otros alumnos, produciéndose interacciones sociales. Generalmente, estos entornos proporcionan mecanismos de interacción social tanto de manera síncrona como asíncrona.

Representación explícita del espacio virtual El entorno proporciona una representación del espacio de información observable por los alumno (que puede ser o no en 3D). El tipo de representación elegida tiene una alto impacto sobre la interacción pedagógica del alumno con el entorno. Los principales factores a tener en cuenta para el diseño de esta representación son la estética, la facilidad de uso y la relación entre la representación espacial y las relaciones estructurales entre la información contenida en el entorno.

Independientes de la localización Los entornos virtuales, como entornos de enseñanza asistida, facilitan que los alumnos puedan progresar en su aprendizaje en cualquier momento, independientemente del profesor y del lugar en el que se encuentren. Sin embargo, no se limitan a la educación a distancia sino que se pueden emplear para enriquecer las clases presenciales con nuevos métodos de enseñanza.

Modifican del rol del alumno El alumno ya no es un individuo pasivo sino que interacciona y participa en actividades dentro del espacio virtual. El alumno es también un actor dentro del entorno, que se siente miembro del mismo y colabora con otros alumnos en la realización de estas actividades.

Integran múltiples herramientas Por defecto, estos entornos suelen proporcionar herramientas para la manipulación y gestión de la información, herramientas de comunicación y para la colaboración entre alumno. Sin embargo, no tienen por qué limitarse a esas. Estos entornos soportan la integración de cualquier otro tipo de herramienta relacionada con el tipo de enseñanza objetivo para la que son diseñados.

Integran elementos del mundo real Los entornos virtuales son espacios a los que se pueden trasladar herramientas y elementos utilizados en las clases presenciales.

5.3.2. Principales características de los entornos virtuales de role-play

El diseño de los entornos virtuales de role-play se ha realizado teniendo en cuenta las dificultades encontradas en las enseñanza de la orientación a objetos y la forma en la que las distintas herramientas vistas en el Capítulo 3 las abordan. También se han tenido en cuenta los deficiencias que se han encontrado en el estudio de dichas herramientas, subsanándolos en la medida de lo posible. Las principales características de las que se propone dotar a los entornos virtuales de role-play son las siguientes:

- *Entornos multifuncionales.* Estos entornos interesa que sean usados tanto para el diseño de nuevas aplicaciones como para la comprensión, revisión y modificación de aplicaciones ya creadas. De hecho, pueden ser también usados como herramientas de visualización, para que el alumno comprenda las interacciones que se producen en un sistema durante un determinado escenario de ejecución.

- *Independientes del lenguaje de programación.* En el entorno se usan conceptos generales de la orientación a objetos como clases, objetos y pasos de mensajes, pero no se emplea una sintaxis propia de un lenguaje de programación. Incluso la interacción entre objetos y el paso de mensajes va envuelto por una metáfora que hace que el alumno se pueda concentrar en los conceptos verdaderamente importantes, en lugar de la sintaxis.

- *Uso de la ámpliamente aceptada metáfora de la antropomorfización.* Al igual que se realiza en el role-play presencial, en los entornos virtuales de role-play el alumno se mete en la piel de un objeto, representado mediante un avatar que interactúa con otros objetos que habitan el entorno.

- *Visualización dinámica e intuitiva de la ejecución del escenario.* Las sesiones de role-play tienen una naturaleza dinámica. En ellas se simulan las interacciones que se producen entre los objetos que conforman un sistema. Por ello hay que proporcionar una metáfora en la que se pueda ver claramente que se está produciendo esta interacción entre los objetos. De nuevo, se hará uso de una metáfora extraída de las simulaciones de role-play para visualizar estas interacciones: una pelota que se va lanzando de un participante a otro como representación del paso de mensajes.

- *Distinción entre clases y objetos.* Se ha puesto un especial hincapié en que este tipo de entornos tengan una clara diferenciación entre clases y objetos, de modo que el usuario del mismo no tenga posibilidad de confundir ambos conceptos.

- *Entornos ricos en información de diseño.* Al igual que en las sesiones presenciales de role-play el alumno debe tener la posibilidad de disponer de gran cantidad de información relacionada con el escenario que está siendo simulado. Dentro del entorno debe estar accesible la información sobre el escenario a simular, el estado del mismo, información sobre las clases que en él intervienen, el estado de los objetos e información sobre el último mensaje que se ha enviado y el estado en el que se encuentra el mismo. Debido a la gran cantidad de información existente, los entornos virtuales de role-play han de proporcionar mecanismos para que el alumno pueda acceder de manera selectiva a toda esta información.

- *Entornos ricos en interacción.* El alumno dispone de un amplio abanico de posibilidades de interacción dentro del entorno, desde el control de la información presentada y el control de la ejecución hasta la comunicación con otros alumnos para trabajar colaborativamente.

- *Uso de ejemplos complejos.* En los entornos virtuales de role-play se han de poder usar los mismos ejemplos que se utilizan en las sesiones presenciales. Esto significa que han de permitir la existencia de múltiples instancias de un mismo objeto y en el que intervengan distintas clases de objetos.

- *Entornos colaborativos.* Los entornos virtuales de role-play pueden emplearse como herramientas de visualización monousuario. Ahora bien, la verdadera potencia de estos entornos se consigue cuando son empleados de la misma manera en la que se desarrollan las sesiones presenciales de role-play, en las que varios alumnos interactúan entre ellos y aprenden unos de otros.

5.4. Necesidades de representación

El traslado de las sesiones de role-play a un entorno virtual requiere que exista una representación visual de los elementos fundamentales que aparecen en las sesiones presenciales. A lo largo de las sesiones presenciales y durante el estudio del patrón general de este tipo de sesiones se ha identificado un conjunto de elementos que son clave en el desarrollo de este tipo de actividades (Jiménez-Díaz et al., 2005a,b,c, 2007a). Para cada uno de estos elementos, se ha decidido proporcionar una representación dentro del entorno virtual. De esta forma, los usuarios del entorno se hacen conscientes de la existencia de estos elementos y, además, pueden interactuar con ellos para la obtención de información o la realización de acciones que les ayuden a completar la tarea de diseño encomendada.

Para cada uno de los elementos fundamentales de las sesiones de role-play

se ha propuesto una representación metafórica. Algunas de las metáforas no son más que el traslado directo al entorno virtual de la metáfora antropomórfica inherente al role-play. Otras tienen su inspiración en los videojuegos del género de aventura gráfica. A continuación se detallan cada uno de los elementos fundamentales de las sesiones de role-play y la representación usada dentro del mundo.

5.4.1. Clases y objetos

El elemento fundamental del role-play son los alumnos y los roles que éstos interpretan. En particular, en estas sesiones de role-play, cada uno de los alumnos interpreta el papel de un objeto durante la ejecución de un escenario. Por tanto, es necesario que los objetos y, así mismo, los alumnos, tengan una representación en el entorno virtual.

Como ya se ha visto a lo largo de varios capítulos, la metáfora de la antropomorfización de los objetos es de gran utilidad en la enseñanza de la orientación a objetos. El role-play es el ejemplo más claro de este tipo de metáfora, donde el alumno se viste con la piel del objeto para comprender su comportamiento. Gracias a la calidad demostrada, el entorno virtual va a conservar dicha metáfora. En el entorno existirán avatares que representarán a cada uno de los objetos que intervienen en la simulación del escenario. Cada uno de estos avatares tendrá una representación antropomórfica y una identidad propia. Los alumnos podrán controlar dichos avatares, interpretando el rol del objeto que representan durante las sesiones de role-play.

Estos avatares también permitirán al alumno moverse por el entorno virtual, explorándolo en busca de más información que le ayude a completar la tarea propuesta. En esta exploración podrá interactuar con otras entidades para obtener dicha información.

Es posible que en ciertos escenarios el alumno se vea obligado a crear nuevos objetos. Por tanto, consideramos que es necesario que exista también una representación para las clases de objetos que puedan existir en el mundo. Como ya se ha visto en los Capítulos 2 y 3, es de suma importancia que el alumno sea capaz de distinguir perfectamente entre clases y objetos, por lo que las representaciones de ambos dentro del entorno han de ser claramente distinguibles.

5.4.2. Paso de mensajes

Una de las bases de la orientación a objetos es la delegación en otros objetos a la hora de completar una responsabilidad. Esta delegación se produce mediante el paso de mensajes entre los objetos que componen el sistema. Como ya se ha visto, esta mecánica aparece perfectamente reflejada en el role-play, de modo que los alumnos se pasan mensajes de unos a otros durante la simulación de un escenario.

En este mecanismo hay dos elementos fundamentales que necesitan de una representación dentro del entorno virtual:

El flujo de control El paso de mensajes supone transferir el control de un objeto a otro. Como ya se comentó en el capítulo anterior algunos autores proponen el uso de una pelota que se va pasando entre los distintos participantes para representar dicho flujo de control. Se ha decidido mantener esta metáfora, de modo que los avatares se irán lanzando la pelota entre ellos para ir avanzando en la ejecución de la simulación del escenario.

El mensaje La pelota usada para transferir el control entre los objetos será la que también hará las veces de mensaje. Como se verá en la siguiente sección, la pelota contendrá toda la información relacionada con el mensaje que se está pasando.

También hay que tener en cuenta que durante la ejecución de un programa orientado a objetos, sólo el objeto que se encuentra actualmente en la cima de la pila de contextos de ejecución puede pasar nuevos mensajes. Debido a la presencia de la pelota, este objeto es perfectamente reconocible dentro del entorno, ya que es aquél que tiene en su poder la pelota. Solamente este objeto podrá construir el siguiente mensaje que se va a pasar y lanzar la pelota para cambiar el objeto activo.

Tampoco hay que olvidar que, por cada paso de mensaje, existe un retorno del mensaje pasado. Por tanto, es necesario que exista algún tipo de distinción entre ambos tipos de paso de mensajes. Se ha decidido que exista una representación distinta para cada uno de ellos. Mientras que el paso de un mensaje se realiza lanzando la pelota por el aire, el retorno de mensajes se representa haciendo que la bola ruede por el suelo hasta el objeto al que hay que devolver el control de la ejecución. De esta sencilla forma, el alumno podrá distinguir claramente qué tipo de mensaje está siendo pasado entre dos objetos.

Por último y siguiendo una de las buenas prácticas descritas en la Sección 3.2, el paso de mensajes va a ir acompañado de explicaciones sobre cuáles son los mensajes pasados, los parámetros reales del mismo y a quién están siendo pasados. Estas explicaciones aparecen sobreimpresas en el entorno a modo de diálogo, similar al que se tiene durante las sesiones presenciales de role-play. Algún ejemplo de este tipo de mensajes son los siguientes:

"*objetoX*, ejecuta *métodoX* usando *valorX* y *objetoY*"

"*métodoX* ha terminado devolviendo *valorY*"

5.4.3. Información del role-play

Cada sesión de role-play necesita distinta información sobre los participantes que intervienen en el escenario simulado. Además, durante la propia

sesión se crea información que permite conocer el momento actual en el que nos encontramos dentro de la simulación. Por tanto, es necesario que cada uno de estos elementos tenga una representación dentro del entorno virtual.

Ya que prácticamente cada una de las entidades que existen en el entorno tendrá algún tipo de información asociada, vamos a hacer que cada una de estas entidades sea responsable de contener dicha información. Para ello, vamos a emplear una metáfora clásica de muchos videojuegos, principalmente de las aventuras gráficas: el inventario. En una aventura gráfica, el personaje interpretado por el jugador explora un mundo por el que va recolectando diferentes artilugios que posteriormente le servirán para la realización de alguna acción. Todos estos objetos se van almacenando en el inventario del jugador. Durante cualquier momento del juego, el jugador puede observar su inventario y obtener información sobre cada uno de los elementos que contiene.

De igual forma, las entidades que pueblan el entorno virtual de role-play dispondrán de un inventario donde almacenarán la información relacionada con ellas. Dependiendo del tipo de entidad, el inventario podrá contener más o menos información. De acuerdo a la información contenida en una sesión de role-play, vamos a distinguir las siguientes entidades con información:

Inventarios de clase Estas entidades contendrán la información relacionada con las clases que intervienen en la simulación, así como información relacionada con la forma en la que se pueden crear objetos de dicha clase.

Inventarios de objeto Estas entidades disponen de información sobre la clase del objeto que representan y del estado actual del objeto.

Inventario de la pelota Esta entidad ha de contener información descriptiva sobre el mensaje que se ha enviado, así como información sobre los parámetros reales con los que ha sido invocado su método asociado. En caso de devolver algún valor, contendrá información sobre el valor devuelto por el mismo, en el momento en el que dicho mensaje ha sido retornado.

Además de éstas y para mantener la coherencia en la forma en la que el alumno puede consultar la información, han de crearse otras entidades que contengan el resto de información relacionada con el role-play:

- Una descripción sobre el escenario que está siendo simulado. La interacción con este entorno virtual tiene un fin y la descripción de éste ha de estar presente dentro del entorno. Se propone que exista una entidad cuyo inventario contenga información textual sobre el escenario que está siendo simulado o diagramas con las relaciones estáticas entre las clases que intervienen en el escenario. Incluso podría ser de especial

interés que el alumno fuera capaz de observar cuál es el comportamiento final del ejemplo de aplicación que está siendo usado al ejecutar el escenario que se propone simular.

- Un histórico de la simulación. Durante las sesiones presenciales de role-play, el instructor iba creando un RPD con los pasos de ejecución que han sido completadas. Por tanto, se propone que el alumno disponga de una entidad dentro del entorno sobre la que pueda consultar el estado actual en el que se encuentra la simulación. Este histórico de la simulación puede ser representado haciendo uso de un RPD o de cualquier otro diagrama que permita observar el flujo de control que se ha producido hasta el momento.

5.5. Necesidades de interacción

Una de las principales conclusiones obtenidas del Capítulo 3 es que de poco sirven los entornos basados en visualizaciones si el alumno no es más que un observador pasivo de las mismas. Además, los resultados obtenidos en los experimentos descritos en el Capítulo 4 han demostrado que los alumnos que son meros observadores de la interacción entre sus compañeros obtienen peores resultados que los que participan en la tarea activamente.

Los entornos virtuales basados en role-play han de ser, por tanto, ricos en interacción con el alumno. De este modo, conseguiremos involucrarle y que se sienta partícipe en su propio proceso de enseñanza. La actividad en sí misma ya hace que el alumno sea partícipe: ha de interpretar el rol de un objeto en un escenario de ejecución. Como se verá más adelante, este tipo de entornos pueden también ser usados para alumnos noveles que realmente no interpretarán dicho rol sino que observarán la ejecución. Especialmente para este tipo de uso hay que proporcionar al alumno mecánicas adicionales que le permitan interactuar con las entidades del entorno, extraer la información del mismo o tomar decisiones puntuales de ejecución, entre otros. Estas mecánicas se han descrito en la Sección 5.5.1. En (Jiménez-Díaz et al., 2005a,b,c, 2007a) se puede ver un resumen de las mecánicas elementales del role-play, mientras que en (Jiménez-Díaz et al., 2007c,d) también se incluyen las mecánicas relacionadas con la modificación de la información del escenario y las acciones colaborativas. En la Sección 5.5.2 se plantean los distintos niveles de participación del alumno que los entornos virtuales de role-play fomentan.

5.5.1. Mecánicas y acciones

5.5.1.1. Mirar a

Esta es la principal forma de interacción entre el alumno y el entorno virtual. Una de las primeras tareas que el alumno ha de hacer dentro del entorno es familiarizarse con las entidades que hay en el mismo, al igual que se hace con las tarjetas CRC antes de una sesión de role-play presencial. El alumno explora el entorno moviéndose con su avatar y consultando la información de las entidades que aparecen en él mediante la acción *Mirar a*. Esta acción es también, al igual que el inventario, típica de las aventuras gráficas. Mediante esta acción, el alumno puede visualizar parte o todo el inventario de la entidad que tiene delante de su avatar. La información presentada variará de acuerdo a lo descrito en la sección anterior. Por ejemplo, mientras que la acción *Mirar a* un objeto permite observar su estado y la información sobre la clase a la que pertenece, *Mirar a* la pelota presentará el inventario de la misma, con información sobre el último mensaje enviado.

Esta acción también ha de poder ser aplicada sobre el propio avatar del alumno, de modo que pueda explorar el inventario del objeto que está representando. De esta forma conocerá cuáles son las responsabilidades de su objeto, que otras clases conoce o cuál es el estado en el que dicho objeto se encuentra actualmente.

5.5.1.2. Paso de mensajes

Esta mecánica es usada por un avatar para transferir el control de ejecución de un objeto a otro. Esta mecánica se puede dividir en tres fases:

1. **Creación del mensaje.** Consiste en crear y configurar el mensaje que va a ser pasado. El invocador del mensaje está fijado ya que es el objeto que actualmente tiene el control de la ejecución. En cambio, es necesario decidir el tipo de mensaje (invocación o retorno), cuál es el receptor del mensaje, cuál es el método invocado con el paso de dicho mensaje, cuáles son los parámetros reales del método y, en caso de que el tipo de mensaje sea de retorno, decidir qué es lo que va a ser devuelto por dicho mensaje.

2. **Lanzar la pelota.** Una vez que el mensaje ha sido creado se debe realizar la transferencia del control. Para ello, el avatar ha de ser capaz de lanzar (o hacer rodar) la pelota en dirección al objeto receptor del mensaje.

3. **Recoger la pelota.** Para concluir la transferencia, es necesario que el receptor del mensaje coja la pelota. Una vez que ésta ha sido recogida, el objeto que la sostiene es el nuevo objeto activo.

El paso de mensajes no está exento de restricciones. El entorno virtual ha de ser responsable de comprobar que dichas restricciones son satisfechas. Por ejemplo, el objeto receptor de un mensaje es el único que ha de poder recoger la pelota en el momento que ésta ha sido lanzada. El resto de objetos no pueden recogerla ya que no son destinatarios del mensaje que contiene. También pueden existir restricciones entre los objetos a los que un mensaje puede ser enviado. De acuerdo a la información contenida en una tarjeta CRC, un objeto sólo conoce a sus clases colaboradoras. Por tanto, sólo puede enviar mensajes a objetos que pertenezcan a su lista de colaboradores.

También consideramos como un paso especial de mensajes aquél que se usa para la construcción de un objeto. En este caso, el destinatario no es un objeto sino que ha de ser una clase. El método que va a ser invocado con dicho paso de mensaje será uno de los constructores de la clase. Este tipo de mensaje también necesita que el nombre del objeto creado sea especificado. Una vez construido este tipo de mensaje, la bola será lanzada hacia una entidad que represente una clase. Tras esto, en el entorno virtual ha de aparecer un nuevo avatar que represente al nuevo objeto creado.

5.5.1.3. Control de la ejecución

Como se ha podido ver en el Capítulo 3, muchas de las herramientas de visualización disponen de acciones que permiten al usuario controlar la ejecución de la simulación. Del mismo modo, los entornos virtuales de role-play han de disponer de acciones similares. En caso de que estos entornos sean usados como meras herramientas de simulación, se ha de poder permitir, al menos, hacer avanzar y retroceder la simulación del escenario. Hay que tener en cuenta que cada uno de los pasos de ejecución impone la modificación del RPD asociado al escenario. También es posible que el estado de alguno de los objetos del escenario sea modificado, por lo que estos cambios han de quedar registrados en el entorno.

En el caso de usar estos entornos como medio para el diseño colaborativo, es necesario que se puedan deshacer acciones. En un determinado momento, un alumno puede enviar un mensaje. Si se hace consciente de que ha cometido un error, entonces el envío de este mensaje ha de poder deshacerse. De nuevo, deshacer una acción implica que el RPD ha de ser modificado, el estado de algunos objetos ha de ser restaurado y la pelota ha de volver al objeto que envió el mensaje erróneamente.

Por último, si el entorno se usa como medio para revisar y modificar un diseño dado, es necesario que se pueda decidir cuándo se considera que la simulación ha concluido. En otro caso, es el propio entorno el que conoce cuándo se puede dar por concluida la ejecución del escenario.

5.5.1.4. Modificación de información del escenario

La revisión de un diseño mediante el uso de actividades de role-play puede inducir a la modificación de las clases de los objetos que participan. Debido a esto, los entornos virtuales de role-play han de proporcionar herramientas que faciliten la modificación de las clases presentes en un determinado entorno. Como ya se ha visto, estas modificaciones pueden ser de distinto tipo: desde la modificación de los métodos hasta la aplicación de un patrón de diseño. La ejecución de estas herramientas puede provocar que el escenario sea reiniciado o que sea necesaria la reevaluación del mismo para ver si se puede continuar con su ejecución de acuerdo a los cambios realizados sin necesidad de volver a empezar de nuevo.

Existe otro tipo de modificación más simple y que puede ser dejada en manos del alumno: la modificación del estado del objeto. Mediante esta acción, el alumno es responsable de modificar el estado de un objeto (si procede) en el momento en el que recibe un mensaje. En este caso, el alumno ha de interpretar el mensaje que ha recibido, los parámetros con los que ha sido invocado y modificar convenientemente el estado del objeto que representa, contenido en su inventario.

5.5.1.5. Acciones colaborativas

Una de las principales virtudes de los entornos virtuales de role-play propuestos es que el instructor y los alumnos, y los alumnos entre sí, puedan colaborar en la realización de una actividad de diseño. Por tanto, se han de proporcionar herramientas que les permitan comunicarse dentro del entorno. Ya que el entorno es en tiempo real, sería recomendable que las herramientas proporcionadas fueran de comunicación síncrona, como chat o sistemas de comunicación a través de un micrófono y cascos.

Además, hay que tener en cuenta que hay acciones descritas en las anteriores secciones que podrían ser realizadas de manera unilateral por un alumno, pudiendo producir conflictos entre los participantes. Por este motivo es conveniente que estos entornos dispongan de herramientas de consenso ante decisiones críticas como, por ejemplo, deshacer una acción, dar por finalizado un escenario o modificar una clase. Estas herramientas de consenso pueden estar soportadas por un sistema de votación: el resto de alumnos han de votar si se realiza o no la acción propuesta. Finalmente la acción puede ser realizada por mayoría o por unanimidad, dependiendo de lo que se decida durante el diseño de este tipo de herramientas.

Debido a la ubicuidad que proporciona este tipo de entornos, el uso de estas herramientas pueden provocar situaciones en las que el entorno quede bloqueado como, por ejemplo, si se espera a que todos los alumnos emitan su voto y por fallos en la red o por ausencia, alguno de los alumnos no emite dicho voto. Para evitar esto se propone que este tipo de herramientas tengan

un tiempo limitado de acción y que, en última instancia, el propio alumno que ha generado la acción conflictiva sea el responsable de decidir si se lleva a cabo o no, aún a riesgo de ir en contra de la mayoría.

También se considera necesario que, de cara al instructor y a la evaluación de las actividades realizadas, las acciones ejecutadas así como las conversaciones que se han producido en estos entornos colaborativos sean guardadas en una bitácora o *log* de la ejecución del escenario. Este *log* será especialmente útil para observar el uso correcto de las herramientas de consenso, ya que en él quedará almacenado si, por ejemplo, un alumno ha tomado una decisión de diseño en última instancia a título personal y sin hacer caso de la mayoría o si, por el contrario, lo ha realizado por falta de unanimidad.

5.5.2. Niveles de participación

La interacción entre el alumno y el entorno virtual es de vital importancia para la utilidad de este tipo de herramientas como medio de enseñanza. Todos los niveles de participación descritos en la Sección 3.1 resultan interesantes para involucrar al alumno en el proceso de enseñanza. Por eso, las mecánicas descritas anteriormente han sido pensadas para que los entornos virtuales de role-play permitan involucrar al alumno en cada uno de los niveles de participación propuestos. A continuación se analizan las mecánicas que dan soporte a cada uno de los niveles.

5.5.2.1. Visionado

Como ya se ha descrito, este nivel propone que, entre otros, el alumno tenga el control de las distintas vistas, así como de la ejecución de la visualización. La acción *Mirar a* es la responsable de que el alumno interactúe con el entorno para seleccionar la información que visualiza en cada momento.

Como es obvio, las mecánicas de control de la ejecución también dan soporte a este nivel de participación. Pero hay que destacar que el paso de mensajes es un medio de control de la ejecución que involucra especialmente al alumno con el entorno. Este mecanismo no es un simple "paso adelante" de una herramienta de visualización sino que hace responsable al alumno de decidir cuál es *exactamente* el siguiente paso que ha de ser ejecutado.

5.5.2.2. Respuesta

En este nivel se propone que el alumno responda a preguntas relacionadas con la visualización a medida que ésta se desarrolla. En los entornos virtuales de role-play, este nivel de interacción está implícito en la mecánica del paso de mensajes. Cuando el alumno recibe un mensaje de otro objeto, se le está pidiendo que decida cuál es el siguiente paso a dar en la ejecución de la visualización. Además, esta pregunta es crítica ya que, mientras que no la

responda, la visualización permanecerá detenida.

En el caso de que el alumno no interprete ningún rol, esta misma mecánica puede ser usada para involucrar al alumno. La simulación puede ser detenida durante su ejecución y el entorno puede solicitar al alumno que, de acuerdo al objeto activo actual, se ponga en su piel y decida cuál es el siguiente paso de mensaje que se ha de realizar. Aunque esta pregunta, al igual que antes, puede ser crítica y detener la ejecución de la simulación mientras no sea respondida correctamente, también puede ser usada para que el alumno evalúe sus expectativas, comparando el paso de mensaje que él había supuesto que se iba a producir con el que realmente se ha producido.

5.5.2.3. Cambio y construcción

Al igual que en el Capítulo 3, la diferencia entre estos dos tipos de participación en estos entornos es muy pequeña. Durante la sesión de role-play, los alumnos se responsabilizan de la ejecución de un escenario de uso. Del mismo modo, los alumnos que se ponen en el papel de un objeto dentro de un entorno virtual de role-play deciden cómo se produce la ejecución de la visualización dentro del propio entorno. Además, acciones de control de la ejecución como la de deshacer un paso de mensajes les permite cambiar la ejecución y experimentar, respondiéndose a preguntas del tipo "¿qué ocurriría si, en lugar de esto que hice, pasase este otro mensaje?".

Por otro lado, las modificaciones de las clases y del estado de los objetos también permiten alterar la ejecución de un escenario. Por ejemplo, se puede observar cómo el cambio del estado de un objeto produce un comportamiento distinto o una modificación en el estado de otros objetos.

5.5.2.4. Presentación

Como ya se destacó en el Capítulo 3, este nivel de participación es especialmente interesante pero no suele ser incorporado por las herramientas de visualización para la enseñanza de orientación a objetos. En cambio, los entornos virtuales de role-play promueven la discusión de los alumnos dentro del propio entorno. Para ello, como ya se ha indicado, este tipo de entornos deben incluir toda clase de herramientas colaborativas con las que los alumnos puedan comunicarse y llegar a acuerdos.

Además, gracias a la información contenida en la bitácora, un escenario simulado por un grupo de alumnos podría ser reproducido posteriormente para que un instructor evalúe la actividad realizada. Igualmente, como se indicó al hablar del aspecto *Evaluador* al final de la Sección 5.2, esta evaluación puede ser realizada por otros alumnos, de modo que la información contenida en la bitácora permita a los alumnos evaluadores criticar y discutir el diseño al que han llegado otros alumnos.

5.6. Necesidades de autoría

Al igual que las sesiones de role-play tienen una gran carga de creación de material por parte del instructor, los entornos virtuales de role-play también necesitan del desarrollo de una gran cantidad de información para cada escenario que se desea simular haciendo uso de estos entornos.

Para empezar, es necesario, al igual que en las sesiones presenciales, decidir cuál es el caso de estudio que se va a emplear durante las sesiones de role-play. Para este caso de estudio, hay que definir cuáles son las clases que lo componen y cuáles son las relaciones existentes entre estas clases. Como se ha visto, las tarjetas CRC son el medio de representación de las clases en las sesiones presenciales de role-play. Pero para los entornos virtuales de role-play, la información contenida en estas tarjetas ha de ser ampliada. Mientras que en las clases presenciales es el instructor quien describe las clases y las responsabilidades de cada clase, dentro del entorno es necesario que cada clase incluya su propia descripción y que cada responsabilidad esté acompañada de una descripción del objeto de la misma. De igual modo, los parámetros que acompañan a estas responsabilidades han de ir comentados, de modo que el alumno sepa cómo hacer uso de estas responsabilidades en el paso de mensajes.

Una vez que se dispone de la estructura estática del caso de estudio, es necesario definir cada uno de los escenarios de ejecución que van a ser simulados. Para cada escenario hay que proporcionar una descripción lo más detallada posible del mismo. En las clases presenciales se suele hacer uso del propio caso de estudio para mostrar el escenario que se desea simular. Para los entornos virtuales de role-play hay que incluir información que pueda facilitar la comprensión del escenario, tales como diagramas de la arquitectura de clases de la aplicación, una descripción textual del escenario de ejecución, vídeos de la aplicación en ejecución, etc.

Además, es fundamental la configuración inicial de cada escenario de ejecución. En esta configuración inicial se ha de incluir información sobre cuáles son los objetos que van a participar en la simulación, la clase a la que cada uno de ellos pertenece y el estado inicial en el que que se encuentran. Este estado puede ser una simplificación del estado real del objeto ya que para cada escenario pueden ser relevantes sólo un subconjunto de todos los atributos que definen el estado completo de un objeto.

También es necesario establecer una configuración física inicial del escenario dentro del entorno. Es decir, hay que proporcionar al entorno información sobre la ubicación y la orientación inicial de cada una de las entidades que en él existen. Esto es lo que en el mundo de los videojuegos se conoce como *mapa del nivel*. Por tanto, los entornos virtuales de role-play también necesitarán un mapa del escenario que va a ser simulado.

La información descrita hasta ahora sería suficiente para el desarrollo de

escenarios en los que, como en las clases presenciales, exista un instructor encargado de que la simulación llegue a buen puerto. También es suficiente para escenarios sin instructor, en los que los alumnos son libres de hacer lo que quieran dentro del entorno virtual de role-play. En cambio, si se desea que el entorno guíe parcial o totalmente el desarrollo de la simulación, es necesario que se desarrolle información sobre una o varias soluciones de ejecución del escenario. Estas soluciones han de contener información sobre la secuencia de pasos de mensaje que se han de producir para completar el escenario de ejecución. Si el entorno virtual no dispone de un simulador que interprete este paso de mensajes y que actualice convenientemente el estado de los objetos, entonces es necesario que la solución también incluya, para cada paso de mensaje, las modificaciones producidas por el mensaje invocado en los estados de los objetos.

Hay que tener en cuenta que la información sobre la solución ha de estar codificada de modo que pueda ser comparable con los pasos de ejecución que los alumnos realizan dentro del entorno virtual de role-play. De este modo se podrá conocer si los alumnos van por buen camino o si, por el contrario, están cometiendo errores. La elaboración de estas soluciones también puede ayudar a la automatización de las tareas de objetos que no son interesantes para ser interpretados por alumnos. Si se proporciona la solución, cuando el alumno pasa un mensaje a un objeto sin alumno asignado, éste podrá responder haciendo uso de la información contenida en la solución. Sin embargo, será necesaria mayor cantidad de información si se desea que estos objetos sean capaces de responder aunque existan errores durante la ejecución del escenario.

Como se verá en el próximo capítulo, es necesario proveer a los instructores que hagan uso de estos entornos virtuales de role-play de herramientas que ayuden a la construcción de toda esta información. Estas herramientas han de ser de fácil uso para el instructor y han de componer la información de manera que sea inteligible para el entorno virtual. Mediante estas herramientas, el instructor ha de poder crear su propio caso de estudio, así como los escenarios de ejecución del mismo que desee simular. El instructor también ha de tener herramientas para la creación de las soluciones (si hay más de una) de un escenario de ejecución.

Teniendo en cuenta que tenemos a nuestra disposición casos de estudio que han sido desarrollados con fines específicamente pedagógicos, como los descritos en la Sección 2.4.2, se ha de estudiar también el desarrollo de herramientas con las que construir escenarios para entornos virtuales de role-play basados en estos casos de estudio. Para ello se pueden crear herramientas que extraigan la información de las clases del caso de estudio y que la transformen en la información de las clases que aparezcan en el entorno virtual. Estas herramientas se aprovechan del objetivo pedagógico del caso de estudio, que hace que estas aplicaciones tengan una gran cantidad

de documentación de desarrollo. Por otro lado, estos casos de estudio suelen ir acompañados de su código fuente y de algunos ejemplos de ejecución. Por tanto, estas herramientas pueden hacer uso de ambos para la generación automática o semiautomática de los escenarios de ejecución y de las soluciones a los mismos.

5.7. Variaciones de los entornos virtuales de role-play

En la Sección 5.2 se analizaron un conjunto de aspectos que generaban variaciones del patrón general de las sesiones presenciales de role-play. Estos mismos aspectos influyen en el desarrollo de un entorno virtual de role-play ya que las decisiones tomadas para cada uno de ellos afecta al diseño del entorno virtual.

Gracias a la identificación de estos aspectos podemos definir no solo un entorno virtual de role-play específico, sino toda una familia de este tipo de entornos. Cada aspecto define un eje de variabilidad de los entornos virtuales de role-play. Las decisiones tomadas para cada uno de estos aspectos define la manera en la que un entorno concreto será diseñado. A continuación se detalla la influencia de cada uno de los aspectos en las decisiones tomadas a la hora de diseñar un entorno virtual de role-play particular. Un resumen de estos aspectos aparece en (Jiménez-Díaz et al., 2007b).

5.7.1. Decisiones sobre aspectos de presimulación

El aspecto **Selección de escenarios** obliga a decidir quién y de qué forma se selecciona el escenario de ejecución antes de entrar al entorno. La opción más sencilla es que el entorno sea el agente selector que elija automáticamente de manera aleatoria el escenario que va a ser simulado. Otras opciones podrían ser seguir una secuenciación de los escenarios —un escenario no puede ser seleccionado mientras que el anterior no se ha completado correctamente—, la selección en base a perfiles de alumnos o por filtrado colaborativo, entre otros. Otra opción es que el alumno o el instructor humano, si existe en el entorno, interpreten el rol del agente selector, escogiendo libremente el escenario a simular.

La decisión tomada sobre respecto quién interpretará el rol del agente generador de solución y escenarios de ejecución no afecta tanto a las decisiones del entorno sino más bien a las tareas de autoría. Los escenarios serán creados con las mismas herramientas de autoría. Sin embargo, será responsabilidad del instructor o de los alumnos la generación de la información asociada a estos escenarios.

La primera decisión que se ha de tener en cuenta dentro del aspecto **Asignación de roles** está relacionada con quién es el responsable de asignar

los roles de un escenario. Lo más sencillo es que el alumno o el instructor sean responsables de seleccionar el rol que van a interpretar. En el caso de hacer uso de sistemas inteligentes de tutoría, éstos pueden emplear el modelo del estudiante para elegir el rol que más se parezca a su perfil.

Este aspecto también conlleva tomar una segunda decisión de diseño, relacionada con qué es lo que se ha de hacer cuando el número de alumnos usuarios de la herramienta es menor que los roles interpretados. Este aspecto puede ser acometido de manera muy sencilla, no dejando que se pueda simular un escenario mientras que no haya suficiente número de alumnos. Por otro lado, al igual que en las sesiones presenciales, se puede proporcionar al instructor, o a cualquier otro que haga el papel de agente moderador, una interfaz con la que él sea responsable del control de los objetos que no han sido cubiertos por estudiantes, en el caso de que sea contemplado el uso de un instructor humano. La solución de mayor complejidad es la que promueve que los roles no cubiertos por estudiantes sean simulados por agentes virtuales dentro del entorno. Diseñar el comportamiento de estos agentes puede no ser nada sencillo de acuerdo al grado de libertad que se dé a los alumnos al ejecutar la simulación, afectando enormemente al desarrollo de los escenarios que se pueden simular.

5.7.2. Decisiones sobre aspectos de simulación

La primera decisión a tomar para este aspecto es quién va a ser el responsable de adoptar el papel del agente moderador. Este rol podría ser interpretado por un instructor humano, por alguno de los alumnos o por algún sistema de tutoría inteligente.

Una vez tomada esta decisión se pasará al aspecto **Grado de libertad del alumno**, que determina el momento en el que el agente moderador toma el control de la ejecución de cara a la corrección de las acciones erróneas del alumno. La decisión más simple es la creación de entornos libres en los que no existe ningún tipo de control. En el caso de que sea el entorno o un sistema de tutoría inteligente los que interpreten este rol, la solución diametralmente opuesta es la que corrige al alumno en el mismo momento en el que se comete el error. En este caso, cuando un alumno ejecuta un paso de ejecución que no se corresponde con el siguiente de acuerdo a la solución que tiene el entorno, éste interviene dejando al estudiante que modifique la última acción realizada. Obviamente, esta decisión obliga a que todo escenario incluya información sobre, al menos, una solución sobre la simulación correcta del mismo. Entre estos dos extremos existe toda una gama de posibilidades, dependientes del nivel de complejidad que se quiera añadir al sistema de tutoría encargado del agente moderador.

De cara al aspecto **Provisión de ayuda**, y si el aspecto anterior contempla la posibilidad de que el entorno intervenga ayudando al alumno, aquél puede proporcionar el siguiente paso de ejecución correcto. En lugar de pro-

porcionar una solución tan directa, se pueden elaborar pistas a partir de la solución al escenario simulado, de modo que el alumno tenga que pensar cuál es la acción correcta, en lugar de proporcionarle directamente la solución.

Especialmente interesante resulta la posibilidad de proporcionar diseños similares para que el alumno adapte la solución de éstos a su propio problema. Esta decisión relacionaría el diseño de entornos virtuales de role-play con la enseñanza basada en casos (Kolodner, 1997; Jiménez-Díaz y Gómez-Albarrán, 2005). Este tipo de entornos se basan en disponer de una base de escenarios de simulación con sus respectivas soluciones. Esta base estará indexada convenientemente, de modo que en cualquier momento se puedan recuperar escenarios similares al que está siendo actualmente simulado. Con este caso recuperado, el alumno tendrá que inferir la solución a su escenario, generalmente adaptando la que acompaña al caso recuperado. Los propios escenarios resueltos por los alumnos pueden pasar a formar parte de la base de casos, pudiendo ser empleados posteriormente como medio de ayuda. Esta propuesta de provisión de ayuda nos es familiar, ya que dentro de nuestra trayectoria investigadora hemos estudiado la aplicación de la enseñanza basada en casos para comprender el funcionamiento de frameworks (Jiménez-Díaz y Gómez-Albarrán, 2004; Jiménez-Díaz et al., 2004a,b).

La decisión tomada de cara a este aspecto, influirá profundamente en el diseño del sistema de tutoría que se encargará de realizar este control de la ejecución. Por contra, si se determina la existencia de un instructor humano, entonces el entorno ha de estar preparado para proporcionar al instructor los medios con los que aportar la ayuda a los alumnos. Generalmente, las herramientas de comunicación serán suficientes. También se le puede proporcionar otros medios como bases de ejercicios para proporcionar a los alumnos escenarios similares y que deduzcan así el siguiente paso de ejecución, o una interfaz con la que tomar el control de la ejecución, para que los alumnos vean cuál es el siguiente paso de ejecución a realizar.

El aspecto **Modificaciones del diseño inicial** influye enormemente sobre las herramientas que se han de integrar en el entorno. Estas herramientas han de dar soporte al tipo de modificación que se pretende realizar dentro del entorno. Algunas herramientas han de permitir crear diseños completos desde cero. Otras, en cambio, han de ser capaces de realizar modificaciones sencillas sobre las clases ya proporcionadas. También hay que contemplar la posibilidad de herramientas más complejas para la instanciación de patrones de diseño, en el caso de que se dé soporte a la refactorización basada en patrones.

Este aspecto también influye en las herramientas colaborativas ya que determina la existencia o no de herramientas de toma de decisiones consensuadas. Si solamente el instructor o un alumno, de manera unilateral, pueden realizar cambios, entonces estas herramientas no serán necesarias. En caso de necesitar consenso, además del desarrollo de este tipo de herramientas,

será necesario decidir cómo romper las situaciones en las que los alumnos no alcancen dicho consenso, ya sea por problemas técnicos o por no ser capaces de llegar a una decisión común.

Por último, si las modificaciones pueden realizarse durante la simulación, es necesario decidir si el entorno va a reiniciar o no la simulación en curso. Una solución muy simple es que toda modificación provoque irrevocablemente el reinicio del escenario. Esto puede hacer que un escenario se vuelva demasiado largo, repitiendo una y otra vez los mismos pasos, por lo que se pueden adoptar soluciones de mayor complejidad, como que el entorno sea capaz de comprobar si las modificaciones realizadas afectan a los pasos ya dados. Sólo en caso de que esto se produzca, se reiniciará la ejecución. En otro caso, el entorno permitirá que los alumnos sigan con la ejecución del escenario.

5.7.3. Decisiones sobre aspectos de evaluación

El aspecto **Evaluador** determinará quién y cuándo se realiza la evaluación del escenario simulado. En el caso de que la evaluación se realice durante la simulación del escenario, las decisiones aquí tomadas serán similares a las realizadas en los aspectos de Grado de libertad del alumno y de Provisión de ayuda. Si el agente evaluador es humano, entonces también deberá proporcionarse para él una interfaz con funcionalidad adicional que le permita intervenir durante la simulación y proporcionar ayuda ante soluciones de diseño incorrectas.

Si la evaluación se realizara tras la simulación entonces lo primero que habrá que determinar es el nivel de detalle de la información contenida en la bitácora generada al concluir la simulación. Esta información ha de ser interpretada por el agente evaluador, por lo que es muy probable que no pueda ser la misma dependiendo de si este rol lo interpreta un humano o una máquina.

En caso de que el agente evaluador sea interpretado por un instructor humano o por los alumnos, es necesario que el entorno virtual disponga de una interfaz de evaluación, de modo que la solución de la sesión de role-play pueda ser visualizada de nuevo. Esta interfaz también será de extrema utilidad si se propone que sean los alumnos quienes evalúen las soluciones y discutan sobre las mismas.

Las decisiones más complejas se tomarán en el caso de que se decida que el agente evaluador sea un sistema inteligente de tutoría. En este caso, el sistema ha de tener la capacidad de comparar soluciones. También se le podrá hacer responsable de proponer soluciones de diseño correctas al escenario —lo que provoca que sea necesaria la generación de dichas soluciones. Especialmente interesante resultaría que el sistema pudiese comparar la solución correcta con la proporcionada por los alumnos y realizar la comparación de

ambas, utilizando estrategias de similitud y de adaptación similares a las que se pueden realizar en sistemas de enseñanza basada en casos.

5.8. Arquitectura de los entornos virtuales de role-play

Como se ha podido ver en la sección anterior, son muchas las decisiones que se pueden tomar a la hora de diseñar entornos virtuales de role-play. A pesar de estas decisiones, existen partes que son comunes a todos los entornos, por lo que no sería necesario crearlas cada vez que se desea diseñar una nueva herramienta de este tipo. En esta sección presentamos una propuesta de arquitectura de alto nivel que sirve como soporte para entornos virtuales de role-play. Esta arquitectura se compone de un conjunto de módulos lógicos que, posteriormente, han de ser implementados empleando una determinada tecnología. La arquitectura distingue claramente los módulos comunes de los configurables.

La arquitectura ha sido diseñada en base a nuestra experiencia en el desarrollo de videojuegos. Se ha tomado como partida el modelo de arquitectura descrito en (McShaffry, 2005) y que suele ser usado, con ligeras variaciones, en el desarrollo industrial de videojuegos y simuladores. Como se puede ver en la Figura 5.2, esta arquitectura se compone de tres módulos fundamentales:

- Capa de aplicación. Es la responsable del control de ejecución del videojuego. En ella se realiza la inicialización y la finalización del mismo y contiene el bucle principal de ejecución. Esta capa, además, proporciona interfaces para todas aquellas funcionalidades que son dependientes de la máquina en la que va a ser ejecutado, como dispositivos de entrada, sistemas de archivos, relojes de sistema o entornos de red, entre otros.

- Capa de lógica. Esta capa es la que contiene la lógica del videojuego en sí mismo. Contiene todas las entidades del juego y es responsable de actualizarlas convenientemente de acuerdo a las acciones y eventos que ocurran durante la ejecución del juego en función del estado en el que éste se encuentre. En ella se encuentran todas las reglas del juego.

- Vistas. Una vista es un medio bidireccional de comunicación con la lógica. La más común es la vista del usuario del juego, que es la responsable de dos funciones principales: traducir las acciones que el usuario realiza mediante los dispositivos de entrada (teclado, ratón, joypad...) a acciones que la lógica es capaz de interpretar y ejecutar sobre las entidades; y proporcionar una representación de lo que ocurre en la capa de lógica, generalmente mediante audio e imágenes por pantalla.

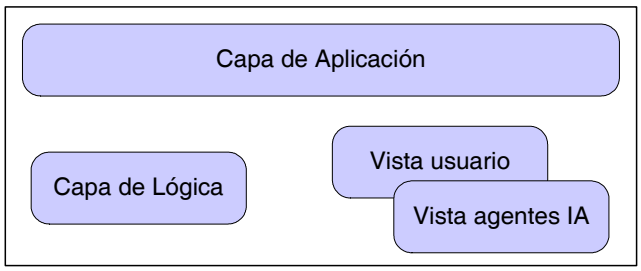


Figura 5.2: Arquitectura para el desarrollo de videojuegos. Extraída de McShaffry (2005).

> Esta no es la única vista ya que, por ejemplo, es posible crear vistas para que un agente pueda realizar acciones en la capa de lógica y para que perciba las acciones que se produzcan en el mundo, y así tomar decisiones sobre las siguientes acciones a realizar.

Una de las principales ventajas de esta arquitectura es que la capa de lógica es completamente independiente de las otras dos capas, por lo que basta con programarla una única vez. Cualquier cambio en la plataforma para la que el videojuego es desarrollado, la forma de interacción del usuario o su aspecto visual es totalmente transparente a la capa de lógica. Este aislamiento de la lógica se consigue de la siguiente forma:

- Cualquier módulo que desee comunicarse con la lógica ha de hacerlo mediante una interfaz de comandos que ésta proporciona. De esta forma, la lógica no necesita saber quién se está intentado comunicar con ella ni el protocolo específico que sigue. Para ello, cada una de las vistas dispone de un módulo responsable de convertir las acciones propias de la vista a comandos que la lógica entiende. Por ejemplo, la pulsación de una tecla en la vista del usuario es traducido al comando *avanzar* de la lógica, que mueve el avatar que representa al jugador.

- La capa de lógica emite eventos de todo aquello que ocurre en el mundo y que considera que ha de ser conocido por otras capas. Si una capa está interesada en estos eventos, se registrará como observador de la lógica. De esta forma, la lógica no necesita saber cómo ha de comunicarse con otras capas sino que dichas capas son las responsables de interpretar los eventos de la lógica y de actualizarse convenientemente. Por ejemplo, si la lógica actualiza la posición del avatar del jugador, emite un evento que avisa del cambio de posición del mismo. La vista del usuario será responsable de presentar por pantalla convenientemente este cambio de posición. También la vista de un agente que representa un enemigo puede recibir este evento e interpretarlo para decidir cuál es la siguiente

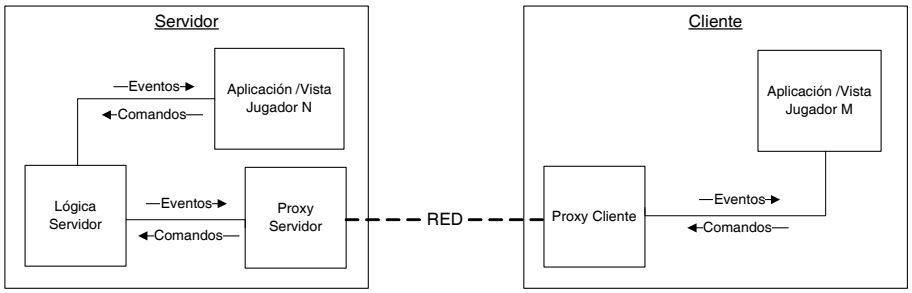


Figura 5.3: Arquitectura cliente-servidor para videojuegos.

acción a realizar para atacar a un jugador.

Además, esta arquitectura es fácilmente extensible gracias a la creación de nuevas vistas sin necesidad de modificar la lógica subyacente. También hay que destacar que es independiente del número de usuarios del videojuego y de dónde se encuentran los mismos, gracias a la forma en la que se comunican con la lógica.

Otra característica importante que nos ha llevado a decantarnos por esta arquitectura es que puede ser transformada fácilmente para pasar de una aplicación monousuario a otra multijugador basada en una arquitectura cliente-servidor. Como se puede ver en la Figura 5.3, tan solo hay que crear dos módulos adicionales:

- En el lado del servidor, hay que crear un módulo de proxy para la vista de cada uno de los jugadores. Este módulo es una vista más de la lógica que recibe los eventos que ocurren en ésta. Cada evento recibido es procesado y enviado a través de la red hasta el cliente del jugador. Además, este módulo también recibe los comandos enviados a través de la red por los clientes, los procesa y se los pasa a la lógica del servidor. En el servidor también puede haber una vista y una aplicación que permitan que haya un jugador.

- En el lado del cliente, hay que crear un módulo de proxy de la lógica del juego. Este módulo es una copia de la lógica del servidor que recibe los comandos de la vista y de la aplicación del cliente, los procesa y los envía por la red hasta el servidor. Este módulo también se encarga de recibir a través de la red los eventos de la lógica del servidor y de redireccionarlos a la vista del cliente.

Esta arquitectura ha servido como inspiración para nuestra propuesta de arquitectura de alto nivel para el desarrollo de entornos virtuales de role-play. En esta arquitectura se han tenido en cuenta los ejes de variabilidad identificados anteriormente. Para ello, la arquitectura ha sido dividida en

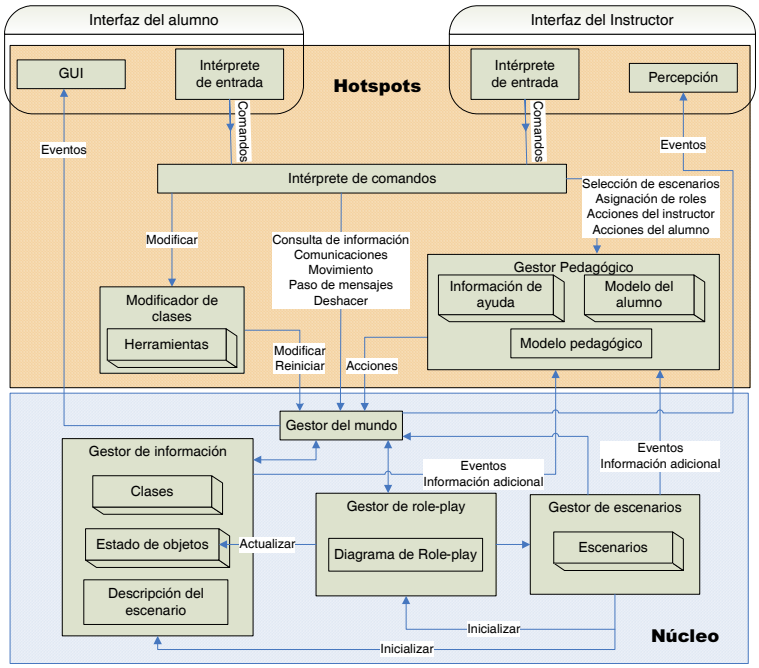


Figura 5.4: Arquitectura genérica para entornos virtuales de role-play.

módulos, de modo que cada uno de ellos puede ser instanciado de acuerdo a las decisiones tomadas para un entorno virtual concreto.

La Figura 5.4 muestra una vista general de la arquitectura propuesta (Jiménez-Díaz et al., 2007b). En ella se pueden diferenciar claramente dos subsistemas:

Núcleo Este subsistema es común a todos los entornos virtuales de role-play desarrollados. Contiene toda la lógica relacionada con las sesiones de role-play y es el responsable de las gestión de las entidades del mundo virtual, y de la información, tanto estática (información de clases y descripciones de los escenarios) como dinámica (estado de los objetos y RPDs) de la simulación.

Hotspots Este subsistema se compone de los módulos configurables que han de ser instanciados para la generación de distintos entornos virtuales de role-play. El nombre de este subsistema ha sido elegido de acuerdo al término empleado por los frameworks orientados a objetos para describir los puntos de variabilidad de los mismos.

A continuación pasaremos a detallar más en profundidad cada uno de los módulos de los que se compone cada uno de los subsistemas.

5.8.1. Módulos del Núcleo

Este subsistema es el responsable del almacenamiento de toda la información relacionada con las sesiones de role-play, así como de gestionar las entidades que habitan el entorno virtual. El Núcleo ha sido dividido en los siguientes módulos gestores: *Gestor del mundo*, *Gestor de información*, *Gestor de role-play* y *Gestor de escenarios*.

Al igual que el módulo de lógica en la arquitectura de McShaffry, es necesario que este módulo permanezca inmutable entre los distintos entornos de role-play que puedan ser creados. Por este motivo, el Núcleo recibe acciones de módulos externos a este subsistema a través de una interfaz definida de comandos. Además, el Núcleo propaga a sus observadores aquellos eventos que considera necesario notificar.

5.8.1.1. Gestor del mundo

Este módulo es el motor del entorno virtual. Se encarga de la gestión de todas las entidades que existen en el entorno virtual. Se encarga del movimiento de las mismas por el entorno y de responder a las solicitudes de información por parte de los usuarios del entorno. Para ello, almacena los datos necesarios para ligar cada entidad con la información almacenada en su inventario. Para ello, se comunicará con el *Gestor de información*, en busca de la información que demanda cada usuario.

Este módulo también es responsable de gestionar todas las acciones colaborativas que se produzcan en el sistema. En particular, da soporte a las comunicaciones entre alumnos e instructor y proporciona la gestión de las mecánicas de consenso bajo la petición de los módulos que necesiten hacer uso de ellas.

De cara al subsistema de Hotspots, el *Gestor del mundo* hace de fachada de comunicaciones. Cualquiera de los módulos de este subsistema que deseen comunicarse con el Núcleo lo deberá hacer a través de la interfaz de comandos que el *Gestor del mundo* proporciona. Posteriormente, este módulo será responsable de distribuir cada uno de los mensajes a los módulos del Núcleo correspondientes.

Por último, el *Gestor del mundo* es el encargado de emitir los eventos de cambios producidos en el entorno virtual. De esta forma, el entorno es independiente de la forma en la que dichos cambios sean representados al alumno así como de la existencia de sistemas inteligentes de tutoría o de tutores humanos. Cualquier módulo que desee que los eventos le sean notificados ha de registrarse como observador de este módulo, de la misma forma que se hace con la capa de lógica en la arquitectura de McShaffry.

5.8.1.2. Gestor de información

Es el módulo responsable de gestionar toda la información relacionada con el escenario que está siendo simulado. Contiene información sobre las clases de los objetos que componen el escenario, el estado de los mismos y la información de descripción del escenario.

Este gestor, además, ha de definir la interfaz de modificación de las clases de un escenario. De esta forma, las herramientas de modificación de clases del subsistema de Hotspots (en caso de existir) harán uso de esta interfaz para realizar todas las modificaciones para las que hayan sido diseñadas.

El *Gestor de información* también emite eventos de modificación de información y de solicitud de información adicional del escenario. Este último se emplea para que las distintas instanciaciones de entornos virtuales puedan presentar cualquier tipo de información adicional de la que dispongan.

5.8.1.3. Gestor de role-play

Este módulo almacena la información relacionada con el estado actual en el que se encuentra la sesión de role-play. En particular, almacena la información relacionada con el RPD.

Por otro lado, es responsable de gestionar el paso de mensajes. Cada vez que se solicita el paso de un mensaje, este módulo es responsable de interpretarlo, validarlo y ejecutarlo, en caso de que el mensaje sea correcto. Su ejecución implica la actualización del RPD y del estado de los objetos, si procede, contenidos en el *Gestor de información*.

También se encarga de las acciones relacionadas con la ejecución paso a paso de un escenario, ya que contiene la solución del escenario simulado, en caso de que ésta fuera proporcionada. También se encarga de deshacer acciones relacionadas con el paso de mensajes, devolviendo el RPD y el estado de los objetos al estado anterior al paso de mensaje deshecho. En caso de necesitar reiniciar un escenario, este módulo también se encarga de ello.

5.8.1.4. Gestor de escenarios

Este módulo actúa como repositorio de los escenarios de role-play contenidos en el entorno. Proporciona la interfaz básica con la que alumnos, instructor y otros sistemas puedan buscar y seleccionar el escenario que se desea simular en el entorno.

También es el responsable de la inicialización del *Gestor de información*. Una vez que se ha decidido qué escenario va a ser simulado, este módulo se comunica con el *Gestor de información* y le proporciona cuáles son los objetos que van a intervenir en el escenario, el estado inicial del que parten, las clases a las que pertenecen dichos objetos y toda la información básica que sirve para describir este escenario.

El *Gestor de escenarios* es capaz de emitir eventos para la carga de información adicional. Si este módulo se encuentra durante la carga de un escenario que éste dispone de información adicional entonces emite este evento, delegando en el subsistema de *Hotspots* —generalmente, en el *Gestor pedagógico*— la carga de esta información, que podrá variar entre distintas instanciaciones de entorno virtual.

5.8.2. Módulos Hotspots

Este subsistema se compone de los módulos que van a proporcionar las distintas variantes de entornos virtuales de role-play. Dependiendo de la forma en la que estos módulos son instanciados, el entorno tendrá unas determinadas características. Estos módulos han sido diseñados de modo que cubren los distintos ejes de variabilidad descritos en la Sección 5.7.

Se han identificado los siguientes módulos: *Interfaz del alumno, Interfaz del instructor, Intérprete de comandos, Modificador de clases* y *Gestor pedagógico*.

5.8.2.1. Interfaz del alumno

Este módulo es el responsable de proporcionar el aspecto del entorno virtual, así como de traducir las acciones físicas del alumno (pulsaciones de teclado, uso del ratón...) a comandos del entorno virtual.

Esta interfaz se compone de dos partes, una relacionada con la entrada de usuario y la otra con la visualización:

- Intérprete de entrada. Recoge los eventos de ratón, de teclado, comandos escritos en consola o cualquier otro dispositivo que use el alumno para interaccionar con el entorno virtual. Estos eventos son interpretados y traducidos a comandos inteligibles para el entorno virtual. Por tanto, este módulo posibilita modificar la forma en la que el alumno interactúa con el entorno de manera transparente a éste.

- Interfaz gráfica de usuario o GUI (del inglés *Graphical User Interface*). Es el módulo responsable de visualizar las entidades que habitan el mundo. Generalmente, esta representación será tridimensional y en este módulo se definirá el aspecto de cada uno de las entidades del entorno virtual de cara a los alumnos. Este módulo se encuentra a la escucha de los eventos que se comunican desde el *Gestor del mundo* para actualizar la visualización. Aunque generalmente usaremos vistas en 3D, hay que tener en cuenta que este módulo es transparente al núcleo del entorno virtual. Por ejemplo, podemos obtener vistas en 2D o podemos crear trazas textuales con las que depurar el entorno virtual durante su fase de desarrollo sin afectar al comportamiento del núcleo del entorno.

5.8.2.2. Interfaz del instructor

Este módulo es muy similar al anterior y es el que proporciona la interfaz de interacción entre el instructor y entorno virtual, así como la vista que el instructor tiene del entorno virtual. Hay que tener en cuenta que, dependiendo de las decisiones tomadas en los diferentes aspectos, el instructor puede ser humano o puede ser un sistema inteligente de tutoría.

En el primer caso, este módulo se comporta de manera muy similar a la *Interfaz del alumno*, pudiendo añadir nuevas acciones propias del instructor o nuevas vistas que ayuden a controlar el escenario que está siendo ejecutado.

Por el contrario, si el instructor es un sistema de tutoría inteligente, esta interfaz se comporta como el módulo de percepción de este sistema, que filtra y comunica los eventos producidos en el entorno virtual al tutor inteligente contenido en el *Gestor pedagógico*. De acuerdo a estos estímulos, el tutor tomará una serie de decisiones sobre la ayuda que ha de proporcionar a los alumnos. Las acciones que el tutor inteligente decide llevar a cabo son traducidas por el intérprete de entrada a los comandos que este entorno virtual entiende.

5.8.2.3. Intérprete de comandos

Este módulo define el repertorio de comandos que pueden ser ejecutados por un entorno virtual y al que serán traducidas las acciones de alumnos, instructores y otros tipos de agentes. Además, se encarga de interpretar los comandos que le llegan y distribuirlos a los diferentes módulos del sistema.

Existen un conjunto de comandos que van a ser comunes a todos los entornos virtuales, por lo que parte de este módulo podría quedar fijo en el núcleo del entorno virtual. En cambio, este módulo se ha llevado al subsistema Hotspots ya que algunos de los comandos pueden ser válidos sólo para algunos entornos. Por ejemplo, se puede definir un repertorio de comandos que sean empleados para modificar las clases de un escenario. Este repertorio sólo es necesario para aquellos entornos que dispongan de herramientas de modificación de clases, mientras que es inservible para un entorno en el que tan sólo se va a observar el comportamiento de un sistema.

Este módulo también es el responsable de coordinar las acciones que son traducidas a un solo comando. Por ejemplo, cuando se quiere pasar un mensaje de un objeto a otro son necesarias varias acciones: crear el mensaje, lanzar la pelota y que el receptor del mensaje lo recoja. El *Intérprete de comandos* se responsabiliza de coordinar todas estas acciones para, finalmente pasar el siguiente paso de simulación de la ejecución al *Gestor de role-play* a través de la fachada de comandos del *Gestor del mundo*. El *Gestor de role-play* se encargará de actualizar el RPD y el estado de los objetos convenientemente.

5.8.2.4. Modificador de clases

Este módulo implementa las herramientas y las estrategias empleadas para modificar las clases de un escenario. Es el responsable de proporcionar los mecanismos que permitan modificar clases de manera aislada o de realizar tareas de refactorización sobre distintas clases.

Independientemente del tipo de modificaciones que se puedan realizar, este módulo ha de traducir todas las acciones de las herramientas a las acciones básicas de modificación de clases que proporciona el *Gestor de información*.

En el caso de que las modificaciones puedan ser realizadas durante la fase de simulación, este módulo ha de notificar al *Gestor de role-play* si se ha de reiniciar o no el escenario que está siendo simulado.

5.8.2.5. Gestor pedagógico

Este módulo es el responsable de la mayoría de las acciones relacionadas con la existencia de un sistema de tutoría inteligente. En concreto, se encarga de las siguientes tareas:

- Define las estrategias de selección de escenarios y de asignación de roles durante la fase previa a la simulación de un escenario. De acuerdo a la estrategia implementada en este módulo, se informará al *Gestor de escenarios* de cuál es el escenario que se va a simular y cuál es la asignación de roles que se ha producido, para que éste realice la consecuente inicialización del entorno virtual.

- Fija la estrategia de intervención del instructor en el entorno virtual. Define las acciones que éste puede realizar y se encarga de comunicar con los pertinentes módulos del núcleo para llevar a cabo estas acciones como, por ejemplo, deshacer un paso de ejecución o comunicarse con el resto de participantes.

- Almacena la información de ayuda y define las estrategias para proporcionar esta información a los alumnos.

- Almacena toda la información del escenario adicional que será accesible a través del inventario del elemento contenedor de la descripción del escenario. Esta información puede ser muy variada, desde diagramas de clase que muestran el diseño final al que los alumnos han de llegar, hasta posibles soluciones del escenario, pasando, por ejemplo, por información referente a patrones de diseño.

El *Gestor de escenarios* delega en este módulo toda la carga de información adicional del escenario. De igual modo, el *Gestor del mundo* delegará en este módulo cuando se le demande información adicional del escenario

simulado. Para ello, este módulo ha de permanecer a la escucha de los eventos relacionados con la carga y la presentación de información adicional que provengan del *Núcleo*.

En caso de los sistemas de tutoría inteligente, este módulo contiene toda la información necesaria para llevar a cabo las estrategias concretas que se definan. Por ejemplo, este módulo contiene el modelo del alumno o el modelo pedagógico del instructor, en caso de que el sistema de tutoría inteligente lo necesite.

5.9. Conclusiones

A lo largo de todo este capítulo se ha desarrollado una de las principales aportaciones realizada por esta Tesis: la definición de los entornos virtuales de role-play. Estos entornos representan el traslado de las sesiones presenciales de role-play a un mundo virtual en el que los alumnos intervienen en el desarrollo de estas sesiones. En estos entornos todos los alumnos participan activamente interaccionando con las distintas entidades que pueblan el entorno virtual. Además, estos entornos también han sido pensados para fomentar la interacción entre los alumnos, al igual que en las sesiones presenciales de role-play. En ellas, los alumnos se comunican, comparten sus opiniones y colaboran en el desarrollo de tareas de diseño o en el análisis y evaluación de aplicaciones software.

En la definición de los entornos virtuales de role-play se han cuidado las dificultades relacionadas con la enseñanza de la orientación a objetos descritas en el Capítulo 2 y se han aplicado las prácticas más interesantes que se han extraído de las sesiones de role-play presenciales descritas en el Capítulo 4. También se han tenido en cuenta las soluciones adoptadas por las herramientas analizadas en el Capítulo 3 y los inconvenientes detectados en dichas herramientas.

El principal propósito de los entornos virtuales de role-play es fomentar las experiencias de diseño de los alumnos, ya sea mediate el análisis de aplicaciones ya desarrolladas, ya sea mediante el diseño y la revisión de otras aplicaciones. Como se ha destacado anteriormente, la experiencia es clave para aprender a diseñar software orientado a objetos de calidad. Las sesiones de role-play fomentan el desarrollo de este tipo de actividades y favorecen que tanto alumnos como el instructor compartan sus conocimientos y experiencias de diseño. Los entornos virtuales de role-play son un reflejo de estas sesiones presenciales, disponen de herramientas que facilitan la colaboración entre alumnos y los escenarios que en ellos se simulan son ricos en información de diseño.

Los entornos virtuales de role-play, a diferencia de la mayoría de las herramientas de enseñanza de orientación a objetos, no hacen uso de ningún lenguaje de programación en concreto. En estos entornos se han definido

mecánicas y representaciones para conceptos generales de la orientación a objetos: clases, objetos o paso de mensajes, entre otros. Se han cuidado las representaciones utilizadas dentro del entorno virtual, de modo que haya una clara diferencia entre conceptos que el alumno ha de distinguir claramente, como clases y objetos. Además, estos entornos hacen uso de la metáfora de la antropomorfización de igual modo que en las sesiones presenciales de role-play. El alumno queda inmerso en el entorno virtual mediante la presencia de un avatar que le representa, que le permite interactuar con otras entidades del entorno y que le hace meterse en el papel de un objeto durante la simulación de los escenarios de ejecución.

En estos entornos virtuales también se ha cuidado la naturaleza dinámica de lo que está siendo representado. Una sesión de role-play representa un escenario de ejecución de una aplicación, por lo que los alumnos han de poder ver claramente las interacciones que se producen entre los objetos y los mensajes que se pasan entre ellos. Estas interacciones producen cambios, por lo que se ha cuidado que los alumnos puedan interactuar con otro objetos para observar las modificaciones que se producen en su estado.

Los entornos virtuales de role-play favorecen el estudio y el diseño de ejemplos complejos. Los escenarios son muy ricos en información pero el alumno tiene total control de qué información desea ver en cada momento. Además, esta información se consigue interactuando con las entidades del mundo, lo que hace que el alumno se involucre más dentro del entorno virtual. Los alumnos también imponen el ritmo de la simulación. Pueden decidir los pasos de mensajes que desean ejecutar, pueden volver atrás y experimentar con el paso de otros mensajes y pueden modificar el diseño. En resumen, pueden explorar, experimentar y desarrollar aplicaciones orientadas a objetos de entidad usando una herramienta sencilla, con mecánicas muy básicas y familiares a los alumnos, ya que son similares a las que se producen en un videojuego y a las que se ven en una sesión de role-play.

La interacción del alumno en los entornos virtuales de role-play ha sido diseñada de modo que se puedan cubrir en gran medida los distintos niveles de participación estudiados en el Capítulo 3. Las herramientas analizadas en ese capítulo cubrían la mayoría de los niveles de participación pero se destacó el poco cuidado puesto en cubrir el nivel de presentación, uno de los más interesantes ya que promueve la reflexión, la discusión y la colaboración entre alumnos. Este nivel es de especial interés en los entornos virtuales de role-play, que sirven como plataforma para la colaboración entre alumnos en el análisis y diseño de aplicaciones orientadas a objetos. Se proporcionan mecanismos para la comunicación entre ellos y el instructor dentro del mundo, para la toma de decisiones de diseño y para guardar las actividades que en ellos se realicen. Éstas pueden ser posteriormente revisadas y evaluadas por el instructor o por otros alumnos.

La propuesta de entornos realizada en este capítulo no es cerrada sino

que se han analizado las variaciones que se pueden realizar en la metodología general basada en sesiones de role-play. De acuerdo a estas variaciones se pueden definir distintos tipos de entornos, todos ellos centrados en la simulación de escenarios de role-play, pero en los que la libertad de los alumnos durante la simulación, las acciones que pueden realizar o la participación del instructor sean configurables de acuerdo a los objetivos para los que ha sido diseñada la herramienta. Para dar soporte al diseño de este tipo de entornos, se ha definido una arquitectura general de los mismos. Haciendo uso de esta arquitectura, en el próximo capítulo se describirá el desarrollo de dos entornos virtuales de role-play distintos: ViRPlay3D y ViRPlay3D2.

Capítulo 6

Desarrollo de entornos virtuales de role-play: ViRPlay3D y ViRPlay3D2

No, no lo intentes. Hazlo, o no lo hagas.
Pero no lo intentes.

La Guerra de las Galaxias (1977)

En el Capítulo 5 se ha descrito una propuesta para el traslado del role-play para la enseñanza de la orientación a objetos a entornos virtuales. Para ello se ha partido del análisis y la generalización de la sesiones de role-play diseñadas y descritas en el Capítulo 4. Se han identificado las fases por las que pasan dichas sesiones, la información utilizada y las principales mecánicas que se emplean durante el role-play. A partir de aquí, se ha propuesto la forma en la que cada una de estas necesidades iba a ser plasmada dentro de un entorno virtual que imitase las sesiones presenciales usando role-play.

Las sesiones de role-play admiten variaciones que las permiten adaptarse a distintos objetivos pedagógicos. Estos ejes de variabilidad han sido trasladados a la propuesta de entornos virtuales de role-play y, con ellos se ha planteado un modelo de arquitectura para el desarrollo de este tipo de entornos. Esta arquitectura presenta un serie de módulos fijos que componen el núcleo de este tipo de entornos pero deja abierta la especificación de otros que serán los que posibilitan el desarrollo de distintos entornos.

Para poner en práctica estas ideas, en este capítulo se especifica el desarrollo de dos prototipos de entornos virtuales de role-play. Ambos han sido diseñados siguiendo la arquitectura general propuesta para este tipo de entornos. Sin embargo, difieren en la instanciación de los distintos ejes de variabilidad, pues tienen objetivos pedagógicos diferentes y, en consecuencia, el desarrollo de las sesiones en cada uno de los entornos será muy distinto.

El primero de los prototipos desarrollados es ViRPlay3D (*Virtual Role-Play 3D*). Es un entorno virtual de role-play monousuario para el análisis y la evaluación de aplicaciones orientadas a objetos ya desarrolladas. Esta primera instanciación muestra cómo se han llevado las mecánicas básicas del role-play a este tipo de entornos. ViRPlay3D se puede considerar como una herramienta de visualización interactiva en la que el usuario observa y participa en episodios seleccionados del comportamiento en ejecución de una aplicación.

En la Sección 6.1 se ha realizado una amplia descripción de este prototipo. En particular, se ha hecho especial hincapié en las decisiones de diseño tomadas de acuerdo a los ejes de variabilidad de la propuesta de los entornos virtuales de role-play, como se puede ver en la Sección 6.1.1. También se ha detallado la representación particular empleada en ViRPlay3D para cada uno de los elementos integrantes de este tipo de entornos, así como el tipo de información contenida y las mecánicas de interacción entre entorno y usuario —Secciones 6.1.2, 6.1.3 y 6.1.4, respectivamente. Posteriormente, en la Sección 6.2 se ha entrado en detalle en cómo se ha realizado la instanciación de la arquitectura general de los entornos virtuales de role-play y algunas particularidades de implementación de este prototipo. Adicionalmente se han desarrollado y empleado un conjunto de herramientas destinadas a la creación de los contenidos necesarios para la simulación de una sesión de role-play en ViRPlay3D. Estas herramientas han sido descritas en la Sección 6.3. Para finalizar con la descripción de este primer prototipo, en la Sección 6.4 se relata un supuesto escenario de uso de un alumno que utiliza el entorno para analizar el funcionamiento del simulador de biología marina (MBSCS, del inglés *Marine Biology Simulation Case Study*), descrito en el Capítulo 3.

Una vez desarrollado ViRPlay3D se ha realizado una pequeña evaluación del mismo. En la Sección 6.5 se relata como se ha llevado a cabo esta evaluación y cuál ha sido la valoración que se ha realizado de este primer entorno virtual de role-play. También se analizan las críticas realizadas al entorno, de modo que éstas sean tenidas en cuenta en el desarrollo del siguiente entorno.

El siguiente entorno diseñado se conoce como ViRPlay3D 2. Éste no es una mera revisión del anterior sino que es un entorno que, aún siguiendo la misma arquitectura de los entornos virtuales de role-play tiene un objetivo pedagógico completamente distinto: ViRPlay3D 2 ha sido desarrollado para la creación y revisión de un diseño orientado a objetos de manera colaborativa. En este entorno ya no participa un solo usuario sino que en ViRPlay3D 2 se reproducen más fielmente las sesiones presenciales de role-play relatadas en el Capítulo 4, en el que un conjunto de alumnos discuten y modifican el diseño de clases de una aplicación a medida que simulan la ejecución de escenarios de uso de la misma. Este entorno realiza pequeñas modificaciones sobre las representaciones visuales y las mecánicas básicas de ViRPlay3D de acuerdo a las conclusiones obtenidas de la evaluación del mismo. Además,

ViRPlay3D 2 añade todas las acciones relacionadas con la colaboración entre los distintos usuarios del sistema y la modificación del diseño de clases.

La descripción de ViRPlay3D 2 se ha realizado de manera análoga a la de ViRPlay3D. Primeramente se ha descrito el nuevo entorno y se han detallado las decisiones de diseño tomadas de acuerdo a los ejes de variabilidad —Sección 6.6.1. Más adelante se han especificado las representaciones concretas empleadas para los elementos de estos entornos —Sección 6.6.2—, el tipo de información accesible desde el entorno —Sección 6.6.3— y la manera en la que los usuarios podrán participar dentro del mismo —Sección 6.6.4. Posteriormente, en la Sección 6.7 se han pormenorizado los detalles de instanciación de la arquitectura así como su implementación. A continuación nos hemos detenido en las nuevas herramientas de autoría necesarias para el desarrollo de los escenarios de simulación de ViRPlay3D 2 —Sección 6.8. Para finalizar del mismo modo que con la descripción de ViRPlay3D, la Sección 6.9 relata un posible escenario de uso de este prototipo en el que se ha imitado una de las sesiones presenciales descritas en el Capítulo 3. Éstas se utilizaron para la enseñanza de patrones de diseño y emplearon como ejemplo de aplicación un editor gráfico muy simple.

Para concluir, nuevamente se ha realizado una pequeña evaluación de ViRPlay3D 2. En la Sección 6.10 hay un resumen de las principales valoraciones obtenidas del prototipo y las críticas que se han hecho a este entorno.

6.1. Descripción general de ViRPlay3D

El primer entorno desarrollado siguiendo las ideas y la arquitectura de los entornos virtuales de role-play es ViRPlay3D (Jiménez-Díaz et al., 2005a,c, 2007a). En la Figura 6.1 se puede ver el aspecto final de esta herramienta. Este primer entorno se ha diseñado con dos objetivos fundamentales:

- De cara al desarrollo de esta tesis, este entorno ha servido para poner en práctica el traslado de las técnicas de role-play a un entorno virtual y comprobar la aceptación de esta aproximación por parte de la comunidad investigadora centrada en temas de enseñanza.
- Desde el punto de vista pedagógico, este entorno ha sido diseñado como herramienta monousuario de visualización de interacciones con el fin de comprender el comportamiento de una aplicación. El alumno observa y controla el avance de la visualización. Además, en cada paso de ejecución tienen la posibilidad de jugar el papel del objeto activo en ese momento, no estando limitado a representar siempre al mismo objeto durante toda la sesión.

Como ya se ha hecho mención en capítulos anteriores, la comprensión de las interacciones y de la delegación de responsabilidades entre objetos

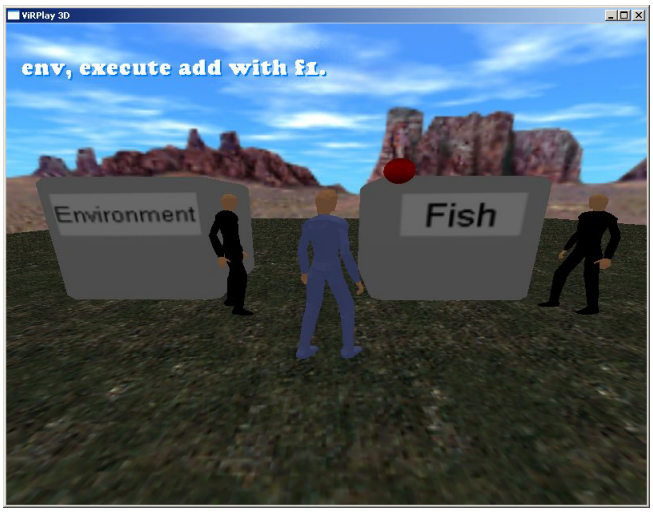


Figura 6.1: Aspecto general de ViRPlay3D durante la ejecución del ejemplo descrito en la Sección 6.4.

es un concepto elemental de la enseñanza de la orientación a objetos. Este concepto es indispensable para que el alumno pueda llegar a comprender el funcionamiento de una aplicación, principalmente en sistemas complejos. Además, la comprensión de las interacciones lleva implícito entender otros conceptos elementales como son la herencia y el polimorfismo. Generalmente, leer documentación sobre las clases que componen un sistema no facilita la tarea de análisis de la aplicación. Además, para los alumnos más novatos, seguir la traza de ejecución en un entorno de desarrollo puede llegar a ser aún más complicado, ya que suelen perderse en una maraña de pasos de mensajes.

ViRPlay3D facilita estas tareas de análisis en alumnos principiantes. En ViRPlay3D, el alumno puede ver y analizar el comportamiento de un sistema complejo de manera más controlada y con una interfaz más motivante e inmersiva que la que puede aportar un entorno de desarrollo. En este entorno, el role-play se usa como medio para realizar el análisis del comportamiento de manera controlada. El alumno no se enfrenta a la aplicación completa sino que va estudiándola poco a poco a través de la simulación de distintos escenarios de ejecución.

En este entorno, el alumno no sólo aprende la estructura dinámica —en tiempo de ejecución— del sistema sino que va a ir adquiriendo también los conocimientos sobre su estructura estática. Y esto lo va a ir realizando a medida que lo necesite, navegando por el mundo virtual e interaccionando con el resto de los elementos que en él aparecen para conseguir la información que necesita en el momento en el que la necesita.

Debido a la importancia que tiene la interacción del alumno en este tipo de entornos, se han incluido mecánicas para integrarlo en el desarrollo de la simulación. Para ello, se permite que en cualquier momento de la simulación el alumno pueda jugar el papel del objeto activo en ese momento, decidiendo cuál es el siguiente paso de mensaje que se ha de realizar. En caso de error, se informa al alumno de que ese no es el mensaje correcto. De esta forma, el alumno comprueba por sí mismo si está comprendiendo realmente el funcionamiento del sistema.

6.1.1. Decisiones de diseño

El desarrollo de ViRPlay3D se ha realizado siguiendo la arquitectura general para entornos virtuales de role-play descrita en el capítulo anterior. Ahora bien, la instanciación de esta arquitectura ha supuesto decidir cómo se han de configurar cada uno de los ejes de variabilidad de los que se compone esta arquitectura.

En primera instancia, se ha decidido elaborar un entorno sencillo en el que se vieran reflejadas las principales características de los entornos virtuales de role-play. Se han dejado a un lado cierta información de los escenarios y algunas mecánicas de estos entornos, así como todas las alternativas que añadirían complejidad al prototipo, como son los agentes pedagógicos o los tutores inteligentes.

Una de las principales características de este entorno frente a lo propuesto en la arquitectura general de los entornos de role-play es que es monousuario. En ViRPlay3D hay un único alumno que observa y controla la sesión de role-play. Esto no implica que el alumno sea un elemento pasivo de la escena. Ya se ha destacado la importancia de que el alumno participe en estas sesiones. Como se verá más adelante, el entorno dispone de un conjunto de mecánicas que harán del alumno una parte activa dentro de ViRPlay3D.

Esta decisión ha influido altamente en la mayoría de los ejes de variabilidad. El aspecto *Asignación de roles* no tiene sentido para esta herramienta ya que el alumno no tiene un rol fijo a lo largo de toda la sesión. Para cada escenario existirán un conjunto de agentes responsables de ejecutar el comportamiento asociado a cada uno de los objetos que intervienen en el sistema. El comportamiento de estos agentes es similar a un guión de role-play: cada agente sabe qué es lo que ha de hacer en cada momento y cómo ha de modificar su estado en respuesta a un determinado paso de mensaje.

En cuanto al aspecto *Selección de escenarios*, el alumno es el responsable de decidir qué escenario va a ser ejecutado de entre un conjunto de escenarios resueltos. Para proporcionar al alumno un número suficiente de escenarios representativos entre los que poder elegir, se han desarrollado una serie de herramientas de autoría que facilitan al instructor la generación de nuevos escenarios de ejecución. Estas herramientas serán descritas en la Sección 6.3.

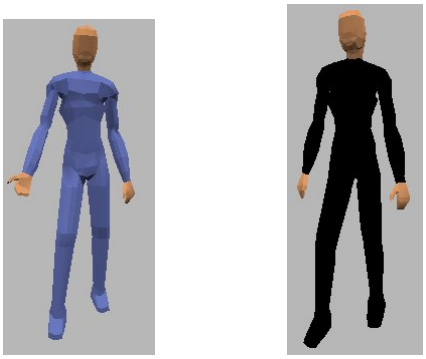
Una decisión de diseño de ViRPlay3D es no contemplar la intervención del instructor a través de la propia herramienta. Sin embargo, el propio entorno hace de agente moderador durante la simulación. De acuerdo al *Grado de libertad* el alumno es libre de moverse por el mundo para consultar información a los distintas entidades que en él existan. El alumno también se encarga de controlar la ejecución del escenario, de modo que va solicitando el siguiente paso de ejecución. En este caso, el entorno toma el control de la ejecución y coordina al objeto activo y al objeto receptor del siguiente mensaje para que realicen el siguiente paso de ejecución.

Además, el alumno puede decidir en cualquier momento interpretar el rol del objeto activo. En este momento, se pondrá en la piel de este objeto y, mediante una interfaz de interacción, se responsabilizará de configurar el siguiente paso de mensaje. En caso de que el alumno yerre con el siguiente mensaje a pasar, se le informará de su error y se le invitará a, o bien intentarlo de nuevo, o bien dejar que el entorno, como agente moderador, le muestre el siguiente paso de ejecución.

Como se puede extraer de lo anterior, el aspecto *Provisión de ayuda* se ha simplificado enormemente: si el alumno se equivoca, puede solicitar el paso de ejecución correcto. Es cierto que esta decisión podría no ser la más adecuada desde el punto de vista pedagógico. Guiar al alumno ciegamente hasta completar el escenario puede no ser, pedagógicamente hablando, un buen método para que el alumno analice y comprenda el diseño de un sistema. Sin embargo, para el objetivo de ViRPlay3D, esta decisión es suficiente para comprobar si la metáfora utilizada en los entornos virtuales de role-play es comprensible y si parece apropiada para interpretar el comportamiento de un sistema.

ViRPlay3D se ha creado con el objetivo pedagógico de ayudar a analizar y comprender una aplicación y centrarse en conceptos de interacción y delegación de responsabilidades entre objetos. De ahí que en este prototipo no tenga sentido la inclusión de herramientas para realizar modificaciones en el diseño de las clases del caso de estudio empleado a lo largo del desarrollo de las sesiones. Por tanto, en el aspecto *Modificaciones del diseño inicial* se han obviado todas las decisiones relacionadas con quién ha de realizar estas modificaciones y cuándo realizarlas.

Por último, relacionado con el aspecto *Evaluador*, ViRPlay3D ha sido desarrollado para que el alumno pueda trabajar libremente en las sesiones de role-play. No necesitará de un instructor humano ya que el entorno es responsable de que el escenario se ejecute correctamente. Sin embargo, ViRPlay3D también guarda las interacciones que el alumno realiza durante la ejecución de un escenario. De esta forma, ViRPlay3D puede ser empleada como una herramienta con la que comprobar si el alumno está comprendiendo el diseño objeto de estudio usando esta herramienta. Los archivos de bitácora generados por el entorno ayudan al instructor humano a saber si los



(a) Avatar del alumno

(b) Avatar de un objeto

Figura 6.2: Avatares antropomórficos utilizados en ViRPlay3D: (a) el alumno y (b) un objeto.

escenarios están cumpliendo su cometido, si existen carencias generalizadas debido a la repetición de errores o si hay que crear nuevos escenarios de ejecución para analizar más ciertos conceptos.

6.1.2. Representación metafórica

Como ya se ha explicado en el Capítulo 2, los entornos virtuales de role-play trasladan a un entorno virtual la misma metáfora que hay tras una sesión de role-play presencial. En ellas, los alumnos representan el rol de los objetos. De acuerdo a esta metáfora, se han utilizado modelos antropomórficos para representar a los objetos que intervienen en la escena. El alumno también dispone de un modelo antropomórfico similar pero distinguible de los objetos por su color —como se puede ver en la Figura 6.2, los objetos son humanos vestidos de negro (6.2(b)) mientras que el alumno es representado por un humano de azul (6.2(a)).

Las clases tienen también su representación dentro del entorno virtual. Éstas son representadas mediante un bloque sobre el cual aparece el nombre de la clase a la que representa, como se puede ver en la Figura 6.3. Como se puede ver, clases y objetos son perfectamente distinguibles dentro del entorno, de modo que el alumno no pueda ser conducido a equívoco alguno.

Los escenarios han sido desarrollados de modo que, inicialmente, cada objeto aparece lo más cerca posible de la clase de la que es instancia. Los objetos no aparecen identificados y son todos iguales por lo que el alumno ha de acercarse a ellos para, mediante la acción *Mirar a*, saber de qué objeto se trata. Esto hace que el alumno tenga que moverse por el mundo y que interaccionar con los objetos para conocer a qué objetos representan, fomentando

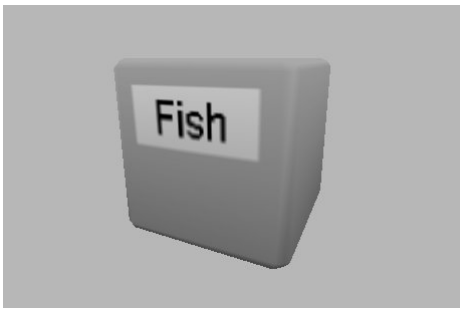


Figura 6.3: Entidad visual que representa a una clase dentro del entorno (en este caso, una clase llamada `Fish`).

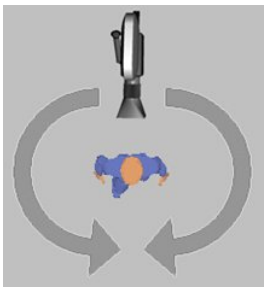


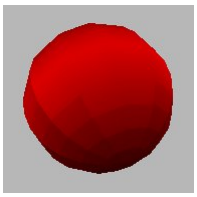
Figura 6.4: Movimiento de rotación de la cámara controlado por el alumno.

su interactividad dentro del sistema.

En este primer prototipo, el alumno no representa el papel de ningún objeto fijado de antemano por lo que se ha decidido la utilización de una cámara en tercera persona para que éste pueda observar la escena. Esta cámara sigue continuamente al avatar del alumno. De esta forma, éste tiene control sobre lo que ve moviendo a su avatar. Además, se ha permitido que el alumno pueda rotar la cámara alrededor del avatar del alumno —como se puede ver en la Figura 6.4— ya que, en ocasiones, no es fácil orientar la cámara tan solo con los movimientos del avatar.

En cuanto al paso de mensajes, éste ha sido representado tal y como fue descrito en el anterior capítulo. En el entorno existe una pelota (Figura 6.5(a)) que se va pasando de un objeto a otro para representar el flujo de mensajes. El objeto que en cada momento sostiene la pelota será el objeto que actualmente se encuentra activo y el responsable del paso del siguiente mensaje.

Por otro lado, cada vez que un mensaje es invocado o retornado existe una ayuda textual que permite que el alumno sepa qué mensaje está siendo pasado. Para ello, el entorno sobreimpresiona en pantalla un texto (Figura 6.5(b)) en el que se puede leer, en forma de diálogo entre los objetos, el mensaje que está siendo pasado. Los tipos de mensajes que aparecen so-



(a) Pelota.

(b) Texto del mensaje pasado.

Figura 6.5: Representaciones visuales relacionadas con el paso de mensajes: (a) la pelota y (b) el texto sobreimpresionado en pantalla relativo al mensaje pasado.

breimpresionados siguen patrones como los siguientes:

"*objetoX*, execute *métodoX* with *valorX* and *objetoY*"

cuando se envía el método *métodoX* al objeto *objetoX* con *valorX* y *objetoY* como parámetros reales, o

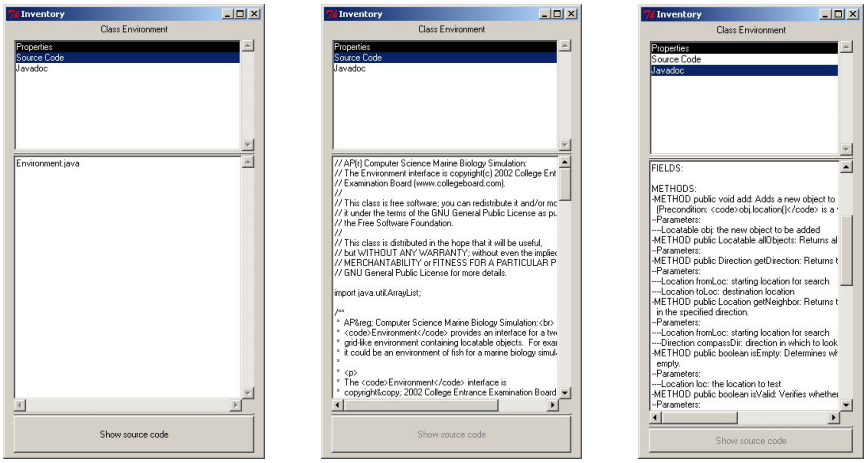
"*métodoX* is done, returning *valorY*"

cuando se trata de la finalización de la ejecución del método *métodoX* devolviendo *valorY*.

6.1.3. Representación de información del role-play

Una de las principales características de los entornos virtuales de role-play es su riqueza en información relacionada con el escenario que está siendo simulado. ViRPlay3D contiene una gran cantidad de información distribuida por los inventarios de las entidades que aparecen en la escena. De acuerdo al tipo de entidad consultada, el inventario contendrá distinto tipo de información. A continuación detallaremos la información accesible desde cada uno de los inventarios.

El primero que vamos a tratar es el inventario de la clase. Para este primer prototipo no se han usado las tarjetas CRC como información de una clase. En su lugar, se ha empleado una versión compacta de documentación de la clase y el código fuente de la misma. De acuerdo a esto, cuando el alumno realiza la acción *Mirar a* sobre alguna de las clases existentes en la escena aparecerá un inventario como el de la Figura 6.6 en el que el alumno controlará qué tipo de información desea consultar. Como se puede ver, el alumno puede decidir si quiere ver la documentación de la clase o si, por el contrario, desea ver su código fuente. Para este segundo tipo de información, el alumno controla si le basta con saber cuál es el archivo que contiene el



(a) Referencia al código fuente.

(b) Presentando el código fuente.

(c) Presentando la documentación de la clase.

Figura 6.6: Inventario de clases en ViRPlay3D: (a) el inventario presenta una referencia al código fuente; (b) el inventario presenta el código fuente; (c) el inventario muestra la documentación descriptiva de la clase.

código fuente de la clase para su posterior consulta o si realmente desea ver su contenido en la ventana del inventario.

Si el alumno se acerca a una entidad que representa a un objeto y usa la acción *Mirar a* aparecerá entonces el inventario del objeto. Este inventario se puede ver en la Figura 6.7. En él, el alumno tiene acceso tanto a la información estática como dinámica del objeto. En cuanto a la información estática, el alumno puede consultar la clase a la que pertenece el objeto y, de igual modo que en el inventario de la clase, puede consultar su documentación y su código fuente. En cuanto a la información dinámica, el alumno puede consultar el estado del objeto y observar el valor actual de sus atributos en cualquier momento de la simulación. Así podrá ver qué pasos de mensajes afectan al cambio de estado de los objetos. Como se puede ver en la figura, el estado aparece como parejas de atributo del objeto y valor actual del mismo.

El alumno también puede consultar el inventario de la pelota para saber más sobre el último mensaje que se ha pasado o sobre el mensaje que actualmente está siendo pasado —si consulta el inventario de la pelota mientras ésta se encuentra en pleno vuelo. En este inventario y como se puede ver en la Figura 6.8, el alumno puede consultar toda la información referente al mensaje pasado: quién es el invocador del método correspondiente, sobre quién ha sido invocado y cuál es el valor de cada uno de los parámetros del método en el momento en el que éste ha sido invocado. En el caso de que el método tenga retorno y el paso de mensaje haya finalizado, el alumno puede

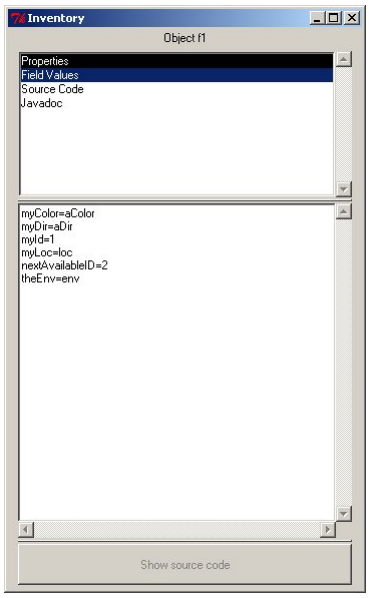
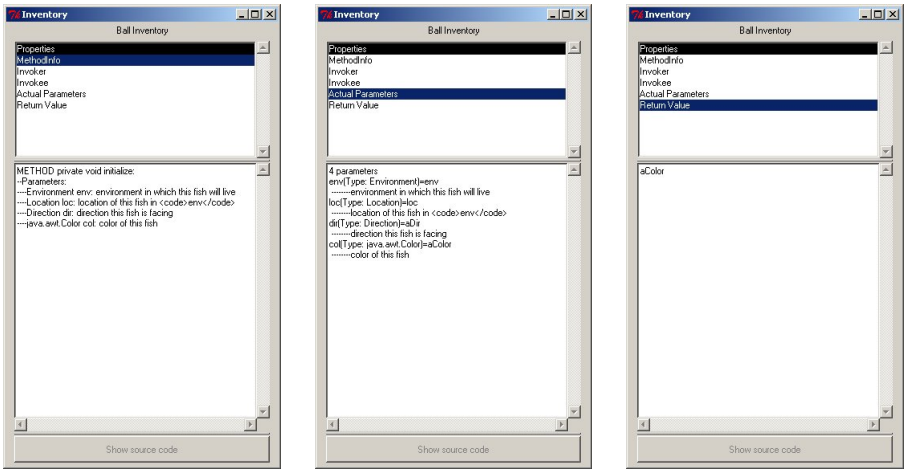


Figura 6.7: Inventario de un objeto. Aquí se está consultando el estado de un objeto.

consultar cuál ha sido el valor retornado. Por último, también se incluye una descripción del mensaje que se ha pasado. Esta descripción es extraída directamente de la información de la clase a la que pertenece el método invocado.

Para este primer prototipo no se ha incluido ninguna entidad que disponga de la información sobre el escenario que actualmente está siendo representado. Como se verá en la Sección 6.4, los escenarios creados están basados en el MBSCS, un caso de estudio ampliamente documentado. Además, estos escenarios han sido simplificados en lo máximo posible, sin dejar a un lado la complejidad que hemos exigido a lo largo de esta Tesis para los ejemplos —varios objetos, de distintas clases o, incluso, varias instancias de la misma clase. Por ello, consideramos que es posible obviar este tipo de información en este primer prototipo.

Así mismo, se ha obviado la representación del histórico de la simulación. Los escenarios son sencillos y están bien documentados, generalmente con diagramas de secuencia, por lo que consideramos que el seguimiento del escenario no entraña grandes dificultades. Si el alumno desea saber en qué momento de la simulación se encuentra puede consultar el inventario de la pelota con el fin de saber cuál es el último mensaje pasado. Asumimos que, para ViRPlay3D, esta información será suficiente para que el alumno se ubique dentro de la simulación del escenario.



(a) Descripción del método.

(b) Valor de los parámetros.

(c) Valor retornado.

Figura 6.8: Distintas vistas del inventario de la pelota en ViRPlay3D: (a) presentando la descripción de un método; (b) presentado el valor de los parámetros; y (c) mostrando el valor devuelto por un método.

6.1.4. Interacción

A pesar de que las decisiones de diseño tomadas puedan hacer creer que ViRPlay3D es un entorno de visualización limitado en interacción, esto no es así. Es cierto que este primer prototipo no presenta algunas de las mecánicas de interacción propuestas en la descripción general de un entorno virtual de role-play pero esto no lo convierte en un entorno pasivo. El alumno sigue siendo una parte importante en el funcionamiento de estos entornos y su participación es necesaria para el desarrollo de las sesiones.

El alumno dispone de su propio avatar dentro de ViRPlay3D. Es su medio para navegar por el entorno y explorarlo. ViRPlay3D obliga a que el alumno se tenga que desplazar por el mundo en busca de información y tenga que explorar las entidades que en él se encuentran para adquirir los conocimientos necesarios para comprender la aplicación que está siendo analizada haciendo uso de esta herramienta.

La mecánica principal para explorar el entorno virtual es la acción *Mirar a* a la que ya hemos hecho referencia con anterioridad. Cada vez que el avatar del alumno está lo suficientemente cerca de otra entidad dentro del mundo de la que puede obtener información, aparece la opción de ejecutar la acción de *Mirar a* dicha entidad, como se puede ver en la Figura 6.9. Si el alumno decide ejecutar dicha acción, entonces aparecerá el inventario relacionado con el tipo de entidad que está siendo consultada —clase, objeto

Figura 6.9: Acción *Mirar a*. En este caso, el alumno tiene la opción de consultar el inventario del objeto *f1* o de consultar el de la pelota.

o pelota. También, como se puede ver en la figura, hay ocasiones en las que el alumno está mirando simultáneamente a varias entidades. En este caso, aparecerán todas las acciones que puede realizar. El alumno tendrá entonces que seleccionar cuál de todas las acciones desea realizar —en la figura, la acción seleccionada actualmente está en rojo.

El alumno tiene el control total de la ejecución de la simulación. Tras cada paso, ViRPlay3D queda a la espera de que el alumno solicite el siguiente paso de mensaje. Mientras esto no ocurra, la simulación permanece detenida. Esto facilita que el alumno pueda moverse libremente para buscar información sin necesidad de estar pendiente de lo que ocurre en el mundo. Una vez que el alumno está dispuesto para el siguiente paso de ejecución, solicita a ViRPlay3D que lo ejecute. Además, el alumno también puede volver atrás para repetir el paso de un mensaje o, incluso, reiniciar la simulación de un escenario.

Ya que este entorno no posibilita la creación o modificación del diseño, no es necesario proporcionar al alumno la opción de dar por concluido el escenario. Éste ha de completarlo de principio a fin. De todas formas, cuando se alcance el final del escenario se notificará al alumno que éste ha concluido para que pueda continuar con la simulación de nuevos escenarios. Por supuesto, tampoco dispone de herramientas de creación o de modificación de clases ya que el objetivo de ViRPlay3D es el análisis y la comprensión de aplicaciones ya creadas.

Tampoco se han tenido en cuenta las acciones colaborativas ya que ViRPlay3D se ha desarrollado como una herramienta monousuario. Por supuesto no dispone de comunicación entre usuarios ni se necesita de herramientas de consenso. Sin embargo, sí se incluye la monitorización de las acciones del

alumno y el volcado de esta información a archivos de bitácora. De cara al instructor humano que emplea esta herramienta, estos archivos son de utilidad para el mantenimiento de los escenarios y para evaluar los conocimientos que va adquiriendo el alumno sobre la aplicación. El contenido de estos archivos ayuda a valorar el comportamiento de los alumnos con respecto a un escenario. Por ejemplo, gracias a estos archivos el instructor puede observar si los alumnos hacen un frecuente uso de los inventarios, cuáles son más consultados y, por tanto, necesitan mayor información. También puede observar si los alumnos cometen frecuentemente los mismos errores cuando hacen uso de la interfaz de paso de mensajes —descrita a continuación. Esto indica que será necesario reforzar el análisis de determinadas partes de una aplicación mediante la creación de nuevos escenarios que profundicen en estas partes en las que se han detectado debilidades.

Para finalizar es necesario describir la mecánica relativa al paso de mensajes. Como se describió en el capítulo anterior, esta mecánica se compone del lanzamiento y recogida de la pelota, así como de la creación del mensaje que se desea pasar. En ViRPlay3D las dos primeras acciones se han automatizado ya que los objetos son controlados mediante agentes. Cada uno de estos agentes sabe cómo lanzar la pelota de un objeto a otro, así como recogerla cuando es el destinatario de la misma. En el caso de solicitar el siguiente paso de mensaje, el entorno sabe cuál es el mensaje correcto que ha de ser creado en el paso de simulación actual.

ViRPlay3D contempla que el alumno se responsabilice de la ejecución de un paso de mensajes. Para ello, el alumno dispone de una interfaz con la que seleccionar y configurar el siguiente mensaje que ha de ser pasado. Una vez que el mensaje ha sido creado, el entorno hará que los agentes responsables de los objetos participantes representen el paso de este mensaje. La interfaz de configuración de paso de mensajes es muy sencilla, tal y como se puede ver en la Figura 6.10. El objeto responsable de pasar el mensaje —*Invoker*— viene fijado por el estado actual de la simulación, ya que será el objeto activo. Sin embargo, el receptor del mensaje —*Invokee*— puede ser cualquiera de los objetos que aparecen en el escenario, incluido el propio objeto activo. De acuerdo al objeto receptor seleccionado, el alumno podrá seleccionar de entre un repertorio distinto de mensajes —*Message passing*. Este repertorio se corresponde con los métodos de los que dispone la clase del objeto receptor del mensaje. Aquí también se puede seleccionar si lo que se desea es invocar un método o, por el contrario, enviar un mensaje de retorno indicando que el método ha concluido. En este caso, también se puede incluir el valor devuelto por el mensaje —*Return value*. En caso de invocar un mensaje, el alumno también debe decidir cuál es el valor de los parámetros reales con los que se va a pasar el mensaje —*Parameters*. Por defecto, los parámetros aparecen vacíos y el alumno deberá ir añadiendo los valores para cada uno de los parámetros.

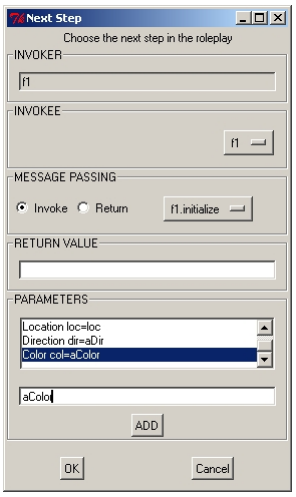


Figura 6.10: Interfaz para la creación de mensajes.

El uso de la interfaz de creación de mensajes es libre. En ningún momento el entorno obliga al alumno a que interprete el rol de un objeto. Pero el alumno puede hacer esto en cualquier momento. Como ya se vio en la Sección 6.1.1, el paso del mensaje creado por el alumno será interpretado sólo en el caso de que éste sea correcto de acuerdo al estado actual del escenario. El entorno será responsable de comparar el mensaje creado por el alumno con el próximo mensaje que se ha de ejecutar. En caso de que los dos mensajes coincidan, se simulará el paso de este mensaje. En caso contrario, se informará al alumno de que este mensaje no es el siguiente que se ha de realizar. El alumno podrá entonces optar por crear un nuevo mensaje o por solicitar que se ejecute el siguiente paso de mensaje. En todo caso, el mensaje creado será guardado en la bitácora de la sesión. Esto puede ser usado por un instructor para evaluar a los alumnos.

6.2. Instanciación de la arquitectura genérica para ViRPlay3D

Siguiendo las decisiones de diseño descritas en las secciones anteriores se ha implementado el primer prototipo de entorno virtual de role-play. La implementación se ha realizado mediante la instanciación de la arquitectura general descrita en el Capítulo 5 y reutilizando parte de la implementación de otra herramienta desarrollada para la enseñanza de la Máquina Virtual y compilación de Java: JV^{2}M (Gómez-Martín, 2008a,b).

JV^{2}M es un sistema educativo que representa de manera metafórica la Máquina Virtual de Java y que propone que el alumno se enfrente a la ejecución de un programa en Java dentro de este entorno metafórico. En

Figura 6.11: Aspecto del sistema educativo JV^{2}M.

la primera versión de este entorno, el alumno hace uso de las mecánicas propias de una aventura gráfica —al estilo de la saga Monkey Island o Grim Fandango, de LucasArts [1]— para interactuar con las entidades del entorno y así completar su tarea de ejecución. Dentro del entorno existe un agente pedagógico llamado Javy al que el alumno puede solicitar ayuda en caso de no saber qué es lo que ha de realizar. En la Figura 6.11 se puede ver el aspecto de JV^{2}M.

Para comprender la instanciación realizada en ViRPlay3D hay que comprender parte de la arquitectura seguida en JV^{2}M. Esta arquitectura, que se pueden ver en la Figura 6.12, se compone de cuatro subsistemas fundamentales:

- Subsistema de tutoría. Es el encargado de tomar las decisiones pedagógicas en JV^{2}M en base al conocimiento del dominio, al perfil del alumno y a las acciones realizadas por éste dentro del entorno virtual.
- Vista lógica del mundo. Almacena la parte del estado del mundo que es relevante desde el punto de vista educativo. Sirve de interfaz de unión entre el subsistema de tutoría y el entorno virtual.
- Entidades del entorno. Se encarga de gestionar la apariencia y el comportamiento de los elementos dinámicos que aparecen en el entorno virtual y que reaccionan ante las acciones del alumno o de otras entidades. Ya que algunas de estas entidades tienen su representación en la vista lógica, a este módulo también se le conoce como *módulo gestor de objetos del interfaz* o módulo `OIM` (*Object Interface Manager*).
- Motor de la aplicación. Es el módulo de más bajo nivel y se encarga

[1] En el Museo de LucasArts se puede ver el aspecto de estas aventuras gráficas: `http://lucasarts.vintagegaming.org/` (último acceso: 15-02-2008)

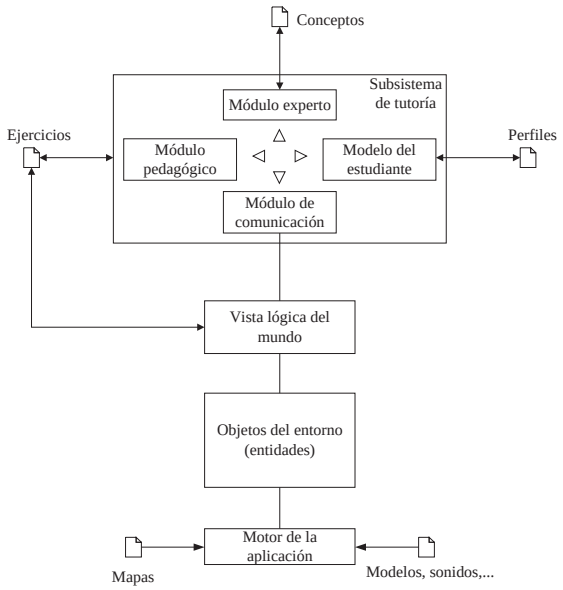


Figura 6.12: Arquitectura de JV^{2}M.

de proporcionar las tareas básicas dependientes del sistema operativo o del motor de juegos empleados. Proporciona módulos para la gestión de la memoria, ficheros, sistemas de log, motor gráfico, gestión y carga de mapas, gestión de entrada, interfaz de usuario y motor de sonido.

El desarrollo de ViRPlay3D usando JV^{2}M ha facilitado la rápida implementación del primero gracias a que el segundo ha servido como un armazón encargado del control de la ejecución, de la gestión de las entidades del entorno virtual, de la gestión de la entrada de usuario y visualización en 3D usando el motor de videojuegos Nebula 1. En la Figura 6.13 se pueden ver los módulos de la arquitectura general que han sido implementados. También en esta figura se puede observar que, debido a las decisiones de diseño adoptadas, algunos módulos no han sido incluidos en este prototipo —están representados en rojo— mientras que para la implementación de otros se han utilizado módulos propios de JV^{2}M —dichos módulos están en azul.

Para el desarrollo de ViRPlay3D se ha desechado toda la parte relacionada con el subsistema de tutoría y la vista lógica del mundo, ya que son elementos fuertemente dependientes del dominio que se está enseñando. Sin embargo, se ha reaprovechado gran parte de todo lo que está relacionado con el entorno virtual en sí mismo.

Para el desarrollo de la *Interfaz del alumno* se ha empleado gran parte del motor de la aplicación de JV^{2}M. Esto implica que se ha empleado el motor de juegos Nebula1 para la gestión de entrada de usuario, para el dibujado en 3D y para parte de la interfaz en 2D del usuario. La interfaz de los inventarios

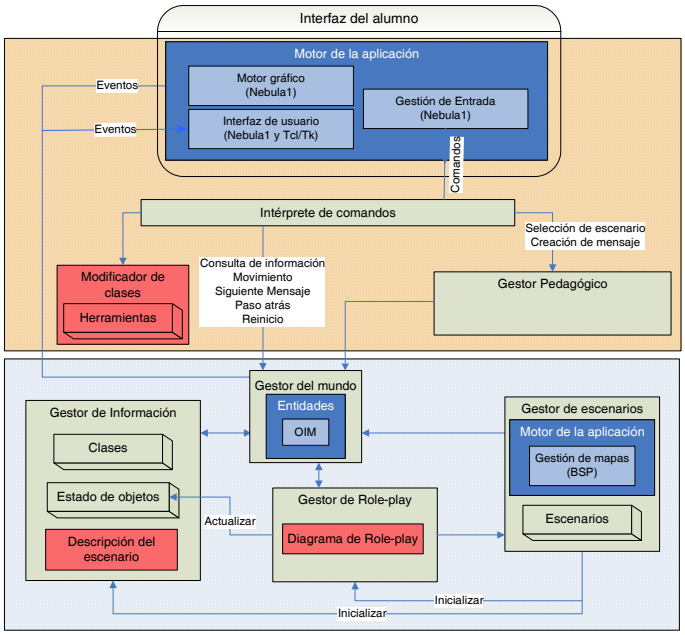


Figura 6.13: Instanciación de la arquitectura general de los entornos virtuales de role-play para el prototipo ViRPlay3D.

y de la herramienta de creación de mensajes han sido desarrollados usando el lenguaje de script Tcl (integrado en Nebula1) y su librería Tk.

El *Intérprete de comandos* ha sido desarrollado *ex-profeso* para ViRPlay3D. Este intérprete es el responsable de distribuir los comandos que llegan desde el módulo *Gestor de entrada* a los distintos módulos que conforman el entorno virtual. Todos aquellos comandos relacionados con el movimiento del avatar del alumno, así como las acciones *Mirar a* y las acciones relacionadas con el control de la ejecución del escenario son enviadas directamente al *Gestor del mundo*. Sólo hay una excepción: si el alumno decide interpretar el rol del objeto activo y usa la interfaz de creación de mensajes, este comando es enviado al *Gestor pedagógico*, quien se hará responsable de decidir qué hacer con él. A este módulo también le llegan las acciones relacionadas con la selección de escenarios del alumno.

El *Gestor pedagógico* ha sido desarrollado de manera muy simple para este primer prototipo. Como ya se ha señalado, cuando el alumno interpreta el rol del objeto activo y decide cuál es el siguiente mensaje que se ha de enviar, el entorno ejecutará este paso si es el correcto, o informará al alumno de su error, en caso de que éste no fuese el siguiente paso. El módulo responsable de esta comprobación es el *Gestor pedagógico*. Una vez que este módulo recibe el comando con el mensaje creado por el alumno solicita al *Gestor de role-play*, a través del *Gestor del mundo*, cuál es el siguiente mensaje de la simulación. El *Gestor pedagógico* compara ambos mensajes y, si es correcto, solicita al

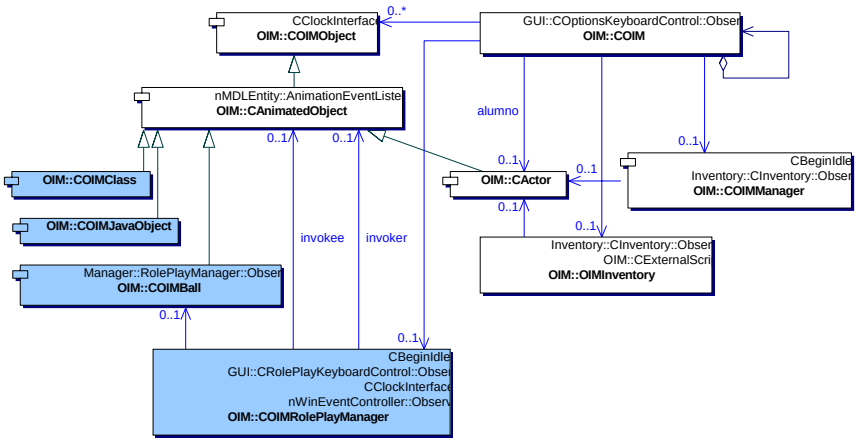


Figura 6.14: Entidades añadidas (en azul) al `OIM` de JV^{2}M para crear el *Gestor del mundo* de ViRPlay3D.

Gestor de role-play que ejecute el siguiente mensaje. En caso contrario, el *Gestor pedagógico* se comunica con el *Gestor del mundo* para, a través de la interfaz de usuario, informar al alumno del error cometido.

El *Gestor del mundo* ha sido desarrollado utilizando toda la arquitectura de clases desarrolladas para el `OIM` de JV^{2}M. Como se puede ver en la Figura 6.14, a este módulo tan solo ha sido necesario añadir las clases que representan a las entidades relacionadas con los entornos virtuales de role-play. La inicialización de estas entidades se realiza en base al escenario seleccionado por el alumno y, de acuerdo a esto, a la información proporcionada por el *Gestor de escenarios*, más concretamente por el Gestor de mapas.

El *Gestor del mundo* también se encarga de la gestión de los inventarios de estas nuevas entidades y de solicitar la información que han de presentar al *Gestor de Información*. Por último, este módulo es responsable de propagar los eventos con los que la *Interfaz del alumno* será actualizada. Los eventos propagados están relacionados con los movimientos de las entidades —cambio de posición y giro—, activación de los inventarios y presentación de información del entorno —mensajes sobreimpresos de paso de mensajes y de error.

El desarrollo de la *Interfaz del alumno* en base al motor de la aplicación de JV^{2}M ha obligado a reutilizar gran parte de los formatos de mapas de escenarios y de modelos que fueron desarrollados para ese sistema educativo. Como ya se verá en la Sección 6.3, los escenarios de ViRPlay3D se componen de información relacionada con la ubicación inicial de las entidades dentro del mundo virtual más la información propia de las clases, objetos y pasos de mensajes que compondrán la simulación del escenario. Para la primera se ha reutilizado el Gestor de Mapas de JV^{2}M. Este gestor se encarga de cargar

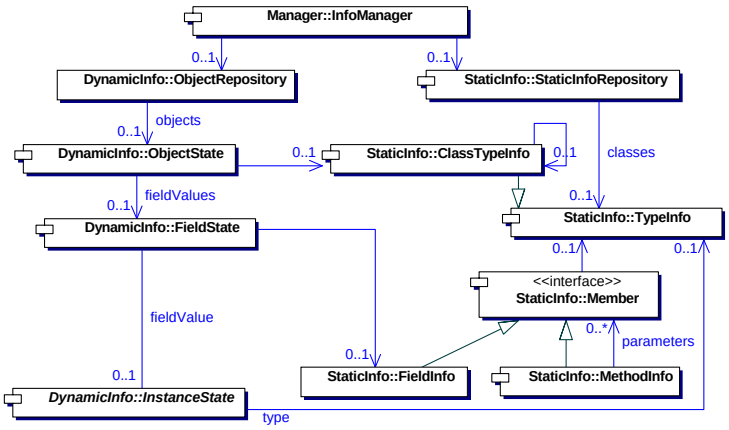


Figura 6.15: Implementación del *Gestor de información* en ViRPlay3D. Contiene tanto la información estática como dinámica.

las entidades procedentes de un archivo BSP, un formato muy popular en el desarrollo de mapas para videojuegos comerciales como Half Life o Quake. Este gestor sólo ha tenido que sufrir ligeras modificaciones para que también sea capaz de cargar los nuevos tipos de entidades que existen en ViRPlay3D.

Sin embargo, el módulo *Gestor de escenarios* ha necesitado de la creación de cargadores específicos para el resto de información. Como se verá más adelante, las herramientas de autoría para este tipo de información generan archivos en XML sobre la información estática de las clases que aparecen en el mundo, así como sobre los pasos de ejecución y los cambios de estado de los objetos que intervienen en el escenario. Para toda esta información se han creado dos analizadores para este tipo de archivos. Estos analizadores se encargarán de inicializar la información contenida en el *Gestor de información* y de crear para el *Gestor de role-play* el guión con la secuencia de acciones que compondrán la simulación del escenario.

El *Gestor de información* ha sido desarrollado íntegramente para ViRPlay3D. Como se puede observar en la Figura 6.15, este módulo se compone de dos repositorios: uno para la información estática y otro para lo referente al estado de los objetos. Cada uno de los repositorios dispone de operaciones de consulta y actualización necesarias para el tipo de información contenida. Por ejemplo, el repositorio de objetos dispone de operaciones para derreferenciar objetos y modificar el estado de algunos de sus atributos, mientras que el repositorio de información estática proporciona operaciones de consulta sobre los métodos, atributos y parámetros de una clase. Como se puede ver en la Figura 6.13, no existe ningún repositorio para la descripción del escenario ya que esta información se ha obviado, como ya se explicó en anteriores Secciones.

Al igual que el anterior, el *Gestor de role-play* ha sido desarrollado ín-

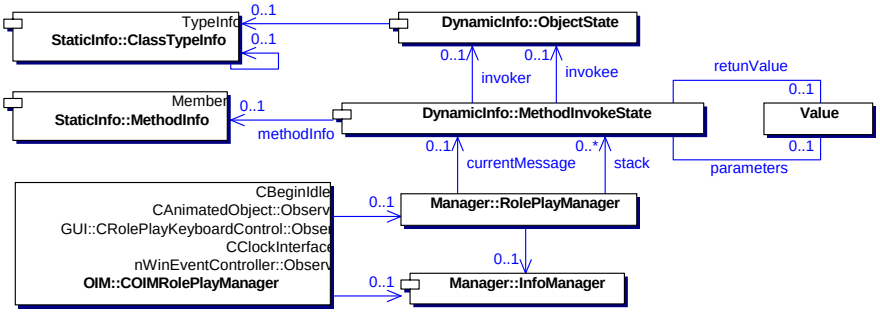


Figura 6.16: Implementación del *Gestor de role-play* en ViRPlay3D.

tegramente para ViRPlay3D, ya que éste no existe en JV^{2}M. Este módulo (Figura 6.16 contiene la secuencia de pasos de mensajes del escenario que va a ser simulado. De acuerdo a esta lista de pasos, el *Gestor de role-play* responde a los comandos de siguiente mensaje, paso atrás o reinicio provenientes del intérprete de comandos. Así mismo, podrá proporcionar al *Gestor pedagógico* cuál es el siguiente paso de mensaje para que aquél actúe en consecuencia. Cada uno de los pasos de mensaje puede contener información relacionada con el cambio de estado de alguno de los objetos del escenario. En este caso, el *Gestor de role-play* se comunicará con el repositorio de objetos contenido en el *Gestor de información* para actualizar los objetos convenientes. Así mismo, el *Gestor de role-play* guarda información sobre el mensaje actual. Esta información está directamente relacionada con la entidad de la pelota que hay en el *Gestor del mundo*.

Por último, el *Gestor de role-play* es el responsable de comunicarse con el *Gestor del Mundo* para que se produzca la simulación visual del paso de mensajes entre dos objetos del mundo. Esta simulación visual se compone de los siguientes pasos:

- Las entidades que representan a los objetos invocador e invocado han de girar sobre sí mismas hasta que queden encaradas.
- El invocador "habla": la interfaz de usuario ha de presentar el diálogo con el mensaje que va a ser pasado.
- El invocador lanza la pelota. La entidad que representa a la misma irá cambiando de posición, representando visualmente un tiro parabólico entre invocador e invocado.
- La entidad que hace las veces de objeto invocado recoge la pelota y se queda con ella una vez que ésta llega hasta su destino.

Por tanto, el *Gestor de role-play* será el responsable de coordinar todo este proceso. Una vez concluido, realizará la actualización de los objetos y

actualizará sus referencias al objeto activo y al siguiente paso de ejecución que se ha de producir.

6.3. Herramientas de autoría para ViRPlay3D

Los entornos virtuales de role-play se basan en la simulación de escenarios ricos en información. A modo de resumen de lo visto en la Sección 5.6, para cada simulación se necesita crear la siguiente información:

- Mapa. Para cada escenario hay que describir qué entidades (clases, objetos, avatar del alumno...) hay en el mundo y cuál es su ubicación inicial dentro de él. Toda esta información se encuentra contenida en el mapa del escenario.

- Información estática. Contiene la información relacionada con las clases de objetos que participan en la simulación y está accesible desde los inventarios de los objetos y de las clases.

- Información de ejecución. Describe cómo se van a comportar los objetos que intervienen en la simulación a medida que el alumno solicita el siguiente paso de ejecución o decide interpretar el papel del objeto activo. Además, esta información se usa para ir actualizando el estado de los objetos tras cada paso de ejecución.

La cantidad de información para cada simulación es grande y cada módulo de información tiene su propio formato. Por tanto es necesario acompañar a ViRPlay3D de las herramientas necesarias para que un instructor pueda fácilmente desarrollar nuevos escenarios de ejecución.

Para este primer prototipo se ha decidido desarrollar herramientas de autoría que sirven para extraer escenarios de ejecución de casos de estudio ya desarrollados. Como se verá en la Sección 6.4, los escenarios usados en ViRPlay3D han sido extraídos del caso de estudio MBSCS. Este caso de estudio tiene la particularidad de estar extensamente documentado y de tener ya descritos varios escenarios de ejecución de cierto interés pedagógico. Gracias a esto se decidió desarrollar herramientas que extrajeran toda la información necesaria para el desarrollo de los escenarios de ejecución del código fuente de este caso de estudio. Estas herramientas no se han implementado *ad-hoc* para este caso de estudio sino que se han construido con la intención de que puedan ser empleadas para extraer información del código fuente de cualquier caso de estudio que pueda ser de utilidad para la creación de escenarios de ViRPlay3D.

A continuación describiremos cada una de las herramientas de autoría empleadas.

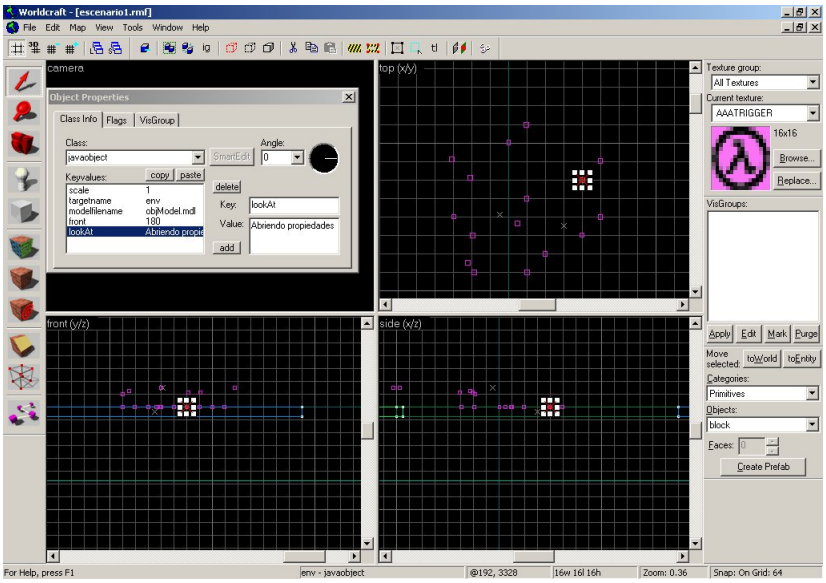


Figura 6.17: Desarrollo de un mapa con *WorldCraft*. En la imagen aparecen las propiedades para el objeto *env*, perteneciente al MBSCS.

6.3.1. Autoría de mapas

La primera versión de ViRPlay3D se ha implementado haciendo uso de JV^{2}M como ya ha quedado patente en la Sección 6.2. En esta herramienta se decidió solventar el problema de la adquisición de recursos gráficos aprovechando la gran capacidad de desarrollo que han tenido algunas comunidades relacionadas con videojuegos que facilitan la personalización de niveles y escenarios de juego como Half Life. Por tanto, los escenarios han sido desarrollados empleando la misma herramienta utilizada en JV^{2}M: *WorldCraft*.

WorldCraft —también conocido como *Valve Hammer Editor*— es una herramienta desarrollada por Valve para la creación de mapas para videojuegos comerciales como Quake o Half Life. El desarrollo de mapas en *WorldCraft* es sencillo: se basa en la creación de un mundo compuesto por sólidos inanimados —llamados brochas o *brushes*— y entidades que se mueven, tienen sonido o con las que se puede interactuar. Las entidades pueden ser de distintos tipos y cada una dispone de una serie de pares atributo-valor y de flags que servirán para definir la representación y el comportamiento de la entidad dentro del juego. *WorldCraft* guarda los mapas desarrollados tanto en formato binario como en texto. Estos mapas necesitan varias fases de postprocesado para que queden en formato BSP —del inglés *Binary Space Partitioning*—, que permite el renderizado más rápido de la escena minimizando el número de polígonos que hay que dibujar cada vez que se realiza el refresco de pantalla.

El desarrollo de escenarios en ViRPlay3D es muy sencillo ya que el número de entidades creadas es limitado y la "geografía" del escenario es muy simple. Como se puede ver en la Figura 6.17, el instructor que desea crear un escenario nuevo parte de un mundo sencillo compuesto con un plano en el que tan solo ha de preocuparse por colocar las entidades que compondrán el escenario. Para cada entidad ha de definir su clase —*javaobject* si es un objeto, *clase* si va a representar a una clase, *ball* para representar la ubicación inicial de la pelota o *punto_inicio_estudiante* para especificar la ubicación inicial del avatar del estudiante—, el nombre que tendrá en la simulación —*targetname*—, el modelo gráfico que va a representar a esta entidad —*modelfilename*—, su orientación —*front*— y su escala —*scale*. Una vez completado el escenario, se guarda y se postprocesa mediante un proceso por lotes que genera el escenario final que se va a usar en la simulación.

6.3.2. Autoría de información estática

Como se ha visto en la Sección 6.1.3, la información estática necesaria para ViRPlay3D se compone del código y de la documentación de las clases del caso de estudio empleado para el desarrollo de los escenarios. De nuevo, en lugar de obligar al instructor a crear toda la documentación desde cero, se ha desarrollado una herramienta para la extracción de la documentación y su conversión a un formato válido para ViRPlay3D.

Para que un caso de estudio sea verdaderamente útil ha de estar bien documentado. Esto facilita que aquél que haga uso de él pueda rápidamente comprender su diseño y su funcionamiento. Valiéndonos de esta propiedad de los casos de estudio, se ha desarrollado una herramienta que procesa el código fuente de una aplicación en busca de los comentarios incluidos por los desarrolladores. Estos comentarios son extraídos, procesados y convertidos al formato desarrollado para los escenarios de ViRPlay3D.

Para la extracción de la documentación se ha empleado la herramienta *javadoc* para la generación de documentación en HTML de una aplicación. Para modificar el comportamiento de un *javadoc* se crean *doclets*, pequeñas utilidades que procesan el código fuente y deciden cómo componer los archivos generados por *javadoc*.

En ViRPlay3D se ha empleado una modificación del *doclet* JelDoclet [2]. Este doclet genera un XML con toda la información sobre las interfaces de las clases, sus atributos y métodos y toda la documentación asociada a los mismos. Este *doclet* genera información que no es de utilidad para ViRPlay3D por lo que se ha decidido hacer algunas modificaciones sobre el mismo. La estructura del XML generado por nuestra herramienta se puede ver en la Figura 6.18.

Para generar la información asociada a un escenario el instructor necesita

[2] Accesible en `http://jeldoclet.sourceforge.net/` (último acceso: 15-02-2008)

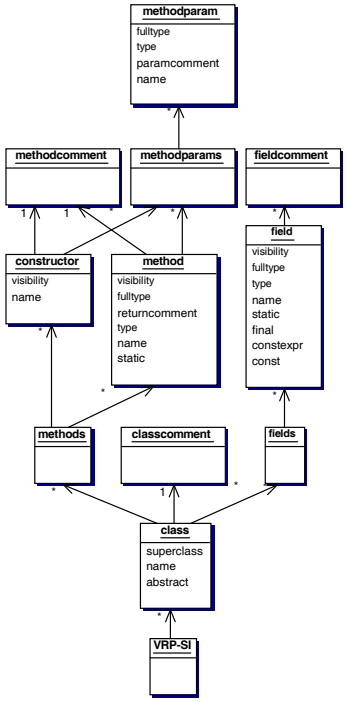


Figura 6.18: Diagrama de la estructura del XML que contiene la información estática de los escenarios de ViRPlay3D.

el código fuente del caso de estudio. Posteriormente, tan solo ha de lanzar un proceso por lotes que ejecuta la herramienta *javadoc* con el *doclet* desarrollado y genera el archivo XML que contiene toda la información estática del escenario. Esta información sólo es necesario generarla una vez si el mismo caso de estudio va a ser empleado para el desarrollo de distintos escenarios ya que éstos compartirán la misma información estática.

6.3.3. Autoría de información de ejecución

Para la ejecución completa de un escenario hace falta tener un guión que contenga todos los pasos de ejecución de los que se compone un escenario. Este guión también ha de almacenar los cambios del estado que se produzcan en los objetos que participen en la simulación. Al igual que con el resto de información, el desarrollo desde cero de esta documentación puede convertirse en un trabajo tedioso.

Al igual que con la información estática, vamos a aprovechar el hecho de que los casos de estudio ofrecen el código fuente para elaborar los guiones de simulación. De la ejecución del caso de estudio se va a extraer toda la información dinámica necesaria para los escenarios de ViRPlay3D. Para ello,

se decidirá cuál es el método inicial que se desea ejecutar y, a partir de aquí, se almacenará una traza con información de algunos de los mensajes que se vayan pasando entre los objetos, así como la información sobre los cambios de estado relevantes que se produzcan.

Ya que se dispone del código fuente, podríamos pensar en modificar el mismo para extraer toda la información que necesitemos. Esto haría que cada vez que deseemos crear un escenario tengamos que cambiar el código fuente del caso de estudio y añadir nuestras líneas de código para extraer esta información. Esta alternativa es intrusiva, ya que debemos modificar el código fuente del caso de estudio, y sigue siendo casi tan tediosa como crear toda esta información desde cero.

Como alternativa se ha desarrollado una pequeña herramienta que hace uso de la programación orientada a aspectos para reducir la intrusión dentro del código del caso de estudio. La programación orientada a aspectos (Kiczales et al., 1997) facilita la modularización avanzada de una aplicación. En una aplicación suele existir cierta funcionalidad que queda distribuida por todo el código fuente de la aplicación —como, por ejemplo, la creación de logs de ejecución, similares a nuestros intereses. La programación orientada a aspectos permite agrupar toda esta funcionalidad en un solo módulo llamado *aspecto*, independiente del resto del sistema y que se integrará con el resto de código en la compilación o durante la ejecución de la aplicación. Adicionalmente, es necesario establecer cuáles son los puntos en los que se ejecutarán determinados métodos del aspecto. Estos puntos son los que se conocen como puntos de corte —del inglés *pointcut*.

La herramienta desarrollada facilita la generación de los archivos de información dinámica. Para ello tan solo hay que definir en un archivo qué objetos van a intervenir en la simulación, con qué paso de mensaje se inicia y con cuál se termina la simulación y qué pasos de mensaje intermedios se desea trazar. Esta información se utiliza para generar un aspecto de extracción de información dinámica, especializando los puntos de corte en los que el aspecto va a ser ejecutado. Este extractor se compila y ejecuta la aplicación. A medida que la ejecución avanza, el extractor genera un archivo XML que contiene la información de ejecución que necesitamos y que sigue la estructura representada en la Figura 6.19.

6.4. Ejemplo de uso de ViRPlay3D

Para ilustrar el comportamiento de ViRPlay3D se va a describir la simulación de un escenario de ejecución. Para este entorno se han desarrollado varios escenarios basados en el MBSCS. Un escenario sencillo extraído de este caso de estudio y con el que vamos a ilustrar el comportamiento de ViRPlay3D es el que muestra cómo se realiza la inicialización de un pez (Jiménez-Díaz et al., 2007a).

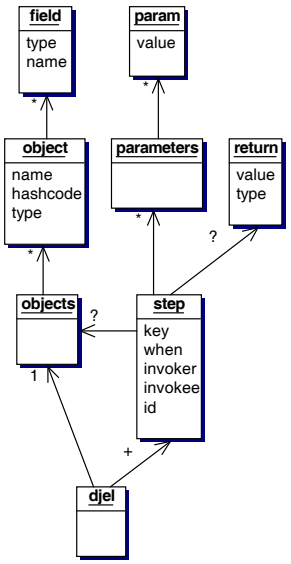


Figura 6.19: Diagrama de la estructura del XML que contiene la información de ejecución de los escenarios de ViRPlay3D.

Antes de describir el escenario describiremos brevemente las clases y objetos que intervendrán en el escenario. La clases principales del MBSCS y que son de interés para nuestro ejemplo son las siguientes:

`Environment.` Esta clase modela una rejilla rectangular que contiene objetos. Esta clase tiene dos subclases: `BoundedEnv` y `UnboundedEnv`, que representan un entorno con y sin límites, respectivamente. Por simplificación, se ha considerado que la clase `Environment` sea equivalente a la clase `BoundedEnv`. A esta clase se añadirán aquellos objetos que han de ser representados en la simulación. Proporciona métodos para la navegación dentro del entorno y es capaz de generar una dirección aleatoria para la inicialización de un objeto que se moverá por el entorno.

`Fish.` Esta clase es la que representa a cada uno de los peces que se moverán por el entorno. Cada objeto de esta clase dispone de un número identificador, un color que se usa para su representación dentro del entorno, su localización dentro del entorno y la dirección en la que está actualmente mirando. Además, guarda información sobre el entorno en el que está contenido para poder navegar por él.

`Location.` Contiene información sobre la posición —fila, columna— de un objeto dentro del entorno.

`Direction.` Contiene información sobre la dirección a la que mira un objeto dentro del entorno.

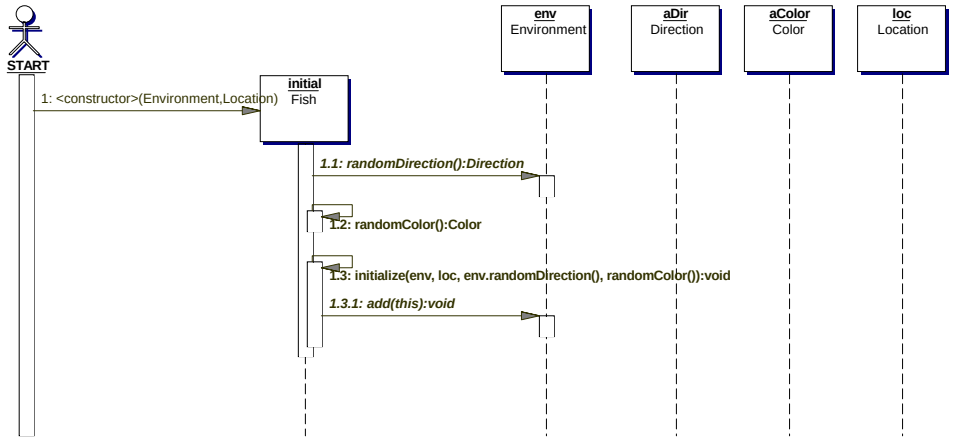


Figura 6.20: Diagrama de secuencia que representa el escenario que va a ser representado.

`Color`. Esta clase es propia de Java y contiene información sobre un color en RGB.

El escenario que va a ser simulado representa la inicialización de un objeto de la clase `Fish`. En la Figura 6.20 se puede ver un diagrama de secuencia en el que se observan las interacciones que se van a producir dentro del escenario. En el escenario desarrollado habrá un objeto para cada una de las clases descritas anteriormente. De aquí en adelante estos objetos se llamarán por su nombre: *f1*, de la clase `Fish`; *env*, de la clase `Environment`; *loc*, de la clase `Location`; *aColor*, de la clase `Color` y *aDir*, de la clase `Direction`.

De acuerdo a la metáfora descrita en la Sección 6.1.2, en el escenario aparecerán cinco avatares, cada uno de ellos representando a un objeto, y cinco bloques que representan a sendas clases. En el centro del escenario se encuentra un avatar más que representa al alumno y que será controlado por éste. Mediante este avatar, el alumno puede interaccionar con el resto de objetos para así conseguir información relevante para comprender el escenario. Además, en el escenario aparece una pelota. Ésta es sostenida por *f1*, ya que es el objeto activo inicial.

Supongamos que el alumno desea saber cuál es el estado en el que inicialmente se encuentra *f1*. Para obtener esta información puede consultar el inventario del avatar que representa a este objeto. Para ello, el alumno se acerca con su avatar hasta el de *f1*. Cuando esté lo suficientemente cerca aparecerá la acción *Mirar a f1*. Al ejecutarla aparece el inventario de *f1*. Este inventario contiene información sobre la clase a la que pertenece el objeto. A través del inventario se accede al código fuente y a la documentación de la clase `Fish`. Además de la información de la clase, el alumno tiene acce-

so al estado del objeto. Puede ver una lista con los valores actuales de los atributos de *f1*:

- *myId*. Es el identificador del pez y su valor inicial es 0.
- *myLoc*. Contiene la posición dentro del entorno. Está inicializado a NULL.
- *myDir*. Contiene la dirección hacia la que nada el pez y su valor actual es NULL.
- *myColor*. Representa el color del pez y su valor actual es NULL.
- *theEnv*. Es el entorno en el que se encuentra el pez. Su valor al iniciar el escenario es NULL.

Al igual que con este objeto, el alumno realiza lo mismo con cada uno de los objetos presentes en la escena, para así saber cuál es el estado inicial de cada uno de ellos. También puede hacer lo mismo con los bloques que representan las clases. Por ejemplo, si el alumno se acerca al bloque que tiene la etiqueta `Environment`, aparecerá la acción *Mirar a Environment*. Al ser ejecutada aparece el inventario de esta clase, desde donde el alumno podrá consultar la documentación asociada a la clase así como su código fuente.

Tras esto, el alumno decide solicitar el primer paso de mensajes. Entonces, los avatares *f1* y *env* se encaran y *f1* le pasa el mensaje *randomDirection* a *env*. El primero dice —el diálogo aparece sobreimpreso en pantalla para que lo vea el alumno—-: *"env, execute randomDirection"* y lanza la bola hacia *env*. La bola vuela por el aire hasta que es recogida por aquél. Si de nuevo solicita el siguiente paso de ejecución, *env* invocará el retorno del mensaje anteriormente pasado, diciendo*"randomDirection is done, returning aDir"* y haciendo rodar la bola hasta *f1*.

Si el alumno acerca su avatar hasta *f1* verá que puede realizar la acción de mirar tanto a *f1* como a la pelota. Si selecciona esta última, aparecerá un inventario con información sobre el último paso de mensaje realizado. En particular, este inventario contiene una descripción del método invocado (extraída de la documentación de las clases), quién ha sido el invocador (*f1*) y el objeto sobre el que ha sido invocado (*env*), los valores actuales de los parámetros (este método no tenía parámetros) y, ya que el método tenía retorno y éste se ha efectuado, el valor que se devolvió (*aDir*).

De nuevo, se realiza el siguiente paso de ejecución. En este caso, invocador e invocado son *f1*, que va a llamar a su propio método para generar un color aleatorio. En este caso, este avatar lanza la pelota hacia arriba y en vertical y dice *"f1, execute randomColor"*. Con el siguiente paso de ejecución se puede ver que la autoinvocación de este método ha servido para generar

un objeto de la clase `Color`, ya que se realiza el retorno de este método diciendo *"randomColor is done, returning aColor"*. Si el alumno se acerca al objeto *aColor* podrá ver cuál es su estado y, en particular, cuáles son los valores R, G y B, que caracterizan el color del pez.

Ahora supongamos que el alumno decide tomar el control de la ejecución de *f1*. Para ello, usa la interfaz de configuración de paso de mensajes. Supongamos que el alumno decide pasar el mensaje *add* al objeto *env*. Para ello, selecciona a *f1* como invocador del mensaje, a *env* como receptor, selecciona el método *add* y como valor del parámetro *obj* escribe el valor *f1*. Además, el tipo de mensaje lo pone en modo invocación. Una vez realizado esto, el sistema muestra un mensaje indicando que el mensaje seleccionado no es correcto. Al solicitar el siguiente paso de mensaje, el alumno verá que el mensaje correcto era la invocación del método *initialize* sobre el propio objeto *f1* y con los siguientes valores para los parámetros actuales: *env*, *loc*, *aDir* y *aColor*.

Si tras este mensaje el alumno vuelve a consultar el inventario del objeto *f1* verá que los atributos de este objeto han cambiado y que han sido inicializados a los valores de los parámetros que han sido pasados en la invocación del método *initialize*. Además, el alumno puede revisar el inventario de estos objetos para conocer el estado concreto de *f1*, ya que estos objetos aparecen en el escenario. En particular, si el alumno consulta el inventario de *env* verá que tiene un atributo llamado *objectCount* que lleva la cuenta de los objetos que hay en el entorno y que actualmente se encuentra a cero.

El alumno vuelve a solicitar el paso del siguiente mensaje: la pelota volará de *f1* a *env* para representar el paso del mensaje *"env, execute add with f1"*. Tras este mensaje el alumno vuelve a consultar el inventario del objeto *env*. En este caso puede comprobar que el estado ha cambiado, de modo que ahora *objectCount* ha sido actualizado a 1, lo que implica que el pez ha sido añadido. Por último se ejecutarán los mensajes de retorno de los métodos *add* e *initialize*. Si el alumno vuelve a solicitar el paso del siguiente mensaje aparecerá un mensaje que le indica que la ejecución ha terminado, por lo que ya no quedan más mensajes que pasar.

6.5. Evaluación de ViRPlay3D

Una vez concluido el desarrollo de ViRPlay3D se ha realizado una pequeña evaluación del mismo en la que han participado 12 profesores de la Facultad de Informática de la Universidad Complutense de Madrid, 7 de los cuales han estado vinculados en alguna ocasión con asignaturas relacionadas con la programación y/o el diseño orientado objetos. Antes de la realización de esta experiencia, un tercio de los participantes desconocía la técnica del role-play en el ámbito del diseño orientado objetos. Sin embargo, de los dos tercios que sí sabían de la existencia de esta técnica, sólo uno la había em-

pleado en sus clases. Esta evaluación persigue los siguientes objetivos:

- Obtener una valoración de la metáfora empleada.
- Analizar la facilidad de uso del entorno.
- Valorar la utilidad de la información contenida en el entorno, así como las mecánicas de interacción dentro del mismo.
- Obtener una retroalimentación sobre mejoras que se puedan realizar sobre este tipo de entornos.

Para la realización de esta evaluación se reunió al grupo para que ejecutaran, de manera guiada, el escenario de simulación descrito en la Sección 6.4. Inicialmente se les dio una breve explicación sobre que es el role-play y su utilización en la enseñanza de diseño orientado a objetos. Posteriormente se pasó a describir cuál era el escenario de ejecución con el que iban a trabajar en el entorno. Posteriormente se les fue guiando en el uso de ViRPlay3D. Inicialmente se describieron los elementos existentes en el entorno y las mecánicas principales del mismo. Posteriormente se realizó la ejecución paso a paso del escenario utilizando como ayuda de guía el diagrama de secuencia de la Figura 6.20. Finalmente, los participantes rellenaron los cuestionarios con los que han valorado distintos aspectos de la herramienta. Para cada aspecto se han definido un conjunto de elementos, cada uno de los cuales se ha puntuado utilizando una escala de Likert de 5 puntos. Para la valoración de la metáfora y de la representación visual de la misma dentro del entorno se ha medido la naturalidad de sus principales elementos. La información contenida en el entorno y las mecánicas de interacción se han evaluado de acuerdo a su utilidad para la comprensión del diseño y de las interacciones producidas entre los objetos. Sin embargo, la usabilidad de ViRPlay3D se ha evaluado midiendo el grado de conformidad del usuario con afirmaciones del estilo:

- *"El entorno me ha resultado fácil de usar."*
- *"He sabido llegar hasta la información siempre que lo he necesitado."*
- *"Me he divertido usando el entorno virtual."*
- *"La herramienta me ha ayudado a comprender las interacciones que se producen entre los objetos del caso de estudio empleado en el escenario de simulación."*
- *"Emplearía este entorno virtual como apoyo a mis clases relacionadas con la orientación a objetos."*

Por último, se ha dejado que los participantes comentaran qué elementos les han resultado especialmente interesantes así como cuáles no les han gustado y se recogieron otro tipo de sugerencias para mejorar futuras versiones del entorno.

Los resultados obtenidos de esta evaluación han sido altamente satisfactorios. El grado de satisfacción general de los encuestados con el uso del entorno ha sido alto —una valoración media de 27,75 puntos en un rango de puntuaciones que oscila entre 7 y 35 puntos. Más del 75 % ha considerado que el entorno es fácil de utilizar y que se han divertido durante la sesión. Aunque el 25 % se han sentido ocasionalmente elementos pasivos dentro del entorno, dos tercios de los participantes emplearían este entorno en sus clases de orientación a objetos. Por último, el entorno se ha valorado mejor como medio para comprender las interacciones que se producen entre objetos —una puntuación media de 4,33 sobre 5 puntos— con respecto a su uso para entender el diseño de clases de una aplicación —3,92 de puntuación media—, como era de esperar de acuerdo al objetivo pedagógico para el que ha sido desarrollado.

La metáfora ha sido evaluada de acuerdo a los elementos que se pueden ver en la Tabla 6.1. La valoración general de la metáfora también ha sido muy alta ya que la puntuación media de todos sus elementos ha sido de 38,42 sobre una escala de va de 9 a 45 puntos. Observando los resultados de la Tabla 6.1 se puede apreciar que las valoraciones más bajas se han obtenido en la metáfora del estudiante y de los objetos, así como en la empleada para visualizar el retorno de un mensaje. De los comentarios de los encuestados se ha podido extraer que las valoraciones más bajas se deben a que no están de acuerdo con que todos los objetos tengan la misma apariencia. Además, también han comentado que la metáfora de retorno de mensajes no es clara cuando el retorno se produce por la autoinvocación de un mensaje. De ahí que esta metáfora haya recibido una valoración menor. Las metáforas mejor valoradas son las relacionadas con la utilización del lanzamiento de la pelota como medio para representar el paso de mensajes.

Con respecto a la información contenida en ViRPlay3D, se ha observado que, aunque la valoración general de su utilidad es alta —19,2 puntos de un rango de puntuaciones entre 5 y 25 puntos— las peores puntuaciones se han obtenido en la información contenida en las clases, como se observa en la Tabla 6.2. Como se pudo constatar en la valoración de la experiencia presencial descrita en el Capítulo 4, el empleo del código fuente para saber qué es lo que una clase puede hacer no resulta especialmente útil. De los comentarios también se ha podido extraer que el aspecto de la documentación de código dentro de los inventarios es mejorable ya que se debería resaltar los elementos más importantes de esta información. Por último, se puede observar que ha sido un acierto presentar la ayuda visual del mensaje que está siendo pasado, ya que es el tipo de información mejor valorada.

Tabla 6.1: Distribución de las puntuaciones y puntuación media de los elementos de la metáfora de ViRPlay3D. Los elementos aparecen ordenados de mayor a menor puntuación media.

Elementos de la metáfora	**N**	**Frecuencia**					**Media**	**Desviación típica**
		1	2	3	4	5		
Objeto activo	12	0	0	0	3	9	4.75	0.45
Paso de mensajes	12	0	0	0	3	9	4.75	0.45
Autoinvocación de un mensaje	12	0	0	1	3	8	4.58	0.67
Inventarios	12	0	0	1	7	4	4.25	0.62
Clase	12	0	1	0	7	4	4.17	0.83
Mensaje actual	12	0	3	0	1	8	4.17	1.34
Objeto	12	0	1	1	7	3	4.00	0.85
Retorno del paso de mensajes	12	1	0	1	7	3	3.92	1.08
Alumno	12	2	0	1	4	5	3.83	1.47

Por último se ha valorado las capacidades de interacción dentro del entorno. Como se puede observar en la Tabla 6.3, todas las mecánicas han obtenido valoraciones superiores a 4 puntos de un máximo de 5, donde esta puntuación indica que esta interacción es totalmente necesaria dentro del entorno. Sin embargo, la valoración más baja la ha obtenido la navegación por el entorno en busca de información. Sorprendentemente, los participantes explican en los comentarios que les ha gustado la idea de navegar y curiosear por el mundo para buscar información. Sin embargo, también afirman que al final se hace tedioso tener que acercarse a cada uno de los elementos del entorno para conseguir información.

En los comentarios, los participantes han destacado que el entorno es sencillo e intuitivo y han destacado lo novedoso de dar espacio y forma a las clases y a los objetos que conforman una aplicación. También han encontrado especialmente interesante que el entorno tenga una gran cantidad de información distribuida por el mismo y que esta información sea accesible de acuerdo a sus necesidades.

Dejando a un lado las valoraciones referentes al aspecto gráfico y de manejo del entorno, que no interesan de cara a la evaluación general de la propuesta, las principales críticas al entorno han sido hechas sobre la falta de distinción visual entre los objetos. Los participantes consideran que, aunque la proximidad a las clases ayuda a saber qué avatar representa cada objeto, sería necesario una distinción mayor, de modo que cada objeto tuviera una representación antropomórfica distinta. También han señalado que no hay

Tabla 6.2: Distribución de las puntuaciones y puntuación media de la información contenida en ViRPlay3D. Los elementos aparecen ordenados de mayor a menor puntuación media.

		Frecuencia						
Tipo de Información	**N**	1	2	3	4	5	**Media**	**Desviación Típica**
Mensaje sobreimpreso	12	0	0	0	5	7	4.58	0.51
Estado de los objetos	12	0	0	1	5	6	4.42	0.67
Mensaje en la pelota	12	0	1	2	5	4	4.00	0.95
Documentación de código	12	0	4	2	4	2	3.33	1.15
Código fuente	12	1	5	2	3	1	2.83	1.19

ninguna información visual que indique que se ha producido un cambio de estado de un objeto, lo que obligaba a consultar constantemente el inventario de los objetos para saber cuándo se producía algún cambio en los mismos.

También se ha sugerido la inclusión de información referente a qué interacciones se van a simular y cuál es el estado actual de la simulación. Nos equivocamos al suponer que la simplicidad de estos escenarios no hacía necesario este tipo de información. Los participantes han comentado que el diagrama de secuencia que se usó durante la sesión de evaluación les ayudó a seguir la simulación, por lo que proponen la inclusión de información similar dentro del entorno.

Finalmente, se hicieron algunas sugerencias con respecto a la representación de las clases. Consideran que una clase es como un molde para la creación de objetos y, como tal, debería haber alguna relación visual entre la clase y las instancias de la misma. También afirmaron que hay objetos que aparecían inicialmente en el escenario pero que son creados en medio de la simulación, lo que resulta algo desconcertante. Por último, algunos participantes apuntaron que sería realmente interesante que este entorno trasladara de verdad las sesiones presenciales de role-play. Es decir, que hubiese varios alumnos en los que cada uno de ellos interpretaran el rol de un objeto. Estas apreciaciones son indicativas del interés que ha despertado en los docentes los entornos virtuales de role-play para enseñar diseño orientado a objetos. Esto nos ha conducido al diseño de una nueva instanciación, la cual ponga en práctica el auténtico valor pedagógico de las sesiones de role-play: la cooperación entre alumnos para diseñar de aplicaciones orientadas a objetos.

Tabla 6.3: Distribución de las puntuaciones y puntuación media de las mecánicas de interacción en ViRPlay3D. Los elementos aparecen ordenados de mayor a menor puntuación media.

Mecánicas de Interacción	N	Frecuencia 1	2	3	4	5	Media	Desviación Típica
Mirar a	12	0	0	0	2	10	4.83	0.39
Control de la ejecución	12	0	1	0	0	11	4.75	0.87
Decidir siguiente paso	12	0	2	0	1	9	4.42	1.16
Navegación para buscar información	12	0	1	2	1	8	4.33	1.07

6.6. Descripción general de ViRPlay3D2

Tras la realización de ViRPlay3D y comprobar su aceptación dentro de la comunidad investigadora se decidió diseñar un nuevo entorno virtual de role-play. Este nuevo entorno se conoce como ViRPlay3D 2 (Jiménez-Díaz et al., 2007b,c,d) y pretende cubrir varios objetivos:

- Solucionar las deficiencias encontradas durante la evaluación de ViRPlay3D y añadir las funcionalidades que el anterior entorno dejó en el tintero.
- Definir por completo el subsistema Núcleo de la arquitectura genérica, de modo que permanezca inmutable y sirva de base para la creación de futuros prototipos.
- Desde el punto de vista pedagógico, ViRPlay3D 2 ha sido desarrollado como herramienta de diseño que traslada al entorno virtual una versión simplificada de las experiencias presenciales en la enseñanza de patrones de diseño descritas en el Capítulo 4.

Una de las características fundamentales de los entornos virtuales de role-play que no ha sido implementada en ViRPlay3D es la colaboración entre alumnos. Por tanto, la principal motivación del desarrollo de ViRPlay3D 2 ha sido el diseño del soporte multiusuario de este tipo de entornos. ViRPlay3D 2 ha sido pensado como un entorno de diseño en el que los alumnos pueden simular escenarios de ejecución conducidos por ellos mismos y en los que se pueden utilizar técnicas sencillas de refactorización para modificar el diseño de clases inicialmente propuesto. De esta forma, mientras que ViRPlay3D da soporte a los niveles de participación Visionado, Respuesta, y Cambio y

Construcción, ViRPlay3D 2 soporta también el cuarto nivel de participación, el de Presentación.

En ViRPlay3D 2 cada alumno está obligado a participar en el entorno representando el papel de uno de los objetos que intervienen en el escenario. Ninguno de los alumnos es observador dentro del entorno, como ocurría en ViRPlay3D, sino que van a participar en algún momento a lo largo de la simulación del escenario. Entre todos han de completar el escenario seleccionado. Posteriormente, el escenario será evaluado por el instructor. Para facilitar el trabajo cooperativo de los alumnos, ViRPlay3D 2 dispone de una herramienta de comunicación. Además, las decisiones críticas del escenario deberán realizarse por consenso, a través de las herramientas de votación que ViRPlay3D 2 proporciona.

Los escenarios de simulación de ViRPlay3D 2 son más similares a los que se han desarrollado en las sesiones presenciales descritas en el Capítulo 4. En este sentido, un escenario se puede componer de varias sesiones distintas pero que están relacionadas ya que forman parte del mismo escenario de ejecución. Las modificaciones de diseño producidas durante una sesión se mantienen para las siguientes sesiones asociadas a un escenario.

Además, en ViRPlay3D 2 se ha añadido una componente lúdica al paso de mensajes: cuando un alumno crea un mensaje ha de lanzarlo al objeto destinatario. Para ello se ha añadido la mecánica de lanzamiento de la pelota, en la que el alumno establece hacia dónde desea lanzar la pelota y con qué fuerza. Por otro lado, el objeto destinatario del mensaje deberá recoger la pelota, ya sea cuando se la lanzan o si la pelota cae al suelo.

También se ha modificado parte de la interacción con otros avatares de acuerdo a las críticas en la evaluación de ViRPlay3D. En lugar de tener que aproximarse hasta una entidad para poder consultar su inventario, ahora cada avatar dispone de un punto de mira con el que apuntar a otros objetos, como se puede ver en el centro de la Figura 6.21. Dependiendo del tipo de entidad se podrán realizar distintas acciones, como se describe en las próximas secciones.

Las críticas vertidas durante la evaluación de ViRPlay3D también nos han obligado a añadir más información a este entorno. En particular, se ha incluido información descriptiva sobre el escenario que está siendo simulado. Además, ViRPlay3D 2 mantiene un histórico de la simulación que puede ser consultado en cualquier momento por los alumnos. Este histórico se ha representado haciendo uso de los RPDs, descritos en la Sección 4.1.3.

Gracias a los cambios que se han producido durante la fase de implementación de este prototipo —como se verá en la Sección 6.7— se ha conseguido mejorar el aspecto de la interfaz, lo que lo hace mucho más amigable de cara a los alumnos. En la Figura 6.21 se puede ver el aspecto final de ViRPlay3D 2.

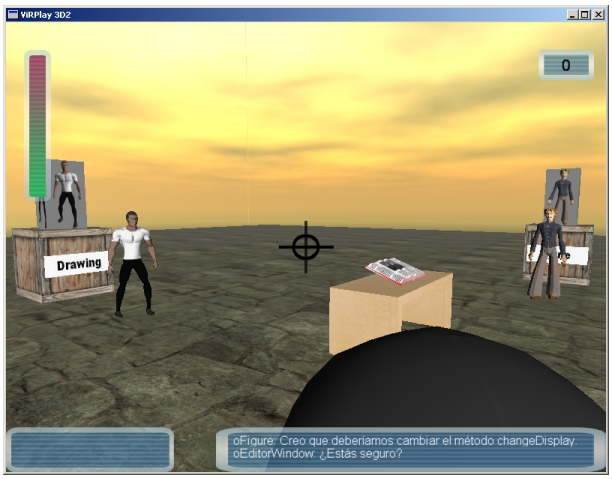


Figura 6.21: Aspecto gráfico de ViRPlay3D 2.

6.6.1. Decisiones de diseño

La principal decisión de diseño tomada está relacionada con el número de alumnos que pueden trabajar simultáneamente en el desarrollo de un escenario dentro del entorno virtual. A diferencia de ViRPlay3D en el que sólo existía un alumno, en ViRPlay3D 2 puede haber tantos alumnos como objetos sean necesarios para un determinado escenario. Esto ha influido enormemente en el resto de decisiones tomadas de acuerdo a los ejes de variabilidad de nuestra propuesta.

El primero de los ejes que vamos a tratar será el *Grado de libertad*. Se ha decidido que ViRPlay3D 2 sea un entorno libre en el que los alumnos controlen en todo momento el desarrollo de la simulación. En ningún caso el entorno es capaz de interpretar el rol de uno de los objetos interpretados por los alumnos. Ellos son los responsables de cada uno de los pasos de simulación relacionados con los objetos que controlan dentro del entorno y deciden cuándo se da por finalizada la ejecución de un escenario.

Sin entrar en conflicto con la decisión anteriormente tomada, se ha decidido que pueda, opcionalmente, haber un agente moderador, interpretado por un instructor humano o por otro alumno, que acompañe a los alumnos durante la ejecución de un escenario. La existencia del moderador no es obligatoria y será durante la asignación de roles cuando se decida si existe o no este agente. Éste se comportará de manera similar a como lo hacía el alumno en ViRPlay3D salvo que no podrá solicitar el siguiente paso de ejecución y dispondrá, al igual que el resto de los alumnos, de una herramienta de comunicación con la que discutir con ellos y ayudarles en la toma de las decisiones de diseño. También podrá interpretar el rol de cualquiera de los objetos que intervienen en una simulación para ayudar en caso de que los

alumnos se encuentren en una situación de bloqueo. Sin embargo, el agente moderador no podrá realizar ninguna modificación en el diseño de clases de un escenario.

En cuanto a la *Provisión de ayuda*, ya que ViRPlay3D 2 ha sido desarrollado para poner en práctica las sesiones de role-play para la enseñanza de patrones de diseño, se ha decidido incluir cierta información sobre los mismos. ViRPlay3D 2 dispone de un catálogo con todos los patrones de diseño orientados a objetos (Gamma et al., 1995) con información relacionada con las clases que en ellos participan y el comportamiento de los objetos durante la ejecución. Algunos escenarios están relacionados con la ejecución de un caso de estudio tras aplicar a su diseño de clases un patrón de diseño específico. Para ayudar en la ejecución de estos escenarios, ViRPlay3D 2 proporciona información sobre el patrón ligado a este escenario. Aquellos escenarios que no están asociados a la ejecución de un patrón estarán exentos de esta ayuda. El alumno es responsable de solicitar esta ayuda a través de una de las entidades del entorno.

La *Selección de escenarios* la realiza uno de los alumnos o el instructor. Una vez que un escenario ha sido seleccionado, la aplicación se queda a la espera de que el resto de alumnos se enrolen a participar en la simulación de ese escenario. Este comportamiento es similar al de los juegos en red, en el que un jugador selecciona el nivel que va a ser jugado y queda a la espera de que el resto de jugadores decidan participar en este nivel.

También relacionado con el aspecto de *Selección de escenarios* se ha decidido que sea un instructor el que tome el papel del agente generador de escenarios de ejecución. Éste será el responsable de crear la base de escenarios para ViRPlay3D 2 de la que los alumnos irán seleccionando el escenario que han de simular.

De cara al eje *Asignación de roles*, se ha decidido que, una vez que un escenario ha sido seleccionado, sea responsabilidad de cada alumno decidir qué rol se va a tomar —los alumnos serán los agentes asignadores de roles. Para cada escenario existe un número de roles que han de ser interpretados obligatoriamente por alumnos. Si no hay suficientes alumnos entonces el escenario no podrá ser simulado. Sin embargo, durante la asignación de roles se podrá decidir si se desea que exista o no la figura del agente moderador. En este caso, sí podrán dejarse sin cubrir algunos roles ya que el agente moderador tiene la capacidad de poder interpretar cualquier rol de igual modo que el alumno lo hacía en ViRPlay3D.

Por supuesto, en ViRPlay3D 2 los alumnos tienen acceso a la información contenida en los inventarios de las entidades del entorno virtual, que es la ayuda intrínseca al propio entorno. Además, la herramienta de comunicación también permite a alumnos —e instructor, si éste ha sido incluido en la simulación— comunicarse para compartir sus conocimientos y experiencia.

En lo que respecta al aspecto *Modificaciones del diseño inicial*, ViR-

Play3D 2 dispone de una herramienta de refactorización sencilla que permite modificar las clases inicialmente proporcionadas por el entorno. Como ya se verá en las siguientes secciones, mediante esta herramienta el alumno es capaz de añadir, modificar o eliminar responsabilidades y colaboraciones de las clases del escenario.

Los cambios en las clases los puede realizar cualquier alumno y sobre cualquiera de las clases existentes. El instructor será el único que no podrá modificar el diseño inicial. Sin embargo, los cambios han de producirse por consenso. Una vez que se ha producido algún cambio en una clase, se activará la herramienta de consenso, que permite a cada alumno aceptar o rechazar el cambio producido. Las particularidades de esta herramienta se describirán más adelante.

La modificación de cualquier clase supone reiniciar el escenario. De este modo se obliga a los alumnos a comprobar que los cambios producidos afectan exclusivamente a aquello que ha motivado la modificación de la clase y no a ninguna otra parte de la simulación.

De cara al aspecto *Evaluador*, se ha decidido que sea un instructor humano quien evalúe la realización de un escenario. Aunque el instructor puede ayudar en la realización del escenario, la evaluación se realizará tras completarlo, haciendo uso de la información de bitácora generada por el entorno. Con toda esta información se puede volver a simular el escenario en ViRPlay3D 2 y observar el resultado final de la simulación de los alumnos. Para fomentar aún más el nivel de participación de Presentación, esta evaluación puede ser realizada también por alumnos que discutan la calidad de la solución dada por sus compañeros.

6.6.2. Representación metafórica

Al igual que su predecesor, ViRPlay3D 2 usa la metáfora antropomórfica de las sesiones presenciales de role-play. Cada uno de los objetos de la simulación dispondrá de un avatar antropomórfico que lo represente dentro del mundo. A diferencia de ViRPlay3D, en este segundo entorno cada uno de los avatares es controlado por un alumno. Teniendo en cuenta los comentarios de los evaluadores de ViRPlay3D se ha decidido que cada clase de objetos tenga un avatar distinto. Tan solo los objetos de la misma clase comparten el modelo del avatar. Aunque los mapas de los escenarios son desarrollados de manera que cada avatar aparece inicialmente cerca de la clase a la que instancia, ahora cada uno de estos avatares se puede mover libremente por el mundo, por lo que si no existe ninguna diferencia entre los avatares será difícil que los alumnos sepan qué objeto es cada cual. En la Figura 6.22(a) se puede ver algunos de los modelos empleados para representar los objetos dentro del mundo.

Puede haber ocasiones en las que el rol de un objeto puede ser interesante

(a) Apariencia normal (b) Objeto no creado

Figura 6.22: Avatares para la representación de objetos en ViRPlay3D 2: (a) apariencia normal y (b) apariencia transparente cuando el objeto aún no ha sido creado.

para que sea interpretado por un alumno aunque este objeto sea creado en medio de la simulación. Aún así sería conveniente que el alumno dispusiera de un avatar con el que aparecer dentro del entorno virtual, poder consultar información de las entidades del mundo y poder participar de las decisiones que se tomen en grupo. Para esto se ha decidido que se siga utilizando el avatar del objeto pero éste tendrá una apariencia transparente, como aparece en la Figura 6.22(b). Sólo adoptará su presencia normal cuando sea creado. Por supuesto, no es posible pasar mensajes a este objeto mientras no haya sido creado. Esto también solventa una de las críticas realizadas sobre ViRPlay3D, en la que algunos evaluadores se quejaban de que de inicio existían objetos que aún no habían sido creados.

Ya que en ViRPlay3D 2 cada alumno representa a un objeto, se desea aumentar la sensación de que el alumno se meta en el papel del objeto. Para ello, la cámara en tercera persona que se empleaba en ViRPlay3D ha sido sustituida en ViRPlay3D 2 por una cámara en primera persona que consigue una mayor inmersión del alumno dentro del entorno. Ahora el alumno navega por el entorno metido dentro de un objeto, por lo que se consigue el efecto del alumno metido en la piel del objeto que representa.

En el diseño de ViRPlay3D 2 se ha descrito que durante la asignación de roles se puede decidir que exista o no un instructor. Aunque este avatar también tendrá un aspecto antropomórfico sería recomendable que fuese distinguible del resto de los avatares que representan objetos ya que, entre otras diferencias, este avatar no dispone de ningún tipo de inventario que otros

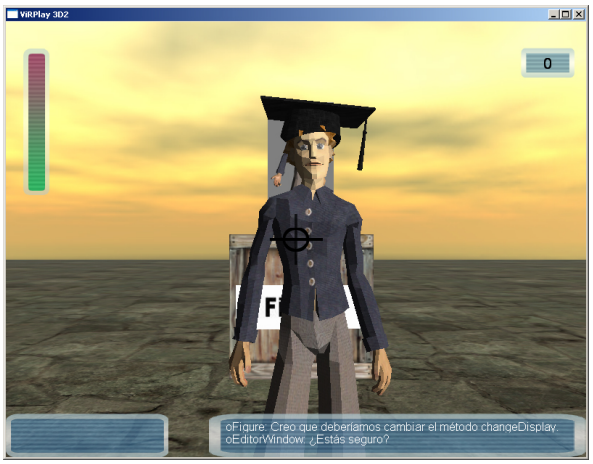


Figura 6.23: Avatar del instructor en ViRPlay3D 2, caracterizado por el birrete.

alumnos puedan consultar. Para que el avatar del instructor sea distinguible dentro del entorno virtual, se le ha añadido un birrete académico. El aspecto final del avatar aparece en la Figura 6.23.

Aquel que controla el avatar del instructor también verá el mundo desde una vista en primera persona desde dentro de su avatar. Sin embargo, para que el instructor pueda tener un mayor control de la simulación, se ha incluido la posibilidad de que éste pueda cambiar a una cámara libre que le permita controlar el mundo por completo. Esta cámara libre se mueve con el ratón y le permite desplazarse por todo el mundo para ver qué es lo que está haciendo cada uno de los avatares. El instructor podrá intercambiar libremente entre las dos cámaras a lo largo de la simulación.

Las clases mantienen una representación muy similar a la de ViRPlay3D pero intentando mejorar la metáfora de acuerdo a los comentarios realizados durante la evaluación de ViRPlay3D. En ViRPlay3D 2 cada clase es representada mediante una caja en el que aparece el nombre de la clase que representa, al igual que en ViRPlay3D. Pero ahora sobre la clase aparece un bloque en el que se puede ver el aspecto de las instancias de esa clase. De esta forma es más fácil distinguir la clase a la que pertenece cada uno de los objetos que aparecen por el mundo. En la Figura 6.24, se puede ver el aspecto de esta entidad. Como se puede ver, estas entidades siguen siendo perfectamente distinguibles con respecto a los objetos, lo que no lleva a confusión con los mismos.

El paso de mensajes no ha sufrido ningún cambio de representación con respecto a ViRPlay3D. Se mantiene la existencia de la pelota que se irá pasando entre objetos para representar el flujo de ejecución. Igualmente, el objeto activo será aquél que sostiene la pelota. También se ha mantenido la ayuda textual que aparece sobreimpresionada cada vez que se pasa un

Figura 6.24: Aspecto de una entidad de clase en ViRPlay3D 2. Junto a ella, un objeto instancia suya.

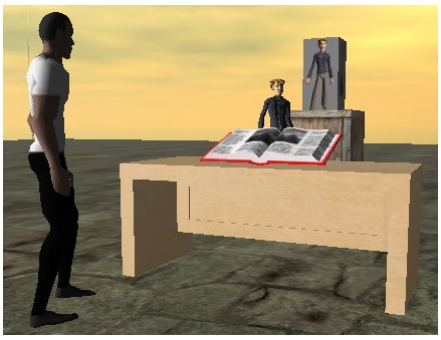


Figura 6.25: Entidad visual en ViRPlay3D 2 sobre la que los alumnos pueden consultar la información asociada al escenario que están simulando.

mensaje. Esta ayuda sigue siendo un diálogo entre los objetos y el tipo de mensajes que aparecen es el mismo que el que se describió para ViRPlay3D.

Como se verá en la siguiente sección, los escenarios de ViRPlay3D 2 van a incluir una entidad sobre la cual, los alumnos puedan consultar información sobre el escenario que están simulando. Por tanto, es necesario que esta entidad tenga una representación visual. Se ha decidido que esta entidad aparezca representada como un escritorio con un libro del que el alumno saca toda la información que se proporciona al iniciar el escenario. La representación visual de esta entidad aparece en la Figura 6.25.

Por último, se ha añadido una nueva metáfora visual de acuerdo a algunos de los comentarios producidos durante la evaluación del anterior prototipo. Se apuntó que durante la simulación no existía nada que hiciera saber que un objeto había cambiado su estado por lo que había que estar consultando constantemente el estado de un objeto para saber cuándo se producía algún cambio. Para paliar este problema se ha decidido incluir un efecto visual

Figura 6.26: Efecto visual alrededor del avatar y mensaje que aparecen cuando un objeto cambia de estado.

que indica cuándo un objeto cambia de estado (Figura 6.26). Además, en pantalla aparece sobreimpresionado un mensaje que indica qué objeto ha sufrido el cambio de estado. Este efecto, junto con el mensaje, aparece en el instante en el que ha cambiado el estado del objeto y permanece durante unos segundos para indicar dicho cambio.

6.6.3. Representación de la información del role-play

ViRPlay3D 2 mantiene la metáfora de los inventarios de las entidades como medio de almacenamiento y consulta de toda la información existente en el escenario. Como se verá a continuación, ViRPlay3D 2 es aún más rico en información que su predecesor ViRPlay3D. La evaluación de ViRPlay3D arrojó que es necesario disponer de información sobre el escenario que está siendo ejecutado y del histórico de la simulación hasta el momento actual, por lo que este tipo de información ha sido incluida en ViRPlay3D 2. A lo largo de esta sección de describirá el aspecto y cuál es la información contenida en los inventarios de acuerdo al tipo de entidad que esté siendo consultada.

La principal diferencia de ViRPlay3D 2 con su predecesor en cuanto a la información del escenario se refiere es que ViRPlay3D 2 abandona la documentación y el código fuente de las clases para pasar a emplear las tarjetas CRC como información asociada a las clases del escenario. La evaluación de ViRPlay3D nos ha hecho ver que la información contenida en el anterior prototipo no resultaba especialmente útil. Además, como se ha podido extraer de la evaluación de las experiencias presenciales descritas en el Capítulo 4, las tarjetas CRC contienen suficiente información para los alumnos a nivel de diseño. Así, la consulta del inventario de una clase en ViRPlay3D 2 mostrará

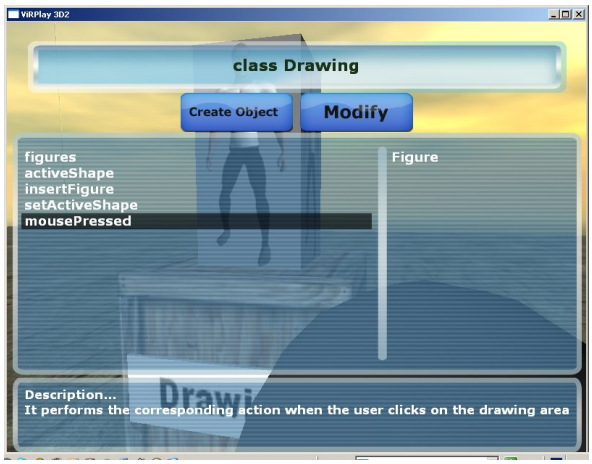


Figura 6.27: Inventario de una clase en ViRPlay3D 2.

la tarjeta CRC asociada a dicha clase. Como se puede ver en la Figura 6.27, el aspecto de este inventario es muy similar al de una tarjeta CRC, lo que simplifica enormemente la comprensión de la información.

Este inventario se ha mejorado con respecto a ViRPlay3D gracias a que el acceso a la información es mucho más ordenado y la información contenida está más simplificada. Como se puede ver en la Figura 6.27, el inventario de una clase queda dividido en cuatro partes:

- En la parte superior aparece el nombre de la clase sobre la que se está realizando la consulta de información y propietaria del inventario.
- A la izquierda aparecen las responsabilidades asociadas a la clase consultada.
- A la derecha aparece una lista con los colaboradores de la clase.
- En la parte inferior aparece el área donde se mostrará la información asociada a los distintos elementos de la tarjeta CRC. Cuando se despliega el inventario, aquí aparece una descripción de la clase y de su cometido dentro del diseño inicial. Cada vez que el alumno seleccione una responsabilidad, en este espacio aparecerá la información asociada a la misma, así como los parámetros que necesita.

Desde este mismo inventario, cualquier alumno podrá realizar modificaciones sobre el diseño inicial de la clase y crear nuevos objetos, como veremos más adelante.

El inventario de un objeto en ViRPlay3D 2 es muy similar al de ViRPlay3D en cuanto a la información contenida, si bien su aspecto es bastante distinto, como se puede ver en la Figura 6.28. Desde este inventario el alumno puede consultar tanto la información estática como dinámica del objeto.

Figura 6.28: Inventario de un objeto en ViRPlay3D 2. A la derecha aparece la interfaz de modificación del estado del objeto.

La información dinámica aparece nada más iniciar el inventario de un objeto. Aquí se muestra una lista con los atributos que definen el estado de un objeto, así como el valor asociado a cada uno de ellos. Cada vez que el alumno selecciona uno de estos atributos, en la parte inferior aparece la descripción asociada al atributo seleccionado. A través de este inventario un alumno puede modificar el estado de un objeto, como se describe en la siguiente sección. Desde esta interfaz también se puede acceder al inventario de la clase de la que es instancia, presentando la tarjeta CRC asociada tal y como ya se ha visto en la Figura 6.27.

Al igual que el anterior, el inventario de la pelota contiene información muy similar al inventario de esta misma entidad en ViRPlay3D. El inventario tiene el aspecto que aparece en la Figura 6.29 y contiene la información sobre el último mensaje que se ha pasado o del que actualmente se está pasando. Desde aquí se tiene acceso a toda la información del mensaje: quién ha sido el objeto invocador y sobre qué objeto ha sido invocado el mensaje, cuál es el mensaje que se ha pasado y con qué parámetros reales ha sido pasado. También existe un marcador que indica si el mensaje es una invocación o si es de retorno y, en este caso, también aparecerá información sobre el valor retornado. Al igual que en los anteriores inventarios, si el alumno selecciona el mensaje o alguno de los parámetros del mismo, en la parte inferior aparece información descriptiva del elemento seleccionado.

Los escenarios simulados en ViRPlay3D 2 son de una mayor complejidad que los de su predecesor ya que serán empleados para enseñar patrones de diseño. Además, el escenario no es guiado sino que los alumnos han de completarlo seleccionando libremente los pasos de ejecución que se han de realizar. Esto obliga a que ViRPlay3D 2 tenga que disponer de información

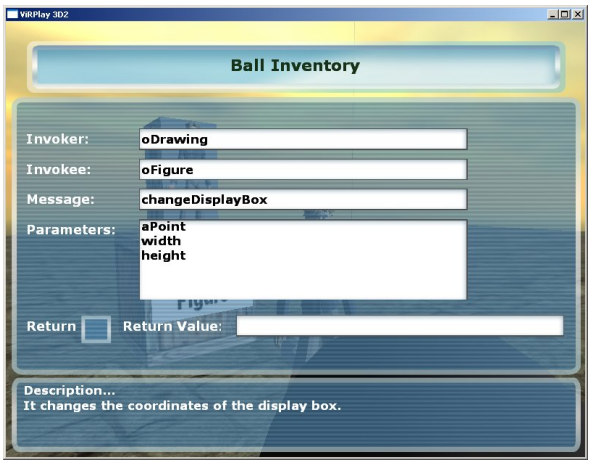


Figura 6.29: Inventario de la pelota en ViRPlay3D 2.

relacionada con el escenario que actualmente está siendo simulado. Al iniciar un escenario aparece un inventario similar al de la Figura 6.30, donde el alumno puede leer una descripción detallada sobre el escenario que va a ser simulado. Además, también dispone de información del diseño inicial de clases que se proporcionan.

El inventario de la descripción del escenario contiene acceso a información adicional, dependiente del escenario que está siendo simulado. Si el escenario está relacionado con algún patrón de diseño, aquí también se puede consultar información sobre el mismo. En particular, se presenta un diagrama de clases con los roles genéricos del patrón de diseño y un diagrama de secuencia que describe el modelo de ejecución que se suele emplear para ese patrón. Toda esta información puede ser extraída de (Gamma et al., 1995).

Una vez que los alumnos han iniciado la simulación de un escenario esta información sigue estando disponible dentro del entorno. Para ello, en el escenario se ha colocado una entidad representada por un escritorio (Figura 6.25) cuyo inventario es el mismo que se ha presentado al inicio del escenario (Figura 6.30). De esta forma, si algún alumno desea revisar la información relacionada con el escenario tan solo ha de acercarse a esta entidad y consultar su inventario. Este contenido permanece inmutable a pesar de los cambios que los alumnos puedan realizar en el diseño de clases.

Por último, en ViRPlay3D 2 se ha incluido información sobre el estado actual de la simulación. En la Figura 6.21 se puede observar que en la parte superior derecha hay un marcador que irá cambiando a medida que los alumnos hagan y deshagan pasos de mensajes. Este marcador exhibe el número de pasos de mensajes realizados hasta el momento. Todos los alumnos pueden consultar en cualquier momento este marcador y, de esta forma, poder ver el diagrama de role-play (RPD) que representa el estado actual de

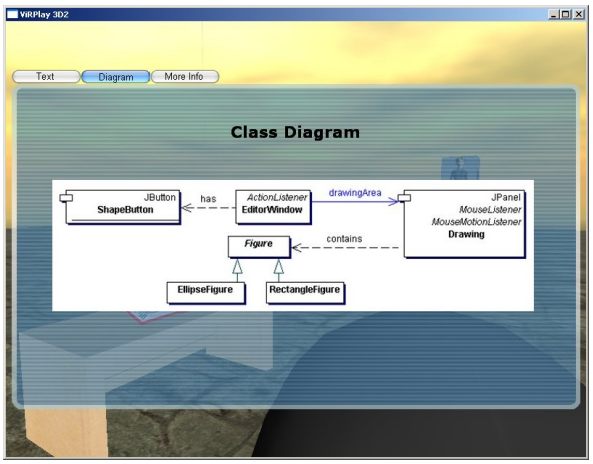


Figura 6.30: Inventario del escritorio: contiene información descriptiva del escenario simulado. Aquí se presenta el diseño de clases inicial.

la simulación. Este RPD se va generando dinámicamente a medida que los alumnos van ejecutando pasos de mensaje gracias a que se está empleando la misma herramienta de autoría de información dinámica desarrollada para ViRPlay3D 2 y que se describirá en la Sección 6.8. El aspecto de estos RPDs dentro del inventario del marcador se puede ver en la Figura 6.31.

6.6.4. Interacción

ViRPlay3D 2 comparte muchas de las mecánicas vistas en ViRPlay3D. Sin embargo, también añade otras distintas relacionadas con las decisiones de diseño específicas que se han tomado para este segundo entorno. También hay que tener en cuenta que ViRPlay3D 2 admite dos roles distintos de usuario: el alumno y el instructor. En la descripción de las mecánicas que comienza a continuación, por defecto, todas las mecánicas serán interpretadas por ambos roles. En caso de que alguna de las mecánicas sea específica de uno de los roles, se detallará para qué rol está disponible.

Como en ViRPlay3D, los alumnos y el instructor disponen de un avatar con el que se pueden mover libremente por el mundo para consultar información del resto de entidades. Para ello se utiliza la acción *Mirar a*. En la evaluación de ViRPlay3D se ha criticado la necesidad de acercarse a una entidad para consultar su inventario. Por esto, esta acción se puede realizar a distancia en ViRPlay3D 2. Cuando un avatar sitúa su punto de mira sobre una entidad que dispone de un inventario, la acción *Mirar a* aparecerá en la lista de acciones disponibles para este avatar. La lista de acciones aparece en la parte inferior como se ve en la Figura 6.21. Al ejecutar esta acción se desplegará uno de los inventarios descritos en la sección anterior. Al igual que en ViRPlay3D, puede ocurrir que un avatar pueda mirar a varias

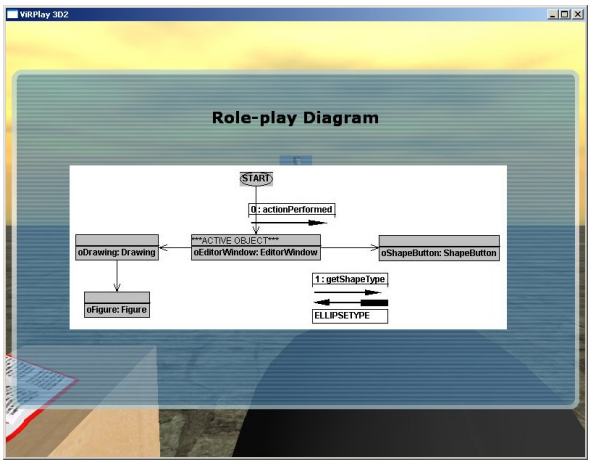


Figura 6.31: Inventario del marcador: contiene el RPD que representa el estado actual de la simulación.

entidades simultáneamente. En este caso, el usuario selecciona la acción a realizar utilizando la numeración que acompaña a cada una de las acciones.

Existen dos inventarios que no pueden ser consultados usando esta mecánica ya que no existe una entidad a la que mirar. El primero es el del marcador que aparece sobreimpresionado en la interfaz de usuario de ViRPlay3D 2. El segundo es el inventario del propio objeto manejado por un alumno. Un alumno ha de poder consultar su propio inventario para conocer o modificar el estado del objeto que representa. Para estos casos se han definido dos acciones que despliegan sendos inventarios. El instructor no dispone de la acción para consultar su propio inventario ya que no está interpretando el rol de ningún objeto. Sin embargo, sí ha de disponer de la acción para consultar el inventario del marcador y así poder ver el estado en el que se encuentra la simulación.

Ya que los alumnos van a tener el control de la ejecución de la simulación hay que proporcionarles mecánicas para que puedan modificar el estado de los objetos y realizar el paso de mensajes. En cuanto a la primera mecánica, cada alumno puede modificar el estado de su propio objeto a través del inventario del objeto. Como se puede ver en la Figura 6.28, al seleccionar un atributo de un objeto aparece el valor de éste en el lado derecho del inventario. Desde aquí se puede modificar el valor de este atributo. Cuando se produce este cambio el avatar del objeto presentará por un tiempo el efecto visual que aparece en la Figura 6.26. Un alumno sólo puede cambiar el estado del objeto que está interpretando. Sin embargo, el instructor puede cambiar el estado de cualquiera de los objetos a través del inventario de los mismos.

La mecánica del paso de mensajes es una de las que más modificaciones ha sufrido con respecto a ViRPlay3D. Ya que cada uno de los alumnos controla

a un objeto, éstos serán responsables de cada uno de los pasos de mensaje del escenario. Además, se ha añadido una componente lúdica al paso de mensajes: el alumno responsable del objeto activo decide hacia dónde y con qué fuerza lanza la pelota. De esta forma, el paso de mensajes se divide en tres fases:

1. Creación del mensaje. Antes de lanzar la pelota, el objeto activo ha de crear el mensaje que desea pasar. Esto se realiza a través del inventario del objeto que va a ser destinatario del mensaje que, como ya se ha explicado, aparece cuando se apunta sobre el objeto y se activa la acción *Mirar a*. Una vez en este inventario se selecciona una de las responsabilidades del objeto y se activa la herramienta de creación de mensajes, apareciendo la interfaz que se muestra en la Figura 6.32. Aquí, el alumno puede seleccionar el tipo de mensaje —invocación o retorno—, el valor real de los parámetros en esta invocación y el valor de retorno, en caso de que el tipo de mensaje pasado sea de retorno.

2. Lanzamiento de la pelota. Como se puede ver en la parte izquierda de la Figura 6.21, la interfaz del alumno dispone de una barra de potencia que le servirá para lanzar la pelota en caso de que el objeto que está manejando sea el activo y, por tanto, esté en posesión de la pelota. La pelota sale lanzada en la dirección en la que mira el avatar y con la fuerza marcada por el lanzador. No es necesario haber creado un mensaje para lanzar la pelota aunque, en este caso, no se estará pasando ningún mensaje.

3. Recogida de la pelota. Una vez creado un mensaje y lanzada la pelota el paso de mensaje no concluye hasta que el receptor del mensaje no recoge la pelota. Ésta puede ser recogida cuando el avatar del objeto receptor apunta a la pelota y está lo suficientemente cerca de ella. En caso de que el mensaje no haya sido creado entonces el único que puede recoger la pelota es el objeto activo.

Durante la creación de mensaje existe una restricción adicional en cuanto al objeto que puede ser seleccionado como receptor del mensaje. En ViRPlay3D 2 se usan las tarjetas CRC como medio de representación de las clases. Dentro de éstas, aparecen los colaboradores, que son las únicas clases con las que se sabe comunicar una determinada clase. Por tanto, si el alumno apunta sobre un objeto que es instancia de una clase que no es colaboradora de su objeto, entonces no será posible activar la herramienta de creación de mensajes, por lo que no podrá crear ningún mensaje. Esto es coherente con el uso de las tarjetas CRC ya que, si no es colaboradora, entonces no sabe cómo comunicarse con ella porque desconoce cuáles son las responsabilidades de las que se encarga.

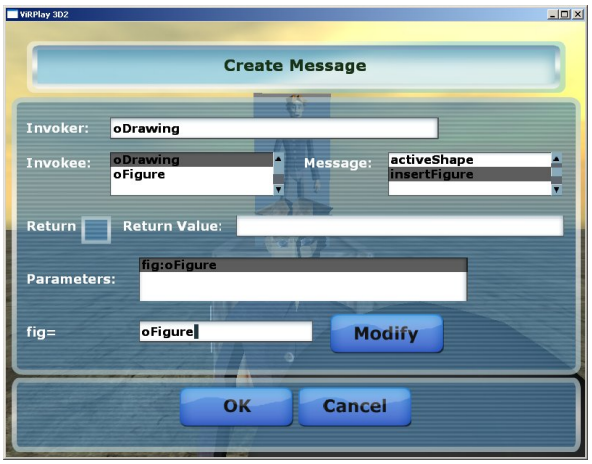


Figura 6.32: Interfaz para la creación de mensajes (accesible desde el inventario del objeto que va a ser destinatario).

Existe un caso especial de creación de mensajes, utilizado para la construcción de nuevos objetos. Esto se realiza a través del inventario de una clase. Cuando un avatar despliega el inventario de una clase que es colaboradora del objeto que representa verá que en dicho inventario aparece activo el acceso a la herramienta de creación de objetos. Al activarla aparece una interfaz muy similar a la de la herramienta de creación de mensajes en la que el alumno puede seleccionar el constructor que va a emplear, los parámetros reales con los que va a invocar este constructor y el nombre que va a tener la instancia que se desea crear. Una vez realizado esto, se deberá lanzar la pelota de modo que ésta toque la caja. En este momento, junto a la caja aparecerá el nuevo objeto creado. En caso de que este objeto sea uno de los avatares ya existentes en el mundo y que estaban con apariencia transparente, este avatar será trasladado junto a la caja y su apariencia pasará a ser normal. La clase devolverá la pelota rodando automáticamente una vez que ha construido el objeto, siendo el objeto que invocó la creación el responsable de recogerla.

El instructor también puede decidir cuál va a ser el siguiente paso de ejecución que se va a dar. Generalmente esto se hará para desbloquear a los alumnos si observa que tienen problemas para continuar con la simulación, o para representar el papel de alguno de los objetos que no son manipulados por ningún alumno. Al igual que antes, el instructor dirigirá su punto de mira hacia el objeto que va a ser destinatario del mensaje y, a través de la interfaz de este objeto, podrá acceder a la interfaz de creación de mensajes. Una vez creado el mensaje, el instructor podrá solicitar la ejecución del siguiente paso de mensaje. En este caso, el paso de mensaje se automatizará de igual forma que se hacía en ViRPlay3D. Los avatares de los objetos origen y destino del

mensaje quedarán bloqueados en una determinada posición, se encararán, se producirá el diálogo y la pelota será lanzada entre el invocador e invocado, recogiendo éste último la pelota automáticamente. Esta automatización se realiza para que no haya opción a la confusión de los alumnos sobre el mensaje que se ha pasado.

En el entorno puede haber también objetos controlados por agentes. Estos objetos suelen ser demasiado sencillos o de ninguna importancia pedagógica en el escenario. El rol de estos objetos no es asignable durante la fase de asignación de roles del escenario. Estos agentes saben cómo responder de cara a determinados pasos de mensaje y los cambios de estado que han de producir en sí mismos o en otros objetos del escenario. Cuando uno de estos objetos es destinatario de un mensaje que ha sido pasado por alguno de los alumnos, el avatar se dirigirá a recoger la pelota y, una vez hecho esto, realizará el paso de mensaje que tiene asociado en respuesta al mensaje que ha recibido.

Al igual que en ViRPlay3D, los alumnos pueden volver atrás en un paso de mensaje o reiniciar la simulación. Sin embargo, a diferencia del anterior prototipo, esta decisión no se puede tomar de manera unilateral sino que ha de ser consensuada por todos los alumnos. Esto se debe a que ahora el alumno no está solo en el entorno virtual sino que ha de trabajar colaborativamente con otros alumnos. Como se verá más adelante, cuando un alumno decide solicitar la vuelta atrás en la simulación o reiniciar la misma, la herramienta de consenso aparecerá en la interfaz de todos los alumnos, pasándose a la votación de la acción. El funcionamiento completo de esta herramienta se describe más adelante en esta misma sección.

En ViRPlay3D 2 también es necesario decidir cuándo se da por concluido un escenario ya que debido a la libertad que se ha dado a los alumnos no se puede controlar cuándo se ha acabado un escenario. Por tanto, cada alumno tiene a su disposición la acción de *Concluir el escenario*. Esta acción también ha de ejecutarse de manera consensuada. Una vez concluido el escenario, se guarda toda la bitácora del mismo para que posteriormente sea un instructor humano el responsable de evaluar la simulación realizada.

ViRPlay3D 2 permite la realización de modificaciones sobre el diseño inicial de clases. Como se ha explicado, estas modificaciones consisten en refactorizaciones muy sencillas como añadir, modificar o eliminar responsabilidades y colaboradores de una clase. Esto se realiza a través del inventario de la clase que se desea modificar, como se puede ver en la Figura 6.33. La interfaz es muy simple: tan solo hay que seleccionar la responsabilidad o colaboración que se desea modificar o eliminar y pulsar en el botón correspondiente. En caso de que la modificación sea la de añadir, se pulsará sobre este botón y se rellenarán los campos necesarios. Cualquier modificación ha de ir acompañada de una descripción que el responsable del cambio ha de rellenar. Esta descripción será de utilidad para el instructor durante

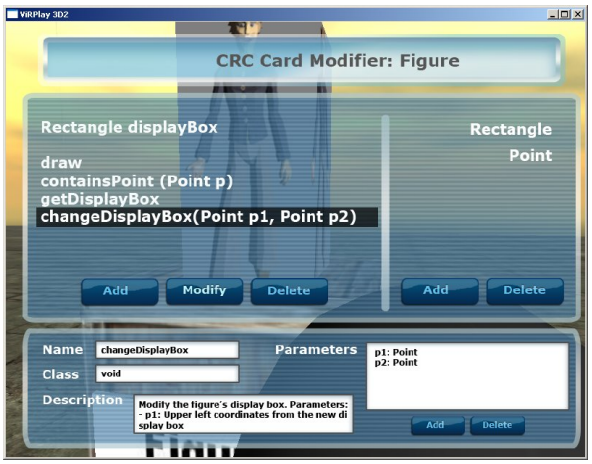


Figura 6.33: Interfaz para la modificación de una clase, accesible desde el inventario de clases.

la evaluación del escenario.

Generalmente, el objeto instancia de esta clase debería ser el responsable de hacer estos cambios. Pero ya que se pueden simular escenarios sin que estén todos los roles asignados y el instructor no va a poder realizar estas modificaciones, se permite que cualquier objeto pueda modificar una clase. De todas formas, al igual que las anteriores acciones, la modificación de una clase requiere del consenso de los alumnos.

Para facilitar la colaboración entre los alumnos y para que el instructor pueda intervenir durante la simulación ViRPlay3D 2 ofrece una herramienta de comunicación. Esta herramienta es un tipo chat y se puede ver en la parte inferior derecha de la Figura 6.21. Cada mensaje aparece identificado por el nombre del objeto que lo ha enviado salvo el instructor, que aparece con las iniciales *instr*. En cualquier momento, un alumno puede desplegar la herramienta de comunicaciones y enviar un mensaje, como aparece en la Figura 6.34. Este mensaje llegará a todos los alumnos y al instructor. Además, todas las conversaciones son almacenadas en la bitácora para que el instructor pueda ver los posibles conflictos que se producen durante la simulación cuando pase a evaluar el escenario.

Algunas de las acciones realizadas por los alumnos necesitan de un consenso entre ellos para que sean llevadas a cabo. Cuando una de estas acciones es realizada, en la interfaz de usuario de cada uno de los alumnos aparecerá la herramienta de consenso, que se puede ver en la Figura 6.35. Aquí aparece quién ha realizado la acción que ha activado la herramienta y qué acción ha realizado —deshacer un paso de mensajes, reiniciar el escenario, modificar una clase, o solicitar la finalización del escenario— y una descripción de la misma. Esta herramienta estará activa solamente durante un determinado tiempo en el que los alumnos podrán aceptar o rechazar la realización de

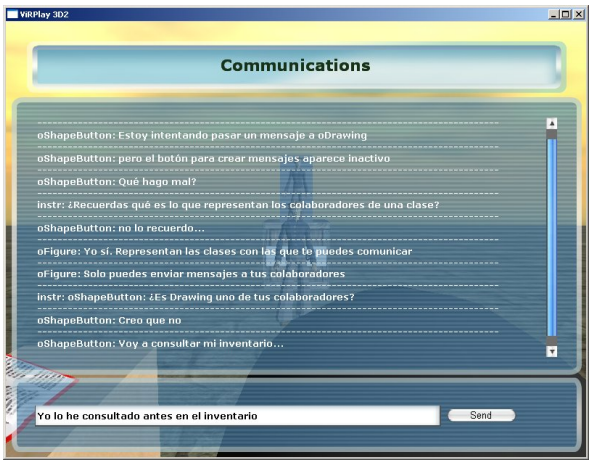


Figura 6.34: Herramienta de comunicación.

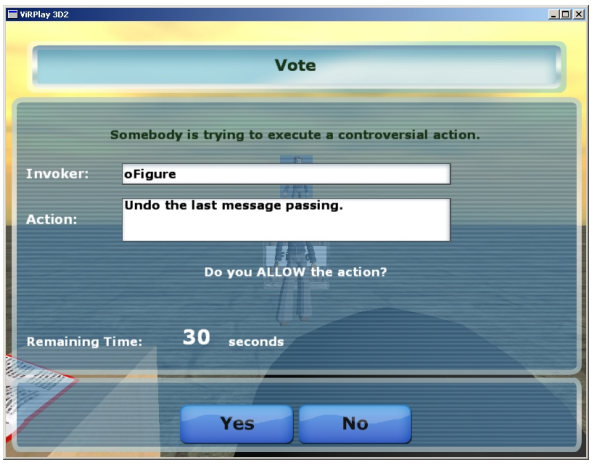


Figura 6.35: Herramienta de consenso.

la acción. Una vez que se ha agotado el tiempo, se recuentan los votos de los alumnos y sólo si se alcanza una mayoría de la aceptación se realizará la acción inmediatamente. Puede ocurrir que por distracción de los alumnos, errores en la red, etc. algunos de los alumnos no voten, por lo que no se alcanza el consenso aunque pudiera ser que la mayoría de los alumnos estuvieran de acuerdo con la realización de una acción. Para evitar estas situaciones que pueden producir el bloqueo de la simulación de un escenario, se permite que el alumno que ha realizado la acción pueda realizarla finalmente de manera unilateral. Sin embargo, este hecho anormal quedará guardado en la bitácora para que el instructor lo tenga en cuenta a la hora de evaluar el escenario.

Finalmente, ViRPlay3D 2 va guardando una bitácora con todo lo que está ocurriendo en el escenario. En particular, se almacenan todas las comunica-

ciones, los pasos de mensajes realizados por los alumnos, las modificaciones de las clases y todas las acciones que necesitan de consenso y si éste se ha alcanzado o no. Toda esta información será usada para que el instructor humano evalúe la simulación que han realizado los alumnos. Con la información contenida en esta bitácora, se puede volver a simular el escenario de igual forma que se hacía con ViRPlay3D. Así mismo, esta información se puede usar con la herramienta de creación de RPDs para, mediante este diagrama, visualizar rápidamente cómo ha ido la sesión de role-play.

6.7. Instanciación de la arquitectura genérica para ViRPlay3D 2

Siguiendo el diseño descrito se ha implementado un entorno que instancia la arquitectura genérica para entornos virtuales de role-play. Para el desarrollo del prototipo se ha desechado gran parte de ViRPlay3D y de los módulos de JV^{2}M utilizados. Esta decisión ha sido tomada por dos motivos fundamentales:

- La implementación del anterior prototipo no estaba preparada para la arquitectura en red ni para dar soporte a múltiples usuarios.
- Se ha querido diseñar e implementar un subsistema Núcleo de la arquitectura genérica de los entornos virtuales de role-play que sea fijo para futuras instanciaciones de dicha arquitectura.

Para el desarrollo de este nuevo entorno se ha utilizado una implementación de la arquitectura de videojuegos de (McShaffry, 2005), descrita en el Capítulo 5, y que hemos empleado a lo largo de los últimos años como ejemplo de prototipo en el Master en Desarrollo de Videojuegos de la Universidad Complutense de Madrid, así como para los Cursos de Programación de Videojuegos impartidos en la Escuela Complutense de Verano. Este prototipo, conocido como *El Laberinto*[3], está implementado usando el motor de juegos Nebula2, una evolución del motor utilizado en el desarrollo de JV^{2}M y de ViRPlay3D. Además, la arquitectura de este prototipo permite tanto la creación de juegos para un único usuario como para el desarrollo de juegos en red basado en arquitectura de cliente-servidor. La utilización de Nebula2 también nos ha permitido mejorar el aspecto gráfico del entorno, sobre todo en cuanto a avatares se refiere, ya que existe un plug-in comercial para Maya que exporta al formato de ficheros propio de este motor, y un plug-in de generación automática de personajes animados, también para Maya, desarrollado por alumnos de la Facultad de Informática de la Universidad

[3] El prototipo desarrollado es una versión en 3D del juego "El Laberinto del Sultán" para Amstrad, que se puede ver en acción en `http://www.youtube.com/watch?v=3x4b4k1pPRE` (último acceso: 15-02-2008)

Complutense de Madrid como proyecto de fin de carrera (Meco-Alías et al., 2006).

La implementación de este nuevo prototipo se ha desarrollado en dos fases:

- Durante la primera fase se ha realizado una reimplementación de ViRPlay3D siguiendo la nueva arquitectura de videojuegos descrita. Esto ha servido para iniciar el desarrollo del subsistema Núcleo. Además, en esta versión ya se ha incluido el nuevo tipo de información contenida en los escenarios descritos para ViRPlay3D 2 —tarjetas CRC e información descriptiva del escenario.

- En la segunda fase se ha dado soporte multiusuario, haciendo las modificaciones del Núcleo necesarias para permitir una arquitectura de red cliente-servidor.

La versión desarrollada durante la primera fase es de utilidad para la evaluación de los ejercicios completados por los alumnos. La información de bitácora relacionada con el diseño de clases y la ejecución del escenario generada durante la simulación de un escenario es empleada por esta primera versión de ViRPlay3D 2 como escenario de ejecución. En él, el instructor puede observar de manera controlada el diseño final de clases de los alumnos, así como cuáles han sido los pasos de mensajes realizados durante la simulación.

Para poder describir la instanciación de la arquitectura realizada para ViRPlay3D 2 necesitamos primeramente comprender la arquitectura de *El Laberinto*. Esta arquitectura se divide en tres módulos fundamentales:

- **Lógica**. Contiene todo el control de las reglas del juego. Es un mundo basado en casillas en dos dimensiones donde está el laberinto y las entidades del mundo. Estas entidades se dividen en entidades estáticas, como las joyas o las letras que hay dentro del laberinto, y en avatares, que son las entidades que tienen movimiento, como el avatar del jugador y los fantasmas que hay dentro del laberinto. La lógica también dispone de un sencillo motor de colisiones que indica cuándo entran en contacto dos entidades del mundo. La comunicación con la lógica es a través de comandos y como único punto de entrada está la clase `Partida`, que hace de fachada del resto de la lógica. Muchas de las clases del módulo de lógica proporcionan interfaces de observador, de modo que los principales eventos que ocurren dentro de la partida son propagados a través de estos observadores a aquellos módulos que se encuentren a la escucha.

- **Aplicación**. Este módulo es el que gestiona el bucle principal de la aplicación, así como la máquina de estados jerárquica responsable de

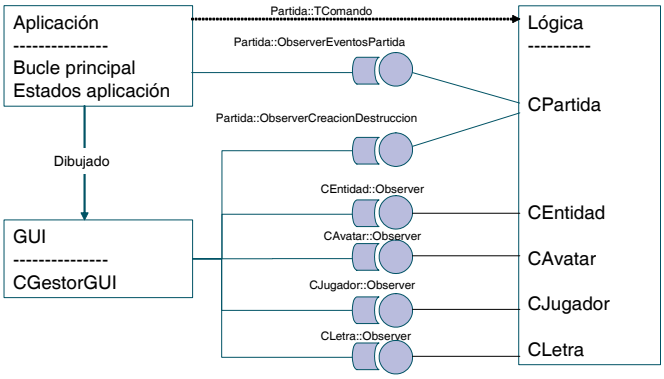


Figura 6.36: Vista de alto nivel de la arquitectura de *El Laberinto*, base para el desarrollo de ViRPlay3D 2.

controlar los estados por los que pasa la aplicación. Cualquiera de los estados sabe cómo comunicarse con la lógica mediante comandos. Algunos de estos estados permanecen a la escucha de los eventos que ocurren durante la partida, ya sea para cambiar el estado del juego, ya sea para realizar cambios en la interfaz del usuario a través de la interfaz de *scripting*.

- **GUI** (del inglés, *Graphical User Interface*). Este módulo es responsable tanto de la gestión del dibujado de la escena en 3D como del interfaz de usuario en 2D usando Nebula 2. Para el primero, existe un gestor de GUI encargado de crear, almacenar, dibujar y eliminar todas las entidades visuales de la escena. Existirá una entidad gráfica por cada una de las entidades lógicas y aquéllas permanecerán a la escucha de los eventos provocados por las entidades lógicas para actualizar convenientemente la escena en 3D. Para la gestión de la interfaz de usuario en 2D, Nebula 2 proporciona una interfaz de scripting en Tcl. Esta interfaz es bidireccional, de modo que el módulo de Aplicación puede modificar los menús e interfaces de usuario y ésta puede comunicar al módulo de Aplicación las acciones realizadas por el usuario.

En la Figura 6.36 se puede ver un esquema de la arquitectura de *El Laberinto*, así como las principales clases contenidas en cada uno de los módulos. A partir de esta arquitectura se ha desarrollado parte de la nueva instanciación de la arquitectura general de entornos virtuales de role-play que aparece en la Figura 6.37.

El *Gestor del mundo* es una versión modificada del módulo lógico de *El Laberinto*. Se han mantenido las clases básicas de las entidades y se han desarrollado las clases propias del entorno virtual de role-play, como se puede ver en la Figura 6.38. Además, se ha añadido un sistema de percepción de modo que cada avatar pueda ver qué es lo que tiene delante de él (o a lo

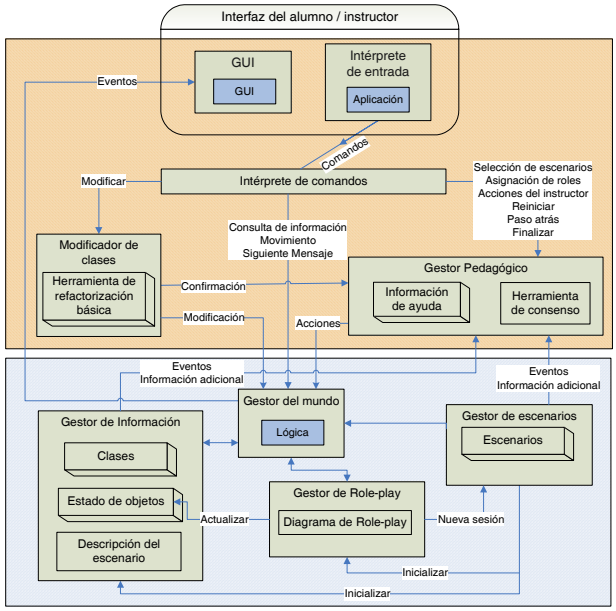


Figura 6.37: Instanciación de la arquitectura genérica para entornos virtuales de role-play, usada en el desarrollo de ViRPlay3D 2. En azul aparecen los módulos pertenecientes a la arquitectura de *El Laberinto*.

que está apuntando) y cuáles son las acciones que puede realizar sobre esa entidad. La inicialización de las entidades existentes en el mundo se realizarán a través del *Gestor de escenarios*, a partir de la información almacenada en el mapa del escenario cargado.

Al igual que en ViRPlay3D, el *Gestor del mundo* también es responsable de los inventarios de cada una de las entidades. Cada inventario es responsable de comunicarse con el *Gestor de información* para solicitar la información que ha de contener y está suscrito a los eventos de modificación de la información que pueda propagar el *Gestor de información*. Existe un inventario especial, el del marcador, que, en lugar de comunicarse con el *Gestor de información* ha de hacerlo con el *Gestor de role-play*, ya que éste es el que contiene la información relacionada con el RPD. Por otro lado, el *Gestor del mundo* implementa la herramienta de comunicaciones entre los alumnos.

El *Gestor del mundo* es el responsable de propagar los eventos con los que las interfaces de los alumnos y del instructor actualizarán su estado de acuerdo a los movimientos y las acciones de las entidades, presentación de la información de los inventarios (en la Figura 6.39), presentación de los mensajes sobreimpresionados en el entorno, activación y desactivación de la herramienta de consenso y los mensajes de la herramienta de comunicaciones.

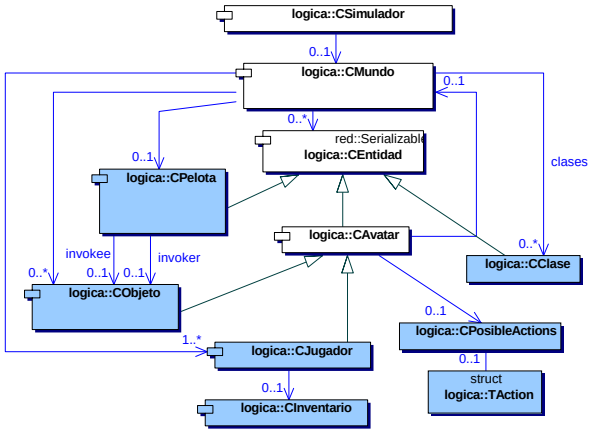


Figura 6.38: Entidades añadidas (en azul) al módulo de lógica de *El Laberinto* para crear el *Gestor del mundo* de ViRPlay3D 2.

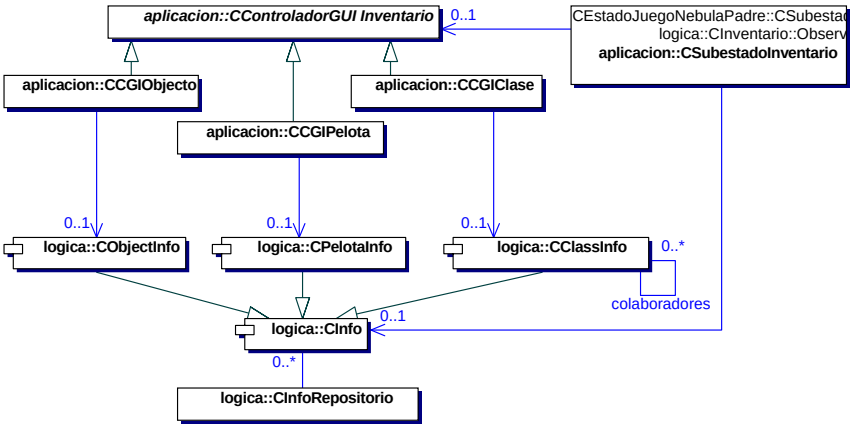


Figura 6.39: Gestión de la presentación de la información de los inventarios en ViRPlay3D 2. Por cada tipo de información a visualizar (módulo Lógica) existe un controlador gráfico responsable de presentarla por pantalla (módulo Aplicación).

El *Gestor de role-play* ha sufrido pocas variaciones con respecto al implementado en ViRPlay3D, ya que se ha mantenido dentro de lo posible el formato de la secuencia de pasos de mensajes del escenario. Las principales modificaciones se han producido de cara a adecuar la comunicación de este módulo con los que se han desarrollado nuevos para ViRPlay3D 2. Al igual que en el anterior prototipo, este módulo soporta la ejecución automática paso a paso de un escenario, deshacer un paso de mensaje y reinicio del mismo. También se encarga de gestionar la ejecución paso a paso de un escenario mediante la creación de mensajes por parte del instructor y de los alumnos y de guardar la bitácora de ejecución de este tipo de simulaciones. Además, dispone de un módulo que comunica a la herramienta de generación de RPDs la secuencia de mensajes que se va produciendo, de modo que cuando algún alumno solicite el inventario del marcador se pueda presentar el RPD del estado actual de la simulación.

Una de las principales diferencias del *Gestor de role-play* de ViRPlay3D 2 con respecto al anterior prototipo está en la gestión de sesiones. Cabe recordar que en ViRPlay3D 2 un escenario se puede componer de varias sesiones, por lo que el *Gestor de role-play* ha de poder comunicarse con el *Gestor de escenarios* para la inicialización de una nueva sesión dentro del escenario que actualmente está siendo desarrollado.

Al igual que en ViRPlay3D, el *Gestor de role-play* es responsable de comunicarse con el *Gestor del mundo* para coordinar las acciones relacionadas con el paso de mensajes, tanto cuando la ejecución se está realizando de manera automática como de manera manual por parte de los alumnos. También se encarga de gestionar el cambio de estados de los objetos y de informar de los mismos al *Gestor de información* y al *Gestor del mundo*, para mostrar el cambio de estado tal y como se describió en la Sección 6.6.2.

El *Gestor de información* se compone de los módulos de información de clases, de descripción del escenario y del estado de los objetos. Este último ha sido el único que se ha conservado del anterior prototipo. El módulo de información de clases ha sido modificado para almacenar las clases como tarjetas CRC y para proporcionar la interfaz de modificación de las mismas. Esta interfaz es muy sencilla y se compone de las acciones de creación y eliminación de responsabilidades y colaboraciones de una clase. El módulo de Descripción del escenario es el que contiene toda la información básica referida al escenario que está siendo simulado.

El *Gestor de información* proporciona una interfaz de observadores para informar de las modificaciones que se produzcan en toda esta información. En particular se suscribirán a estas interfaces los inventarios que deberá actualizar la vista de los mismos cuando se produzcan modificaciones de diseño en las clases.

El *Gestor de escenarios* también ha sufrido cambios con respecto al desarrollado para ViRPlay3D, debido fundamentalmente a los cambios de forma-

to de los escenarios. En primer lugar, se ha renunciado al Gestor de mapas de JV^{2}M ya que los mapas de este prototipo son mucho más simples, como se verá en la Sección 6.8.1. Por otro lado, se ha modificado el cargador de información de clases, que ahora ha de analizar el nuevo formato XML que genera la herramienta *CRC Card Editor*, descrita en la Sección 6.8.2. Por último, el cargador de información dinámica es muy similar al anterior ya que se ha conservado el formato del anterior prototipo. La única diferencia estriba en que un escenario se puede componer de varias sesiones, por lo que el *Gestor de escenarios* también ha de ser soportar la inicialización de sesiones bajo demanda del *Gestor de role-play*.

De cara al subsistema de Hotspots, se ha desarrollado una interfaz común para el alumno y para el instructor. Las acciones que se pueden realizar desde ambos interfaces son las mismas aunque, dependiendo del rol seleccionado, se podrán realizar unas u otras. Esta interfaz ha sido desarrollada utilizando los módulos de GUI y de aplicación de la arquitectura de *El Laberinto*. La primera se encarga del motor gráfico y de la interfaz de usuario, mientras que la segunda se encarga de la gestión de los estados de la aplicación y de la entrada de usuario. Como ya se ha mencionado, el motor empleado para el dibujado tanto de la interfaz en 2D como de la escena en 3D es Nebula 2. Gran parte de la gestión de modelos en 3D y del aspecto de la interfaces en 2D se ha realizado a través de la interfaz de *scripting* en Tcl que proporciona Nebula 2, lo que ha facilitado el desarrollo del aspecto del entorno de manera independiente al resto de la aplicación.

Esta interfaz también es observador del *Gestor del mundo*, al igual que ocurría en ViRPlay3D. De este modo, todos los eventos que se produzcan en el mundo tendrán su acción asociada en la interfaz de los alumnos y del instructor, ya sea modificando el aspecto y la ubicación de las entidades, ya sea cambiando el estado de la aplicación que gobierna el bucle principal del entorno virtual.

El *Intérprete de comandos* también ha sido modificado para ViRPlay3D 2 ya que este prototipo tiene un mayor número de comandos que el anterior. Dependiendo de los comandos que reciba, este módulo los enviará a uno y otro módulo de los que conforman el entorno virtual. Las acciones relacionadas con la selección de escenarios, asignación de roles, acciones del instructor y acciones que impliquen la activación de la herramienta de consenso serán redirigidas al *Gestor pedagógico*. Todas las relacionadas con la creación de mensajes, con la consulta de información a través de los inventarios, movimiento de los avatares y lanzamiento y recogida de la pelota serán trasladadas al *Gestor del mundo*. Por último, las acciones relacionadas con la herramienta de modificación de clases serán pasadas al módulo *Modificador de clases*.

El *Gestor pedagógico* de ViRPlay3D 2 es responsable de distintas funcionalidades. Por un lado, se encarga de controlar la asignación de roles de

un escenario seleccionado, de modo que no se podrá simular este escenarios hasta que todos los roles sean cubiertos por los alumnos o haya alguien que interprete el rol del instructor. También es responsable de gestionar las acciones del instructor como la interpretación de un rol no asignado o la simulación de un escenario automáticamente cuando el entorno está siendo usado para la evaluación de un escenario.

En el *Gestor pedagógico* de ViRPlay3D 2 se han incluido dos módulos nuevos. Por un lado, este módulo contiene la herramienta de consenso. Este módulo deberá ser configurado indicando el tiempo de espera de la votación, cuál es la proporción para alcanzar el consenso y si se permite o no realizar las acciones consensuadas de manera unilateral. El módulo responsable de la herramienta de consenso se comunicará con el *Gestor del mundo* para activar y desactivar la herramienta y mostrar todos los mensajes relacionados con la misma, así como para ejecutar las acciones que se ha aceptado que pueden ser realizadas.

El otro módulo contenido en el *Gestor pedagógico* para ViRPlay3D 2 es el repositorio de patrones de diseño. Si el escenario que actualmente está siendo simulado está marcado con información sobre algún patrón de diseño, el *Gestor pedagógico* presentará la información relacionada con el mismo que contenga en este repositorio en respuesta a un evento de presentación de información adicional.

El módulo restante del subsistema Hotspots es el módulo *Modificador de clases*. Este módulo será el responsable de gestionar todas las acciones de modificación de las clases de los alumnos y de comunicarlas al *Gestor de información* a través de su interfaz de modificación. Como se ha podido ver, esta interfaz sólo permite la creación y eliminación de responsabilidades y de colaboradores, por lo que esta herramienta será la responsable de transformar la modificación de una responsabilidad —esta es una modificación que los alumnos pueden realizar, según el diseño descrito en secciones anteriores— en las operaciones de eliminación de la responsabilidad y de creación de la nueva responsabilidad con el mismo nombre que la antigua. Ya que la modificación de clases requiere del consenso de los alumnos, se necesitará una confirmación antes de realizar las acciones. Ésta se requerirá al *Gestor Pedagógico* usando la herramienta de consenso.

En lo que respecta a la arquitectura de red, ViRPlay3D 2 se ha desarrollado siguiendo una política de cliente-servidor. La forma de que varios alumnos puedan simular un escenario es la siguiente:

- Uno de los alumno configura ViRPlay3D 2 en modo servidor y selecciona un escenario. De esta forma, queda a la espera de que se conecten los clientes.

- El resto de los alumnos han de configurar ViRPlay3D 2 en modo cliente y conectarse a la máquina del que está en modo servidor. Aquí, selec-

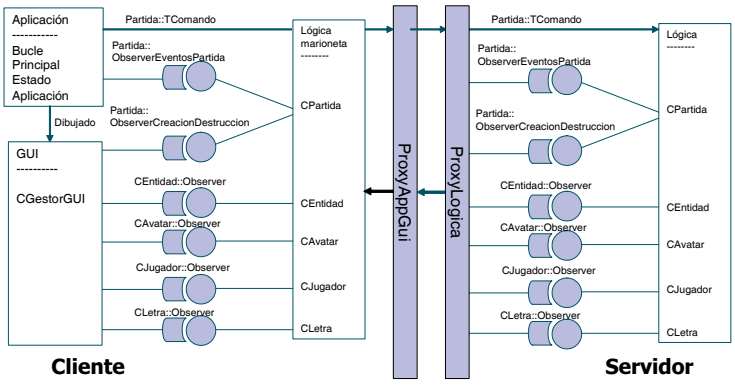


Figura 6.40: Vista de alto nivel de la arquitectura de red de *El Laberinto*.

cionarán el rol que desean interpretar del escenario seleccionado por el primero.

- Una vez que todos los roles han sido cubiertos —o existe un instructor— el primer alumno inicia la simulación del escenario. A partir de ahora, todos los alumnos podrán actuar dentro del entorno virtual.

Para el desarrollo de esta arquitectura de red se han tenido que añadir los nuevos estados de la aplicación relacionados con la red, así como los proxys para la comunicación en red tal y como se describió en la Sección 5.8 del Capítulo 5, adaptando la arquitectura de red de *El Laberinto* que se puede ver en la Figura 6.40.

6.8. Herramientas de autoría para ViRPlay3D 2

Los escenarios de ViRPlay3D 2 son mucho más ricos en información que los de su predecesor. Como se puede extraer de las anteriores secciones, un escenario necesita de una gran cantidad de información para poder ser simulado dentro de ViRPlay3D 2. Haciendo una recopilación de toda esta información, un escenario de ViRPlay3D 2 se compondría de los siguientes datos:

- Un mapa de la escena que contenga la ubicación inicial de las entidades dentro del entorno virtual.
- Una descripción textual del escenario. Esta descripción aparece al iniciar el escenario y cuando cualquier alumno consulta el inventario de la entidad contenedora de la información del escenario.
- Un diagrama de clases que muestra el diseño inicial que se va a usar en el escenario. Se usa en las mismas circunstancias que la descripción textual.

- Información sobre el patrón de diseño asociado al escenario, si existe. Esta información se compone del diagrama de clases general del patrón y de un diagrama de secuencia de su comportamiento en ejecución.
- Información estática de cada una de las clases empleadas en la simulación del escenario. Esta información es guardada en forma de tarjetas CRC.
- Qué roles son interpretables por los alumnos. Son los objetos que los alumnos pueden elegir durante la fase de asignación de roles. Como ya se ha mencionado, un escenario puede contener varios objetos, algunos de los cuales no son de interés para que un alumno lo simule.
- Información dinámica del escenario. Dentro de esta información se incluye de cuántas sesiones se compone un escenario, el estado inicial de los objetos para cada una de las sesiones (incluyendo qué objetos aún no han sido construidos) y el comportamiento de los objetos que no van a ser simulados por los alumnos.

Se puede ver que es muy importante proporcionar a los instructores herramientas que faciliten el desarrollo de toda esta información. Si cada escenario necesita ser creado a mano es muy probable que pocos instructores hicieran uso de ViRPlay3D 2 ya que sería demasiado tedioso la construcción de nuevos escenarios. Para facilitar este trabajo se han desarrollado distintas herramientas para la creación de los distintos tipos de información. Algunas de estas herramientas también han sido pensadas para que, en la medida de lo posible, se pueda reutilizar parte de los datos creados para el prototipo anterior. A continuación describiremos cada una de las herramientas desarrolladas.

6.8.1. *GV Map Editor*: autoría de mapas

Un mapa contiene la ubicación inicial de cada una de las entidades que aparecen en el escenario así como información relacionada con el aspecto visual de estas clases. Para el anterior prototipo se utilizó *Worldcraft* para este cometido. Se ha visto que esta herramienta era demasiado compleja para lo poco que se necesitaba realizar con ella. Además, el cambio de motor de juego nos ha permitido olvidarnos del formato propio de los mapas generados con la anterior herramienta, así como del postprocesado que había que realizar antes de poder hacer uso de los mapas.

Para el desarrollo de mapas se ha utilizado un editor mucho más sencillo. Este editor, llamado *GV Map Editor*, es una modificación del editor desarrollado por los alumnos del Máster de Videojuegos de la Universidad Complutense de Madrid durante el curso 2005-2006. Es un editor en dos dimensiones basado en un mundo en forma de cuadrícula y de muy fácil uso:

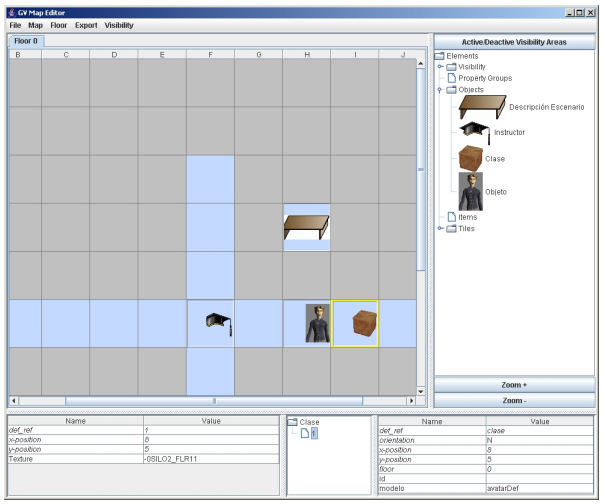


Figura 6.41: *GV Map editor*: herramienta de edición de mapas para ViRPlay3D 2.

Tan solo hay que seleccionar el tipo de entidad que se desea crear, ubicar dicha entidad en la cuadrícula y configurar sus propiedades. El aspecto de este editor es el que se muestra en la Figura 6.41.

Algunas entidades, como la ubicación inicial del instructor o de la entidad con información del escenario, no requieren más que de la casilla que van a ocupar inicialmente dentro de la escena y su orientación. Sin embargo, las entidades de clases y objetos requieren de información adicional. Para una clase hay que definir su ubicación inicial y su orientación, su nombre y el modelo gráfico asociado a las instancias de esa clase. Para un objeto también hay que definir su ubicación inicial, orientación y nombre. Sin embargo, en lugar de definir su modelo gráfico se definirá la clase a la que pertenece. De aquí es de donde se extraerá su apariencia en ViRPlay3D 2.

Una vez creado el mapa, éste se exportará a un archivo en XML, mucho más sencillo que el BSP empleado en ViRPlay3D. El formato de este archivo se puede ver en la Figura 6.42. Estos mapas pueden ser empleados en distintos escenarios que compartan las mismas clases y objetos.

6.8.2. *CRC Card Editor*: autoría de información estática

El cambio de la información de las clases entre las dos versiones de ViRPlay3D ha ocasionado que el material generado por el primero no sea inmediatamente reutilizable para el segundo de los prototipos. Además, la orientación pedagógica que se ha dado a ViRPlay3D 2, en la que se ha hecho más énfasis en el trabajo de diseño de los alumnos, hace que ya no sea tan útil emplear aplicaciones ya implementadas. Para este nuevo entorno es conveniente que el instructor pueda crear escenarios basados simplemente

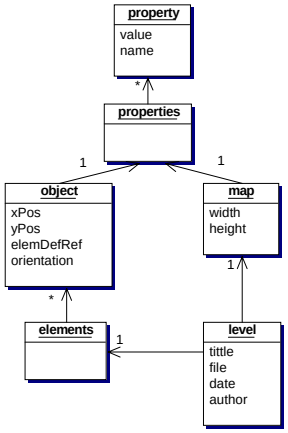


Figura 6.42: Formato XML de los mapas generados con *GV Map Editor*.

en el boceto de un diseño de clases sobre el que los alumnos trabajarán y realizarán modificaciones para desarrollar el diseño final.

Para el apoyo a la autoría de escenarios en ViRPlay3D 2 se ha desarrollado la herramienta *CRC Card Editor*. Es un editor gráfico para la creación de tarjetas CRC y su exportación al formato XML empleado por ViRPlay3D 2. Como se puede ver en la Figura 6.43, esta herramienta es muy sencilla de emplear y con ella se puede, de manera visual, crear nuevas tarjetas, añadir comentarios a las mismas y añadir y modificar tanto responsabilidades como colaboradores. Las tarjetas creadas, además de poder ser exportadas para ser incluidas como información estática dentro de un escenario de ViRPlay3D 2, se pueden guardar como imágenes y se pueden imprimir, por lo que esta herramienta también puede servir de apoyo para el desarrollo de sesiones presenciales de diseño orientado a objetos basadas en role-play.

Para no arrojar por la borda el trabajo de extracción de información del anterior prototipo, *CRC Card Editor* también es capaz de importar el formato utilizado en ViRPlay3D y generar las tarjetas CRC asociadas a las mismas. Luego será tarea del instructor editar estas tarjetas para completar la información que falte o para eliminar aquella que considere sobrante. En particular, una de las tareas que deberá realizar el instructor será incluir la información relacionada con las colaboraciones entre clases ya que la herramienta no se responsabiliza de la generación de este tipo de información.

Finalmente, *CRC Card Editor* exporta toda la información generada visualmente a un archivo XML interpretables desde ViRPlay3D 2. La estructura de este archivo se puede ver en la Figura 6.44

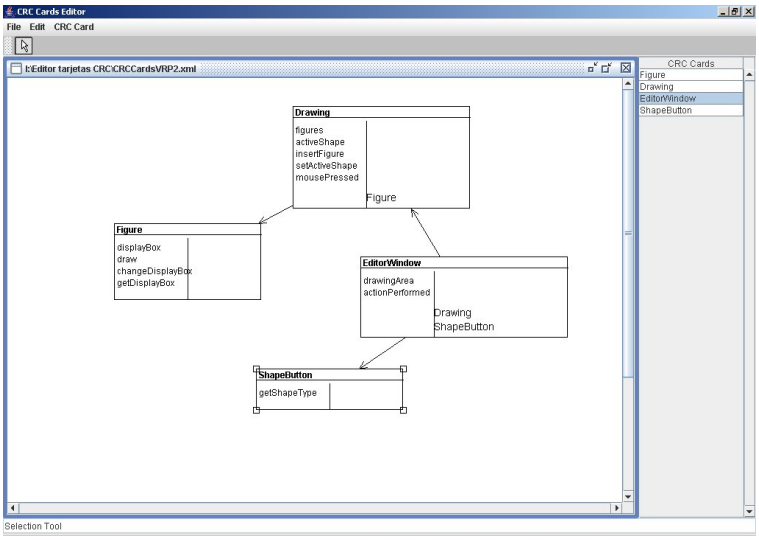


Figura 6.43: *CRC Card Editor*: herramienta para la generación de tarjetas CRC para ViRPlay3D 2.

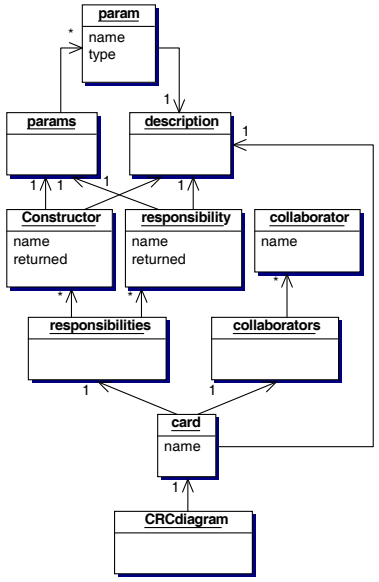


Figura 6.44: Formato XML de los diagramas generados con *CRC Card Editor*.

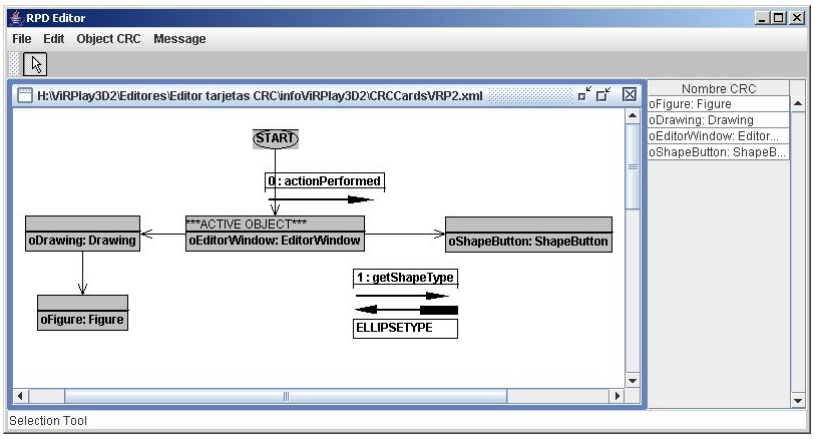


Figura 6.45: *RPD Editor*: herramienta para la creación de RPDs.

6.8.3. *RPD Editor*: creación de diagramas de role-play

A medida que los escenarios son ejecutados por los alumnos se va generando un RPD con la información de los pasos de mensajes producidos. Este RPD se puede consultar a través del inventario del marcador y tiene un aspecto como el de la Figura 6.31. Para la creación de este RPD se usa la herramienta *RPD Editor* (Figura 6.45). Esta herramienta es un editor visual de RPDs que se emplea en varias fases:

- Durante la generación de escenarios, el instructor carga las tarjetas CRC creadas con *CRC Card Editor* y crea las instancias que van a intervenir en el escenario de cada una de estas clases. Con esto crea el esqueleto inicial del RPD e inicializa los objetos del escenario. Si el escenario contiene varias sesiones, este proceso se repetirá para cada una de las mismas. Esta información se guarda finalmente en un formato muy similar al de la información dinámica de ViRPlay3D.

- Al iniciar un escenario, ViRPlay3D 2 carga en *RPD Editor* la información anteriormente creada. Durante la ejecución del escenario dentro del entorno virtual, el *Gestor de role-play* irá pasando todos los eventos relacionados los pasos de mensaje al núcleo de *RPD Editor*. Éste irá actualizando el RPD con esta información. Si alguien solicita visualizar el estado actual del RPD, *RPDEditor* generará el diagrama que aparecerá en el inventario del marcador.

- Al final de la simulación, *RPD Editor* genera un archivo XML con el mismo formato que el visto en la Sección 6.3.3 y que será el resultado final del escenario ejecutado.

Este editor puede ser también empleado durante las clases presenciales de role-play, ya que es un editor visual muy sencillo sobre el que el instruc-

tor también puede crear manualmente un RPD. Además, debido al formato en el que exporta los RPDs, este editor también sirve como sustituto a la herramienta de generación de información dinámica vista en la Sección 6.3.3 o para refinar la información generada por esta herramienta automática.

6.9. Ejemplo de uso de ViRPlay3D 2

Vamos a ilustrar el funcionamiento de ViRPlay3D 2 describiendo un escenario de uso del mismo. Para este prototipo se han desarrollado algunos escenarios extraídos de la experiencia del uso de las sesiones de role-play para la enseñanza de patrones de diseño (Jiménez-Díaz et al., 2007d). En particular, se va a mostrar parte de la sesión empleada para la enseñanza del patrón *Prototipo*. Como ya se señaló, durante estas sesiones presenciales se ha utilizado un editor gráfico sencillo como caso de estudio. Las clases que conforman este editor son similares a las que se describió en la Sección 4.4:

`EditorWindow` Es la clase responsable de coordinar la aplicación. Contiene los botones de la barra de herramientas y es responsable de responder a los eventos de pulsación de dichos botones.

`ShapeButton` Es la clase que representa cada uno de los botones de creación de figuras que hay en la barra de herramientas. En ejecución, existe una instancia para cada uno de los tipos de figuras que se pueden crear.

`Drawing` Es la clase que representa el lienzo del editor gráfico y se encarga de almacenar las figuras que son creadas. Es responsable de dibujarlas y de responder a los eventos que se producen cuando el usuario pulsa sobre el lienzo.

`Figure` Es la superclase que representa a las figuras que se pueden dibujar usando el editor gráfico. Cada figura es responsable de dibujarse y de saber cómo modificar el rectángulo que la circunscribe (llamado área de presentación). Bajo esta clase aparecen las subclases `EllipseFigure` y `RectangleFigure`, que son las particularizaciones de `Figure` para las elipses y los rectángulos, respectivamente.

Los alumnos van a simular un escenario en el que se representa cómo se realiza la creación de una figura nueva en este editor. Una vez que han seleccionado el escenario y cada uno ha seleccionado un rol pasan a ver la información asociada al escenario. Aquí pueden ver el diagrama de clases que aparece en la Figura 6.46. También pueden leer la siguiente descripción:

> El usuario desea crear una nueva elipse en el editor gráfico. La creación de figuras se realiza en dos fases. Primero, el usuario selecciona el tipo de figura que desea crear pulsando sobre uno

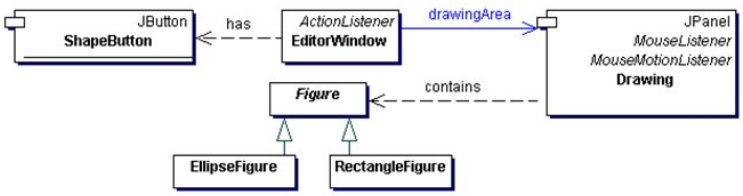


Figura 6.46: Diagrama de clases inicialmente propuesto para el desarrollo del escenario.

> de los botones de la barra de herramientas (puede crear rectángulos o elipses). En este caso, el usuario ha pulsado el botón de creación de elipses. Después, el usuario pulsa sobre el lienzo y la aplicación crea una elipse cuyo área de presentación tiene la esquina superior izquierda en la posición donde está el puntero del ratón. La primera pulsación del ratón la recibe `EditorWindow` a través de la responsabilidad *actionPerformed*, mientras que la segunda pulsación es tratada por la responsabilidad *mousePressed* de la clase `Drawing`.

Una vez que los alumnos han leído estas instrucciones pasan al entorno virtual y da lugar la primera sesión del escenario. Aparecen seis cajas —una por cada una de las clases presentadas anteriormente— y cuatro objetos: *oShapeButton*, *oEditorWindow*, *oDrawing* y *oFigure*. Este último puede interpretar el rol de cualquiera de las subclases de `Figure`. Además, el avatar de este objeto aparece transparente ya que este objeto no ha sido creado todavía. En el escenario también hay un escritorio donde los alumnos pueden consultar la descripción del escenario anteriormente vista.

Además de los avatares de los objetos controlados por usuarios hay un avatar que representa al objeto *oEvent1*, que es instancia de la clase `ActionEvent` y que es controlado por el entorno virtual. La pelota aparece en manos de *oEditorWindow* y en el entorno aparece sobre impresionado el mensaje: *"oEditorWindow, execute actionPerformed using oEvent1"*. Este alumno no sabe por dónde empezar y consulta su inventario. Aquí puede leer la descripción de la responsabilidad *actionPerformed* para saber algo más sobre ella y puede comprobar que sus colaboradores son las clases `ActionEvent`, `ShapeButton` y `Drawing`.

Para saber algo más sobre *oEvent1*, *oEditorWindow* apunta a este objeto y accede a su inventario mediante la acción *Mirar a*. Aquí lee que este objeto sólo tiene la responsabilidad *getSource*, que "devuelve una referencia al objeto que ha generado el evento". El alumno que controla a *oEditorWindow* considera que éste puede ser un buen punto para empezar la simulación. Por tanto, selecciona esta responsabilidad del inventario de *oEvent1* y pulsa sobre el botón de creación de mensajes. Lo único que ha de hacer es in-

dicar que el tipo de mensaje es de invocación, ya que esta responsabilidad no tiene parámetros. Una vez hecho esto, el alumno apunta hacia *oEvent1* y, usando el controlador de potencia, lanza la pelota hacia este objeto. En el entorno aparecerá sobreimpresionado el mensaje *"oEvent1, execute getSource"*. Éste recoge la pelota cuando está cerca de él y, pasados unos instantes, este objeto devuelve la pelota rodando a *oEditorWindow* mientras que aparece sobreimpresionado el siguiente mensaje: *"getSource is executed, returning oShapeButton"*.

oEditorWindow apunta a la pelota y, cuando está lo suficientemente cerca de ella, realiza la acción de recogerla. Ahora *oEditorWindow* sabe que el evento ha sido producido por *oShapeButton*, por lo que apunta hacia él y despliega su inventario en busca de más información. Comprueba que este objeto tiene una única responsabilidad llamada *getShapeType*, por lo que crea un mensaje usando esta responsabilidad y lanza la pelota a *oShapeButton*.

Mientras que *oEditorWindow* realizaba las acciones anteriormente descritas, *oShapeButton* se ha dedicado a revisar el inventario del resto de avatares que hay en la escena. En particular, ha descubierto que la responsabilidad *setActiveShape* de la clase `Drawing` sirve para "fijar el tipo de figura que va a ser creada durante la siguiente operación de creación". *oShapeButton* también ha visto que dispone de un atributo *shapeType* con valor ELLIPSETYPE dentro del estado de su objeto por lo que decide que él se responsabilizará de notificar el tipo de figura que se ha de crear a *oDrawing*.

Cuando *oShapeButton* muestra el inventario de *oDrawing* e intenta crear el mensaje que va a pasar observa que el botón de creación de mensajes no aparece activo. Confuso, utiliza la herramienta de comunicación para comentar a sus compañeros que no es capaz de construir el mensaje con el que fijar el tipo de figura en *oDrawing*. *oFigure* contesta que sólo se pueden enviar mensajes a los colaboradores y le pregunta: *"oShapebutton, ¿es `Drawing` uno de tus colaboradores?"*, a lo que éste contesta que no y sugiere a *oEditorWindow* que se asegure de que él sí tiene a `Drawing` entre sus colaboradores.

Ya que ve que él no puede enviar este mensaje, *oShapeButton* decide enviar el mensaje de retorno a *oEditorWindow* y que éste se responsabilice del siguiente mensaje. En este mensaje de retorno, *oShapeButton* devuelve ELLIPSETYPE como valor retornado. *oEditorWindow* sí tiene a `Drawing` como colaborador por lo que decide crear el mensaje *setActiveShape* con ELLIPSETYPE como valor del parámetro y lanzar la pelota a *oDrawing*. Cuando éste la recoge, entra en su inventario y actualiza su atributo referente al tipo de figura que va a crear usando el valor del parámetro pasado. *oDrawing* decide devolver el control de la ejecución a *oEditorWindow*, creando el mensaje de retorno y devolviéndole la pelota.

oEditorWindow decide que la primera fase del escenario ha terminado, por lo que solicita la finalización del escenario. Esto hace que se active la herramienta de consenso. Todos los alumnos aceptan la finalización del esce-

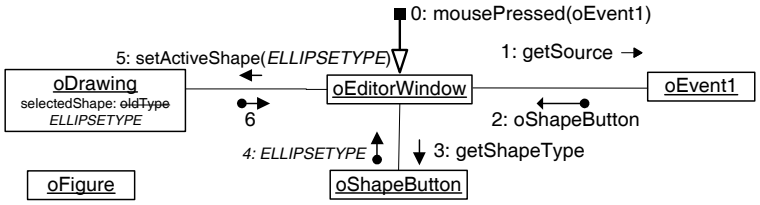


Figura 6.47: RPD creado tras la primera parte de la sesión de role-play.

nario, por lo que la pelota desaparece. Tras la finalización de esta sesión los alumnos pueden desplegar el inventario del marcador —cuyo valor actual es 6, ya que se han producido seis pasos de simulación— y ver un RPD similar al de la Figura 6.47

Pasados unos segundos, da comienzo la segunda sesión del escenario y aparece un nuevo avatar llamado *oEvent2*, instancia de `MouseEvent`, controlado por el entorno. La pelota aparece ahora en manos de *oDrawing* y en el entorno aparece sobreimpresionado el mensaje *"oDrawing, execute mousePressed using oEvent2"*. Primero, el alumno responsable de *oDrawing* revisa su inventario para ver en qué consiste la responsabilidad que le han pedido ejecutar. También observa que sus colaboradores son `MouseEvent` y `Figure` y que su estado contiene un atributo con el valor ELLIPSETYPE que representa el tipo de figura que ha de ser creada por el editor. Para seguir sabiendo qué es lo que ha de hacer, revisa ahora el inventario de *oEvent2*. En la tarjeta CRC de este objeto puede ver que la clase `MouseEvent` dispone de una única responsabilidad llamada *getPoint* que devuelve un objeto de la clase `Point` con las coordenadas del lugar en el que el usuario ha pulsado dentro del lienzo del editor gráfico.

oDrawing cree saber qué es lo que ha de hacer. Apunta hacia la caja con el rótulo `EllipseFigure` y despliega su inventario. Aquí pulsa sobre el botón de creación de objetos y selecciona el constructor sin argumentos. Además, el alumno indica que el identificador de la nueva instancia será *oFigure*. Una vez fuera del inventario, el alumno vuelve a apuntar hacia la caja y lanza la pelota ajustando la fuerza para que caiga sobre la caja. En el entorno aparece sobreimpresionado el mensaje *"EllipseFigure, create oFigure"* y cuando la pelota toca la caja, el avatar de *oFigure* es teletransportado automáticamente junto a la caja y retorna a su apariencia normal. La pelota sale rodando de la caja y vuelve a *oDrawing*, que la recoge.

Ahora *oDrawing* piensa que lo siguiente que ha de hacer es invocar un mensaje sobre sí mismo. Utiliza la acción para ver su propio inventario y selecciona la responsabilidad *insertFigure* para crear el mensaje. Lo configura para que *oFigure* sea el valor del parámetro que acompaña a esta responsabilidad. Una vez creado, lanza la pelota y la recoge él mismo, apareciendo en el entorno el mensaje *"oDrawing, execute insertFigure using oFigure"*. Ahora

el marcador muestra un 3.

oFigure decide intervenir en la simulación usando la herramienta de comunicación. Con ella explica a *oDrawing* que no está de acuerdo con el último paso de mensaje. Al revisar su inventario ha visto que tiene un atributo que representa su esquina superior izquierda y que ahora mismo tiene el valor de un punto con coordenadas (0,0). Comenta que, según las instrucciones que ha podido consultar en la entidad del escritorio, este atributo debería apuntar al lugar donde el usuario ha pulsado antes de que la figura sea insertada en el lienzo. Los alumnos instan a *oDrawing* a que deshaga la última acción y que modifique el área de presentación de *oFigure* antes de pasar el mensaje *insertFigure*.

Siguiendo el consejo del resto de alumnos, *oDrawing* realiza la acción de deshacer el último paso de mensajes. Tras esto se lanza la herramienta de consenso, informando de que *oDrawing* quiere realizar la acción de deshacer un paso de mensaje. Todos los alumnos aceptan la acción y ésta es realizada. Esto se puede corroborar observando que el marcador ahora muestra un 2 y revisando su inventario para comprobar que esta acción ha sido deshecha.

oDrawing apunta hacia *oEvent2* y despliega su interfaz para crear el mensaje *getPoint*. Lanza la bola, que es automáticamente recogida por *oEvent2*, y éste la devuelve rodando con el mensaje *"getPoint is executed, returning aPoint"*. Luego, *oDrawing* apunta a *oFigure*, muestra su inventario y revisa su tarjeta CRC. Observa que tiene una responsabilidad llamada *changeDisplayBox* que recibe dos puntos y que es la responsable de cambiar el area de presentación de una figura. Usando la herramienta de comunicación, *oDrawing* lanza una pregunta: ¿Por qué no añadir una responsabilidad que también cambie el área de presentación pero que reciba la esquina superior izquierda y el alto y el ancho de dicho área? *oShapeButton* sugiere que *oDrawing* cree un nuevo punto con la anchura y la altura y que pase el mensaje con la antigua responsabilidad. Sin embargo, *oShapeButton* no se ha dado cuenta de que esto no lo puede hacer porque la clase `Point` no aparece entre los colaboradores de `Drawing`. *oFigure* también sugiere que se añada a la clase `Point` como colaboradora de la clase `Drawing`. *oDrawing* replica que esto provocaría un acoplamiento innecesario entre las clases `Drawing` y `Point`. El resto de alumnos aceptan la argumentación de *oDrawing*.

oFigure apunta hacia la clase `Figure`, muestra su inventario y desde él accede a la herramienta de modificación de clases, con la que añade la nueva responsabilidad, incluyendo una descripción de los motivos de esta inclusión para la futura evaluación del instructor. Cuando el alumno termina con esta acción el entorno muestra la herramienta de consenso. En este caso, alguno de los alumnos se ha despistado y no ha aceptado la acción, por lo que ésta es rechazada. Sin embargo, ViRPlay3D 2 da la oportunidad a *oFigure* de realizar la acción unilateralmente, guardando este evento anormal en la bitácora. *oFigure* lo acepta y advierte a sus compañeros de lo que ha ocurrido.

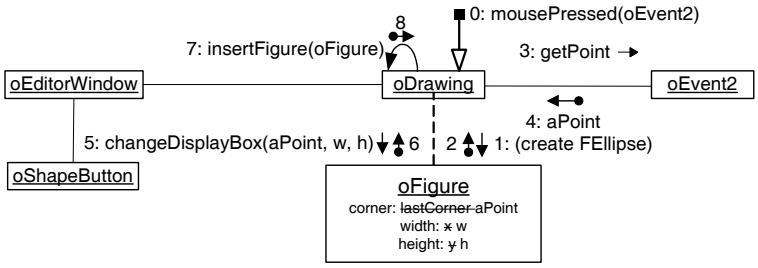


Figura 6.48: RPD creado tras la segunda parte de la sesión de role-play.

Una vez realizada la modificación el escenario es reiniciado, apareciendo un 0 en el marcador y siendo *oDrawing* quien sostiene la pelota y apareciendo en el entorno el primer mensaje. Los alumnos repiten todos los pasos de ejecución anteriores y ahora *oDrawing* crea el mensaje con la nueva responsabilidad *changeDisplayBox* y lanza la pelota a *oFigure*. Cuando este último devuelve la pelota a *oDrawing*, este objeto ejecuta el paso del mensaje *insertFigure*, se produce el retorno de este mensaje y el retorno de *MousePressed*. Mediante la herramienta de comunicaciones todos los alumnos están de acuerdo en dar por concluido el escenario. *oDrawing* es el responsable de concluirlo y todos los demás alumnos lo aceptan cuando la herramienta de consenso aparece. El aspecto final del RPD que representa la simulación del entorno es el que se puede ver en la Figura 6.48.

6.10. Evaluación de ViRPlay3D 2

Una vez desarrollada una implementación de un prototipo que cumple las especificaciones básicas descritas para ViRPlay3D 2, se ha realizado una nueva evaluación, similar a la realizada para ViRPlay3D y descrita en la Sección 6.5. Los objetivos de esta evaluación han sido los siguientes:

- Realizar una estimación de la validez de los nuevos elementos incluidos en la metáfora.
- Analizar ventajas e inconvenientes de los cambios producidos en la información que contiene el entorno virtual.
- Valorar las nuevas mecánicas de interacción.
- Obtener una nueva retroalimentación de los aciertos de esta instanciación y de las mejoras a incluir en futuros entornos virtuales de role-play.

Se ha empleado ViRPlay3D 2 para la simulación de un escenario similar al descrito en el ejemplo de uso de la Sección 6.9 y con ello valorar los nuevos elementos metafóricos, la información contenida en los inventarios y

las capacidades de interacción mostradas. Se solicitó a los participantes que valorasen dichos elementos y que comentasen todos aquellos aspectos que más les habían interesado y los motivos por los que otros no eran considerados apropiados.

Al igual que con ViRPlay3D la evaluación se ha realizado dividiendo cada uno de los aspectos de evaluación —metáfora, información e interacción— en distintos elementos, cada uno de los cuales se ha puntuado utilizando una escala de Likert de 5 valores. Para los elementos de la metáfora se ha vuelto a evaluar la naturalidad de integración de los mismos dentro del entorno, mientras que para los otros dos aspectos se ha evaluado su utilidad para la comprensión y revisión de un diseño de clases.

En esta evaluación participaron los mismos individuos que en la realizada para ViRPlay3D, lo que les ha permitido valorar realmente las diferencias existentes entre los dos entornos diseñados. Estamos satisfechos por comprobar que las modificaciones realizadas han mejorado, aún más si cabe, los resultados conseguidos con el primer entorno.

La metáfora ha sido muy positivamente evaluada en general, obteniendo una puntuación media de 27 puntos de un rango que oscila entre los 6 y los 30 puntos. La valoración individual de cada uno de los elementos de la metáfora se puede ver en la Tabla 6.4. Se observa que en general se han mejorado las valoraciones de los elementos comunes a ambos, sobre todo en cuanto a la metáfora mejorada de los objetos y de las clases. El peor valorado, aunque con una puntuación muy alta, ha sido el escritorio utilizado para contener toda la información referente al escenario que está siendo simulado. De los comentarios se ha extraído que algunos participantes consideran que esta información ha de estar más fácilmente accesible desde la interfaz y que no consideran útil que haya que andar buscando por el entorno virtual la ubicación del escritorio para poder acceder a esa información.

La valoración de la información contenida en el entorno, en la Tabla 6.5 también ha tenido una evaluación general muy buena, con una puntuación media de 18,7 puntos de un rango que oscila entre los 4 y los 20 puntos. Ha mejorado ostensiblemente la puntuación relacionada con las clases ya que para este entorno las tarjetas CRC ha sido el tipo de información mejor valorada —4,83 sobre 5 puntos. También se observa que hemos acertado con la inclusión de los RPDs, ya que estos han sido el segundo tipo de información mejor valorado, con una puntuación similar a la de las tarjetas CRC.

Las nuevas mecánicas de interacción han tenido una gran aceptación, como se desprende de las puntuaciones que se resumen en la Tabla 6.6. La más valorada ha sido la herramienta de comunicaciones, con una puntuación media de 4,83 sobre 5. Esta herramienta también ha sido la más apreciada en los comentarios de los participantes. También han encontrado de gran utilidad la herramienta de modificación de clases —4,75 puntos de media sobre un total de 5— ya que consideran que es de especial utilidad para integrar

Tabla 6.4: Distribución de las puntuaciones y puntuación media de los elementos de la metáfora de ViRPlay3D 2. Los elementos aparecen ordenados de mayor a menor puntuación media.

Elementos de la metáfora	N	Frecuencia 1	2	3	4	5	Media	Desviación Típica
Inventarios	12	0	0	1	2	9	4.67	0.65
Objeto	12	0	0	1	3	8	4.58	0.67
Clase	12	0	0	0	5	7	4.58	0.51
Mensaje	12	0	0	0	6	6	4.50	0.52
Objeto activo	12	0	1	0	3	8	4.50	0.90
Descripción del escenario	12	0	1	1	5	5	4.17	0.94

Tabla 6.5: Distribución de las puntuaciones y puntuación media de la información contenida en ViRPlay3D 2. Los elementos aparecen ordenados de mayor a menor puntuación media.

Información disponible	N	Frecuencia 1	2	3	4	5	Media	Desviación Típica
Tarjetas CRC	12	0	0	1	0	11	4.83	0.58
RPDs	12	0	0	1	0	11	4.83	0.58
Estado del objeto	12	0	0	1	2	9	4.67	0.65
Descripción del escenario	12	0	1	0	5	6	4.33	0.89

el diseño orientado a objetos dentro del entorno virtual. El uso de apuntar a las entidades para acceder a sus inventarios ha sido un cambio con respecto a ViRPlay3D acertado, como se desprende de la alta puntuación obtenida por esta mecánica —4,67 sobre 5— y de los comentarios que los participantes han hecho al respecto sobre la misma. Sin embargo, la puntuación más baja la ha obtenido la mecánica concerniente a que los alumnos sean responsables de actualización del estado de los objetos. En los comentarios afirman que esto debería ser automatizable y que puede ser fuente de errores si alguno de los alumnos olvida realizar las actualizaciones en el momento adecuado.

En los comentarios, los participantes han destacado el aspecto colaborativo de ViRPlay3D 2. Les ha gustado especialmente la idea de desarrollar un entorno multiusuario en el que los alumnos puedan colaborar y comunicarse para realizar el diseño de una aplicación orientada a objetos. También consideran que la posibilidad de modificar las clases dentro del propio entorno es

Tabla 6.6: Distribución de las puntuaciones y puntuación media de las mecánicas de interacción en ViRPlay3D 2. Los elementos aparecen ordenados de mayor a menor puntuación media.

Mecánicas de interacción	**N**	**Frecuencia**					**Media**	**Desviación Típica**
		1	2	3	4	5		
Herramienta de comunicación	12	0	0	0	2	10	4.83	0.39
Herramienta de modificación de clases	12	0	0	0	3	9	4.75	0.45
Creación de mensaje	12	0	0	0	4	8	4.67	0.49
Lanzamiento de la pelota	12	0	0	0	4	8	4.67	0.49
Apuntar con punto de mira	12	0	1	0	2	9	4.58	0.90
Mirar a	12	0	0	1	4	7	4.50	0.67
Cambiar estado de un objeto	12	0	2	1	2	7	4.17	1.19

un acierto aunque algunos han apuntado que no les parece apropiado tener que reiniciar la simulación cada vez que se realiza alguna de estas modificaciones.

También han valorado muy positivamente la presencia del RPD. Lo consideran de gran ayuda a medida que se va simulando el escenario. Sin embargo, algunos participantes han pedido la inclusión de más información de guía. Consideran que un entorno en el que se dispone de tanta libertad debería haber información disponible sobre posibles soluciones de simulación —el aspecto final del RPD— o cuál debería ser el diseño final de las clases tras la refactorización que se debería realizar en un escenario.

Los participantes también han detallado que, aunque la nueva apariencia de los objetos mejora la distinción de los mismos, se debería hacer también distinción entre objetos de la misma clase. Consideran que en un escenario de simulación complejo y debido a la movilidad de los avatares podría ser caótico buscar a un objeto, sobre todo si hay de más de una instancia de la misma clase. Valorarían cualquier tipo de distinción visual, como el cambio del color de su vestuario o algún tipo de marca distintiva.

El lanzamiento de la pelota y su factor lúdico ha generado controversia. Algunos participantes han considerado que es una mecánica poco natural, que genera distracción y que puede ser un obstáculo para el desarrollo de una simulación. Sin embargo, otros consideran que es una mecánica muy útil, que ayuda a amenizar las sesiones de role-play más pesadas. Incluso hay participantes que han pedido que existan más elementos que potencien

la idea de tomar las sesiones de role-play como un juego. Afirman que se podría incrementar la motivación de los alumnos añadiendo un mayor factor de competitividad, haciendo que los alumnos obtengan recompensas por las acciones bien realizadas y penalizaciones por los pasos de ejecución incorrectos. El uso de videojuegos para la enseñanza siempre es un aspecto que genera controversia en la comunidad educativa, por lo que sería necesario realizar una evaluación más completa para saber si de verdad esta mecánica dificulta o mejora las capacidades de enseñanza de los entornos virtuales de role-play.

6.11. Conclusiones

A lo largo de este capítulo se han descrito dos entornos concretos que han puesto en práctica la propuesta de entornos virtuales de role-play para la enseñanza diseño orientado a objetos. Estos entornos han sido diseñados como instanciaciones de la arquitectura general para entornos virtuales de role-play, descrita en el Capítulo 5.

ViRPlay3D, el primero de los entornos, se ha desarrollado como punta de lanza de nuestra propuesta de entornos virtuales de role-play en el ámbito de la investigación relacionada con la enseñanza de la orientación a objetos. ViRPlay3D muestra el traslado de las mecánicas básicas del role-play al entorno virtual. Emplea esta técnica como medio para el análisis de una aplicación orientada a objetos, lo que convierte a ViRPlay3D en una herramienta de visualización de las interacciones que se producen entre objetos dentro de la ejecución de una aplicación.

ViRPlay3D es un entorno en el que la ejecución del escenario es muy guiada. El alumno puede o bien solicitar que se simule la ejecución del siguiente paso de mensajes, o bien interpretar el rol de cualquiera de los objetos puntualmente y decidir el siguiente paso que se ha de realizar. Pedagógicamente, el margen de error del alumno es nulo ya que si se equivoca durante la interpretación del rol el entorno no ejecutará dicho paso de ejecución. Esto facilita que este entorno sea completamente independiente de la existencia de un instructor humano salvo para la creación de los escenarios simulables en el entorno.

La evaluación de ViRPlay3D ha sido muy satisfactoria. Los participantes en la evaluación de ViRPlay3D han destacado de él lo novedoso de trasladar el role-play a un entorno virtual. También han considerado muy interesante la idea de dar una forma visual a conceptos abstractos como clases y objetos y de ponerlos todos dentro de un entorno que el alumno ha de explorar. Además, destacan la importancia de que el alumno acceda a la información tan solo cuando lo necesite. Esta evaluación también ha sido útil para detectar deficiencias en la representación de algunas metáforas, para mejorar ciertas interacciones y para ver que es necesario incluir en estos entornos

más información sobre el estado de la simulación y sobre el escenario que está siendo simulado.

El segundo entorno, ViRPlay3D 2, va un paso más allá y se mete de lleno en el uso del role-play como técnica para poner en práctica el diseño orientado a objetos. Este entorno traslada una versión simplificada de las sesiones de role-play interpretadas en una clase presencial, donde existen varios alumnos que interpretan los roles de los objetos de la aplicación en ejecución. Además, se ha añadido la capacidad de modificar el diseño de clases, de modo que los alumnos disponen de herramientas dentro del propio entorno con las que poder cambiar las responsabilidades y colaboradores asociados a una clase.

ViRPlay3D 2 también ha sido evaluado entre docentes y, al igual que el anterior, ha tenido una valoración muy positiva. Ha solventado gran parte de los problemas encontrados en ViRPlay3D y las nuevas mecánicas de ViRPlay3D 2 han tenido una gran aceptación. Principalmente se ha destacado el valor del entorno como herramienta colaborativa, en la que varios alumnos puedan trabajar simultáneamente.

Los entornos desarrollados trasladan las mecánicas básicas del role-play de manera muy intuitiva, de modo que sean fácilmente identificables. Las representaciones visuales empleadas son sencillas pero permiten una clara distinción de los conceptos elementales de la orientación a objetos. También se destaca la potencia de una mecánica tan sencilla como *Mirar a*, típica de las aventuras gráficas, y el inventario de las entidades para gestionar la información que presenta el entorno. Esta mecánica promueve en el alumno el análisis de las clases y objetos del escenario mediante la exploración del entorno virtual. Él y sólo él decide qué es lo que necesita saber y cuándo lo necesita. Además, le proporciona un control total sobre la presentación de la información disponible en el entorno. Hay que recordar que estos entornos son muy ricos en información por lo que no disponer de este control podría hacer que el alumno se sintiese perdido ante toda la información disponible.

El carácter multiusuario de ViRPlay3D 2 nos ha hecho ver la necesidad de desarrollar herramientas colaborativas que permitan la comunicación y la coordinación de acciones entre los alumnos. Así, algunas de las decisiones de diseño tomadas nos han obligado al desarrollo de herramientas de consenso integradas dentro del entorno que son empleadas a la hora de tomar decisiones críticas. Además, durante el diseño de estas herramientas se puso de manifiesto que pueden provocar situaciones de bloqueo de la simulación del escenario, por lo que hay que proporcionar vías de escape ante situaciones que comprometan la terminación de una simulación.

ViRPlay3D 2 ha sido diseñado intentando eliminar la necesidad de la presencia de un instructor durante el desarrollo de una sesión de role-play. Pero la libertad de acciones que los alumnos tienen en este entorno hacen que la evaluación del resultado final de una simulación sea intratable si no es con la ayuda de un instructor humano. Por este motivo, las simulaciones

generan una gran cantidad de información con la que el instructor podrá evaluar el resultado final. Además, toda esta información ha sido generada de modo que sirva como información de entrada para ViRPlay3D 2. Así, el instructor también podría utilizar el propio entorno para la evaluación. También se ha dado al instructor la oportunidad de trabajar, opcionalmente, con los alumnos durante la simulación de un escenario. Éste puede participar durante la simulación aunque su capacidad de acción dentro del entorno está limitada y se usa principalmente para solventar situaciones de bloqueo de los alumnos.

Los entornos virtuales de role-play se encuentran supeditados al desarrollo de los escenarios ricos en información. La información contenida es costosa de generar de manera manual por lo que es ineludible el desarrollo de herramientas de autoría que faciliten a un instructor la construcción de los escenarios de ejecución. Es tal la importancia que damos a estas herramientas que éstas han sido desarrolladas para ambos prototipos. Cabe destacar que se han diseñado tanto herramientas para la generación desde cero de casos de estudio y de escenarios de simulación como de herramientas de extracción de información estática y dinámica de aplicaciones. Aunque las herramientas creadas para uno y otro prototipo son distintas se ha intentado, en la medida de lo posible, conservar formatos o facilitar la conversión entre ellos y así favorecer la reutilización en ViRPlay3D 2 de los escenarios desarrollados para ViRPlay3D.

Se ha podido observar que la propuesta de entornos virtuales de role-play realizada admite la construcción de entornos que cubren los distintos objetivos de uso del role-play en el diseño orientado a objetos —análisis, creación y revisión del diseño de clases de una aplicación. Podemos señalar que la selección de los ejes de variabilidad de este tipo de entornos ha sido acertada. Las decisiones tomadas para instanciar cada uno de estos ejes posibilitan la construcción de entornos que emplean el role-play con distinto fin. Así, mientras que ViRPlay3D se ha desarrollado de cara al análisis de una aplicación, ViRPlay3D 2 es útil como herramienta para el diseño y revisión de un diseño de clases. Además, las instanciaciones de los ejes de variabilidad también fomentan el desarrollo de distintas aproximaciones pedagógicas. Si ViRPlay3D no da lugar a la equivocación del alumno, ViRPlay3D 2 se sitúa en el extremo opuesto, dándole libertad total de acción. Aproximaciones intermedias también serían viables.

El desarrollo de ambos entornos también demuestra la versatilidad de la propuesta de entornos virtuales que se describió en el Capítulo 5. Ambos entornos son significativamente distintos pero se ajustan a la arquitectura general de los entornos virtuales de role-play. Esto pone de manifiesto que el modelo de arquitectura propuesto es apropiado y que tiene la suficiente flexibilidad como para soportar las distintas decisiones de diseño tomadas de acuerdo a los ejes de variabilidad propuestos.

Capítulo 7

Conclusiones y trabajo futuro

¡Vida, vida! ¿Me oyes? Dame creación...¡Vida!
El jovencito Frankenstein (1974)

En esta memoria de Tesis se ha presentado un estudio sobre la viabilidad de combinar las principales bondades de las herramientas de visualización y de las técnicas de aprendizaje activo para mejorar el proceso de enseñanza de la orientación a objetos. El resultado ha sido un estudio de la aplicabilidad de las técnicas de role-play en entornos virtuales que sirvan de apoyo en la enseñanza de dicho paradigma. Seguidamente pasaremos a exponer las principales conclusiones y aportaciones de dicho trabajo así como posibles líneas de continuación del mismo.

7.1. Conclusiones

La orientación a objetos ha tomado la cabeza entre los paradigmas de programación gracias a que facilita el desarrollo de software a gran escala y de calidad. Esta ventaja proviene del hecho de que la orientación a objetos propone abordar la complejidad del modelado de una aplicación de la manera natural en la que los humanos tratamos la complejidad en el mundo real: descomponiendo el problema complejo en pequeñas piezas con estado y funcionalidad propias que nos sirvan para componer una solución al problema. La popularidad de este paradigma ha hecho que en los últimos años tome una posición predominante dentro de los currículos de enseñanza de Informática.

A primera vista puede parecer que esta naturalidad del paradigma le aporta sencillez. Sin embargo ni la enseñanza ni el aprendizaje de la orientación a objetos están exentos de dificultades. La orientación a objetos ayuda al desarrollo de software cuya principal característica de calidad es que facilita la reutilización. Sin embargo, la creación de software *reutilizable* es mucho

más complicada que la creación de software simplemente *utilizable*.

Obviamente, es necesario que aquellos que se inician en este paradigma alcancen una clara comprensión de lo que los conceptos del mismo, abstractos en la mayoría de los casos, representan en el desarrollo de software. Generalmente estos conceptos se exponen con ejemplos muy sencillos que, en ocasiones, no enseñan al alumno la verdadera potencia del paradigma. La autentica potencia se observa durante el desarrollo de aplicaciones complejas. Por esto hay que tratar que, desde el principio, los alumnos se enfrenten con ejemplos de suficiente complejidad como para que aprecien el valor real de la orientación a objetos.

Una vez que comprenden los conceptos fundamentales, el siguiente paso consiste en aprender a diseñar este tipo de aplicaciones. La descomposición de un problema en objetos y su posterior generalización a clases no es una tarea trivial. Requiere de un trabajo constante del alumno para adquirir experiencia de diseño. Esta experiencia se consigue fundamentalmente de dos formas. La primera es mediante el estudio de ejemplos complejos en los que el alumno pueda observar las decisiones de diseño tomadas por desarrolladores con mayor experiencia para resolver un problema. La forma alternativa de ganar experiencia de diseño es, obviamente, diseñando. La experiencia se adquiere a base de cometer errores de diseño y de resolverlos. De esta forma se consigue que el alumno aprenda a valorar un problema, evaluar las distintas alternativas de diseño que aparezcan y elegir la que más le convenga de acuerdo al problema actual.

Como apoyo a la enseñanza de la orientación a objetos se han desarrollado un gran número de herramientas pedagógicas. Gran parte de ellas se basan en usar la visualización como medio para facilitar el entendimiento de los conceptos abstractos que forman parte de la orientación a objetos. Además, la visualización facilita la comprensión de la naturaleza dinámica de las aplicaciones. Las visualizaciones ayudan a que el alumno observe las interacciones que se producen entre los objetos que conforman la aplicación y cómo el estado de éstos es modificado por dichas interacciones.

Pero la mera visualización no es suficiente para que el alumno comprenda las ideas fundamentales de la orientación a objetos. Estas herramientas, además de apoyarse en la visualización, han de fomentar la interactividad del alumno. Éste no puede ser un mero observador sino que ha de poder participar en las propias visualizaciones, de manera que se sienta parte activa del proceso de enseñanza. Para ello, el alumno ha de tener control sobre las visualizaciones y las herramientas han de proporcionar mecánicas que fomenten la actividad del alumno con la visualización.

Las herramientas basadas en visualizaciones interactivas de apoyo a la docencia de la orientación a objetos se pueden clasificar principalmente en dos tipos: los *entornos de desarrollo integrado pedagógicos* y los *micromundos*.

Los primeros son entornos que, al igual que los entornos de desarrollo integrados profesionales, incluyen varias herramientas de ayuda al desarrollo de programas, como editores de código fuente, compiladores y depuradores. Pero tienen una serie de características adicionales que los hacen especialmente aptos para la enseñanza de la orientación a objetos: son más sencillos que los profesionales, de fácil uso, facilitan la comprensión de los conceptos elementales de la orientación a objetos y proporcionan gran cantidad de información visual tanto de la estructura estática de las clases como del comportamiento en ejecución de aplicaciones.

Los micromundos son entornos habitados por entidades cuyo comportamiento es programado por el alumno de acuerdo al repertorio de operaciones que cada entidad puede realizar. Estos entornos tienen siempre una representación visual que muestra las respuesta del micromundo y de sus entidades a las operaciones que el alumno ha solicitado que las entidades realicen. Mediante estas respuestas visuales se muestran los cambios de estado y el comportamiento de las entidades. Algunos micromundos también permiten que el alumno construya nuevas entidades, ya sea especializando las existentes, ya sea creándolas desde cero. Los más avanzados permiten incluso la creación de nuevos micromundos.

Tanto unos como otros fomentan ampliamente la participación del alumno mediante el desarrollo de actividades de programación. Además, son ricos en visualización y suelen cubrir muchas de las dificultades relacionadas con los conceptos elementales de la orientación a objetos. Sin embargo, prácticamente todas estas herramienta se centran en la programación orientada a objetos, dejando a un lado las tareas de diseño. Además, estas herramientas están pensadas para el trabajo individual del alumno o para que el instructor las use para mostrar la estructura y el comportamiento en ejecución de aplicaciones orientadas a objetos. Únicamente algunas de estas herramientas se han diseñado como medio de apoyo al trabajo colaborativo, como soporte para que los alumnos compartan sus experiencias y juntos resuelvan problemas relacionados con el desarrollo de aplicaciones orientadas a objetos.

En las clases presenciales existen técnicas que fomentan la adquisición de habilidades de los alumnos haciendo que éstos sean parte activa del proceso de aprendizaje. Éstas se conocen como técnicas de aprendizaje activo. Para la enseñanza del diseño orientado a objetos existe una técnica de aprendizaje activo que fomenta especialmente la colaboración entre los alumnos: el role-play. Esta técnica se basa en la metáfora antropomórfica de la orientación a objetos: un objeto es como una persona que tiene un cierto conocimiento propio, sabe realizar una serie de tareas y es capaz de comunicarse y delegar en otros objetos aquellas tareas complejas que no es capaz de resolver por sí solo. En el diseño orientado a objetos, el role-play propone la simulación del comportamiento de una aplicación durante la ejecución de un caso de uso específico. El rol de los objetos es interpretado por los propios alumnos

que se meten en la piel de un objeto y se responsabilizan de completar la simulación del escenario de ejecución propuesto interaccionando con el resto de alumnos participantes en la simulación.

El role-play se usa tanto para tareas de análisis y comprensión como de diseño de aplicaciones orientadas a objetos y se fundamenta en el hecho de que el alumno aprende activamente de sí mismo, metiéndose en el papel del objeto, y del resto de participantes, interaccionando con ellos y discutiendo sobre los pasos de ejecución realizados. Como apoyo al role-play se utilizan las tarjetas CRC, que sirven de ayuda para representar las clases que conforman un sistema software. Estas tarjetas guardan información sobre cuáles son las responsabilidades asociadas a cada clase y cómo colaboran unas clases con otras.

Existen trabajos en los que se describe el uso de las sesiones de role-play como medio para enseñar los conceptos elementales de la orientación a objetos. También existen trabajos que describen el uso de esta técnica para realizar sesiones de diseño. Pero no se ha encontrado información referente al uso de esta técnica en la enseñanza de conceptos más avanzados relacionados con el diseño orientado a objetos, como son los patrones de diseño. Por este motivo, se ha realizado una experiencia con alumnos de la Facultad de Informática de la Universidad Complutense de Madrid para experimentar el uso de esta técnica en la enseñanza de patrones de diseño. Conjuntamente con la realización de la experiencia, se ha realizado una evaluación de la aceptación del role-play entre los alumnos, así como el efecto de la utilización de esta técnica en su aprendizaje.

Los resultados obtenidos durante la evaluación de las sesiones de role-play muestran que los alumnos valoran muy positivamente esta técnica porque se sienten más motivados durante las clases. Además, valoran ser partícipes en tareas de diseño porque ellos mismos son conscientes de que necesitan de la experiencia para aumentar sus conocimientos en esta disciplina. Desde un punto de vista más objetivo, los resultados muestran que estas técnicas mejoran el aprendizaje de los alumnos y que su efectividad reside especialmente en la participación. Se ha constatado que los alumnos que han participado en las sesiones de role-play han obtenido mejores resultados que los que tan sólo han observado pasivamente el desarrollo de las sesiones.

Llegados a este punto se ha decidido hacer converger el uso de herramientas de visualización interactivas destinadas a la enseñanza de orientación a objetos, por un lado, con el role-play como técnica de aprendizaje activo aplicada en la enseñanza de la orientación a objetos, por otro. La fusión de ambos conceptos constituye la aportación global de este trabajo de Tesis: el estudio de la aplicabilidad del role-play en entornos virtuales como medio de enseñanza asistida en el dominio de la orientación a objetos. Esta aportación se puede desglosar en dos partes:

- Realización de una propuesta de entornos virtuales de role-play. Estos

entornos introducen al alumno en un entorno virtual en 3D, con gran capacidad de interacción, rico en información de diseño y en el que se pueden realizar las misma actividades que en las sesiones presenciales de role-play, ya sea solo o en colaboración con otros alumnos.

- Diseño de una arquitectura de alto nivel para el desarrollo de entornos virtuales de role-play. Se han identificado los principales ejes de variabilidad de las sesiones de role-play y se han trasladado al diseño de estos entornos. De este modo se ha propuesto una arquitectura general que permite el desarrollo de entornos que usan el role-play con distintos fines y que emplean distintas alternativas pedagógicas para controlar y evaluar las sesiones desarrolladas por los alumnos.

Para la realización de esta propuesta se han identificado inicialmente los principales elementos de las sesiones de role-play que necesitan una metáfora dentro del entorno virtual. Posteriormente, se ha catalogado la información necesaria durante las sesiones presenciales de role-play y, mediante la metáfora del inventario de una entidad como contenedor de información, se ha distribuido entre las entidades del entorno. Más adelante se han definido las mecánicas de interacción de los entornos virtuales de role-play. Estas mecánicas se emplean para acceder a la información del escenario contenida en el entorno, interactuar con el resto de objetos para completar una simulación mediante el paso de mensajes, modificar el diseño de clases del escenario simulado y la comunicación y colaboración entre usuarios del entorno. Por último, se han analizado las necesidades de autoría de los entornos. Para cada escenario se ha identificado la información sobre ubicación de entidades necesaria para la configuración inicial del entorno; la información, tanto estática como dinámica, de las clases y objetos que van a participar en un escenario; y la información relativa a los pasos de mensaje que se han de dar para completar la simulación de un escenario.

Para el diseño de la arquitectura de los entornos virtuales de role-play primeramente se ha generalizado la estructura de una sesión de role-play. En esta estructura se han identificado aquellos aspectos que pueden variar entre sesiones que persiguen distintos objetivos. Posteriormente se han trasladado estos aspectos variables a la propuesta de entornos virtuales de role-play realizada, obteniendo como resultado un conjunto de ejes de variabilidad para estos entornos. Finalmente se ha realizado una propuesta de arquitectura de alto nivel, basada en una arquitectura genérica para el desarrollo de videojuegos. En esta arquitectura se han identificado los módulos que van a permanecer fijos entre distintas instanciaciones de la arquitectura y aquellos que, por el contrario, van a variar para definir las características concretas del entorno virtual. Estos módulos son los que serán instanciados de acuerdo a las decisiones de diseño tomadas en los ejes de variabilidad de la arquitectura general.

Para poner a prueba esta propuesta se han realizado dos instanciaciones diferentes de los entornos virtuales de role-play. La primera, ViRPlay3D, ha sido desarrollada como una herramienta para el análisis del comportamiento en ejecución de una aplicación. Es un entorno monousuario en el que se han incluido las metáforas y las mecánicas más básicas de este tipo de entornos. De este modo se ha podido verificar que el traslado de las sesiones de role-play a estos entornos es abordable. La segunda instanciación, ViRPlay3D 2, se ha diseñado como un entorno en el que se simulan sesiones de role-play similares a las presenciales experimentadas para la enseñanza de patrones de diseño. En esta instanciación se han incluido todas las mecánicas relacionadas con la colaboración entre los alumnos, así como operaciones para la realización de refactorizaciones sencillas de un diseño de clases.

De acuerdo al diseño realizado para cada instanciación se han implementado sendos prototipos. Posteriormente, éstos han sido evaluados por docentes para valorar la idoneidad de la metáfora empleada en el entorno, así como la utilidad de la información contenida en el mismo y las mecánicas empleadas para fomentar la interactividad de los alumnos. Las observaciones realizadas por los participantes de la evaluación del primer prototipo también han sido empleadas para paliar los problemas y mejorar el diseño de la segunda instanciación.

A modo de resumen, las aportaciones principales concretas de este trabajo de Tesis son:

- Se han analizado en profundidad las principales dificultades existentes en la enseñanza de la orientación a objetos.
- Se ha realizado un análisis crítico de las distintas herramientas de apoyo a la docencia de la orientación a objetos basadas en visualizaciones interactivas. Se han identificado sus principales características y debilidades, y se ha detectado en qué medida abordan las dificultades propias de la orientación a objetos.
- Se ha analizado el uso del role-play como técnica de aprendizaje activo para la enseñanza de la orientación a objetos.
- Se ha evaluado el impacto en los alumnos de la utilización del role-play para la enseñanza de conceptos de diseño orientado a objetos (en particular, conceptos avanzados como los patrones de diseño).
- Se ha estudiado la aplicabilidad de trasladar las sesiones de role-play a entornos virtuales.
- Se ha diseñado un arquitectura general de los entornos virtuales de role-play basada en una probada arquitectura genérica para el desarrollo de videojuegos.

- Se han diseñado dos instanciaciones de dicha arquitectura general para verificar la viabilidad de la propuesta realizada.

- Se ha realizado una evaluación de ambas instanciaciones para validar la aceptación de los entornos virtuales de role-play entre los docentes.

7.2. Trabajo futuro

El trabajo descrito en esta memoria muestra el resultado final del mismo después de haber estudiado distintas alternativas que nos condujesen hasta la elaboración de una propuesta para la aplicabilidad de las técnicas de aprendizaje activo en entornos virtuales para la enseñanza de orientación a objetos. Pero existen líneas que aún no has sido tratadas y que se presentan como futuras líneas de continuación a partir del trabajo presentado.

Tal vez el trabajo pendiente más evidente sea la realización de una evaluación más exhaustiva de este tipo de entornos. En este trabajo se ha expuesto la realización de una evaluación formativa de los entornos virtuales de role-play con docentes. Sin embargo, es necesario también evaluar las impresiones que sobre este tipo de entornos pueden tener los alumnos, principales destinatarios de estos entornos. Esta evaluación también debería incluir un estudio sobre la efectividad del uso de estos entornos virtuales en la enseñanza de la orientación a objetos. Para la realización de esta evaluación es necesario implementar un entorno virtual de role-play robusto, generar un conjunto suficientemente grande de escenarios de simulación (tarea simplificada por las herramientas de autoría desarrolladas) y elaborar un conjunto de experimentos con los que se pueda comparar el aprendizaje de alumnos que han empleado este tipo de entornos con aquellos que han seguido una formación más tradicional.

Aunque se ha realizado un gran esfuerzo para aportar herramientas de autoría que faciliten en gran medida a un instructor la generación de escenarios simulables en entornos virtuales de role-play, la creación de estos escenarios sigue requiriendo una cierta carga de trabajo por parte del instructor. Éste es responsable de generar varios repertorios de información con cada una de las herramientas propuestas. Mucha de esta información tiene elementos comunes que, en la mayoría de las ocasiones, han de ser cruzados y verificados manualmente por el propio instructor. Por esto, sería necesario realizar una mayor integración de las herramientas propuestas para que estas referencias cruzadas de información se realizaran de manera automática. Por otro lado, el instructor es un experto relacionado con temas de orientación a objetos que también ha de encargarse del diseño de los mapas de los escenarios. Una línea de trabajo para solventar estos problemas de autoría sería el estudio de la generación automática de escenarios a partir de la información de las clases y de los objetos contenidos en ejercicios desarrollados por el

instructor. Dentro de esta línea de trabajo se estudiaría la aplicabilidad a los entornos virtuales de role-play de otras líneas de trabajo de nuestro propio grupo de investigación, que proponen la separación de conocimiento del dominio a enseñar y del diseño del entorno en sistemas educativos basados en videojuegos (Gómez-Martín, 2008a,b).

Las aproximaciones pedagógicas diseñadas para las dos instanciaciones de los entornos virtuales de role-play han sido muy sencillas. Esto se debe a que el objetivo principal de este trabajo de Tesis ha sido el estudio del traslado del role-play a entornos virtuales. Una vez validada la viabilidad de este traslado existe toda una línea de trabajo sobre las posibilidades de tutoría inteligente aplicables en este tipo de entornos. En la arquitectura general propuesta para estos entornos virtuales se ha dejado el hueco —el módulo *Gestor pedagógico*— para incluir estos sistemas de tutoría inteligente. Sin embargo se ha dejado como objeto de trabajo futuro el estudio de las necesidades de información para estos sistemas inteligentes y las alternativas de tutorización que se pueden ofrecer en este tipo de entornos. En particular, y como consecuencia directa de nuestra experiencia previa, una de las posibles vías a explorar es la introducción de un modelo de enseñanza basada en casos (Jiménez-Díaz y Gómez-Albarrán, 2004; Jiménez-Díaz et al., 2004a).

Continuando dentro de las aproximaciones pedagógicas, los entornos virtuales de role-play son un buen banco de pruebas para el estudio de sistemas de tutoría inteligente en entornos colaborativos. En la segunda instanciación diseñada para este tipo de entornos, los alumnos trabajan solos o con un instructor humano que les acompaña utilizando la interfaz que el entorno proporciona para este cometido. Pero no se ha analizado la forma de que sea un tutor inteligente el que ayude a los alumnos dentro del entorno. Este tutor ha de ser capaz de tratar con varios alumnos y de ayudar a coordinarlos para que, entre todos, completen el escenario de simulación.

Los entornos virtuales de role-play propuestos generan bitácoras de las acciones de los estudiantes. Estas son empleadas por el instructor para el mantenimiento de la base de escenarios de simulación y para la evaluación de los escenarios realizados por los alumnos. Pero la monitorización de las acciones y eventos que ocurren dentro del entorno puede ser utilizada para generar una gran cantidad de información que abra distintas líneas de trabajo de estudio del comportamiento de los alumnos en estos entornos. Por un lado, esta información se puede emplear para analizar patrones de comportamiento de los alumnos y errores comunes. Con esto se puede estudiar la generación de modelos de estudiantes que, usados por un sistema de tutoría inteligente, podrían aportar un apoyo más personalizado a los alumnos durante la simulación de un escenario. Por otro lado, la monitorización del entorno se puede emplear para la generación de trazas de simulación anotadas semánticamente. Estas trazas pueden ser explotadas para la generación de herramientas de autoría de agentes que participen con los alumnos en la

simulación de escenarios interpretando el rol de otros objetos. Estos agentes pueden ser creados por los instructores a partir de la grabación del comportamiento de éstos durante la simulación de un escenario. Estos agentes almacenan las experiencias del experto del dominio —en nuestro caso, el instructor— y tendrán un comportamiento que, a ojos del alumno, aparente ser inteligente y valioso para su aprendizaje dentro del entorno.

Finalmente se propone estudiar el uso de este tipo de entornos en otras disciplinas. Existen otras áreas dentro de la Informática que aceptan también la metáfora de la antropomorfización y el paso de mensajes. Este es el caso de los agentes software o de los protocolos de red, entre otros. La ventaja de usar entornos virtuales de enseñanza es que facilitan el desarrollo de casi cualquier tipo de metáfora. Para el traslado a estas disciplinas sería necesario estudiar cuáles son las necesidades particulares de representación y de información para estos dominios y qué cambios han de realizarse sobre los entornos virtuales de role-play para la inclusión de estos nuevos elementos. Esto mismo puede ser aplicable a otras áreas fuera del campo de la Informática. Como se puede ver en la literatura, el role-play es empleado en muchas otras disciplinas. Tan sólo hay que encontrar el área de interés y estudiar, al igual que se ha hecho para la orientación a objetos, su aplicabilidad.

Bibliografía

Detrás de cada uno de esos libros hay un hombre.
Eso es lo que a mí me interesa.

Fahrenheit 451 (1966)

ACM, 2001. Computing Curricula 2001. Computer Science. Informe técnico, The Joint Task Force on Computing Curricula. ACM / IEEE Computer Society, 2001.

ACM, 2004. Software Engineering 2004. Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. Informe técnico, The Joint Task Force on Computing Curricula. ACM / IEEE Computer Society, 2004.

ACM, 2005. Computing Curricula: 2005 Overview Report. Informe técnico, The Joint Task Force on Computing Curricula. ACM / IEEE Computer Society, 2005.

Alexander, C., Ishikawa, S. y Silverstein, M. *A Pattern Language*. Oxford University Press, 1977.

Alphonce, C., Caspersen, M. y Decker, A. Killer "Killer examples"for Design Patterns. En *38th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2007)*, páginas 228–232. ACM Press, Covington, Kentucky, USA, 2007.

Alphonce, C. y Ventura, P. Object orientation in CS1-CS2 by design. En *7th annual conference on Innovation and Technology in Computer Science Education (ITiCSE 2002)*, páginas 70–74. ACM Press, Aarhus, Denmark, 2002.

Andrianoff, S. K. y Levine, D. B. Role Playing in an Object-Oriented World. En *33rd SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2002)*, páginas 121–125. ACM Press, Cincinnati, Kentucky, 2002.

Armstrong, D. J. The Quarks of Object-Oriented Development. *Communications of the ACM*, vol. 49(2), páginas 123–128, 2006.

Astrachan, O., Mitchener, G., Berry, G. y Cox, L. Design Patterns: An Essential Component of CS Curricula. En *29th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 1998)*, páginas 153–160. ACM Press, Atlanta, Georgia, United States, 1998.

Bacvanski, V. y Börstler, J. Doing Your First OO Project - Discussion Paper. En *Educators Notes of the Educators Symposium at OOPSLA97*. ACM Press, Atlanta, Georgia, United States, 1997.

Barnes, D. J. Teaching Introductory Java through LEGO MINDSTORMS Models. En *33rd SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2002)* (editado por J. L. Gersting, H. M. Walker y S. Grissom), páginas 147–151. ACM Press, Cincinnati, Kentucky, USA, 2002.

Barnes, D. J. y Kölling, M. *Object First with Java. A practical introduction using BlueJ*. Pearson Education, Harlow, United Kingdom, segunda edición, 2005.

Beck, K. y Cunningham, W. A Laboratory For Teaching Object-Oriented Thinking. En *Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA 1989)*, páginas 1–6. ACM Press, New Orleans, Louisiana, United States, 1989.

Becker, B. W. Teaching CS1 with Karel the Robot in Java. En *32nd SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2001)*, páginas 50–54. ACM Press, Charlotte, North Carolina, United States, 2001.

Becker, B. W. *Java: Learning to Program with Robots*. Thomson Course Technology, 2006.

Beer, R. D., Chiel, H. J. y Drushel, R. F. Using autonomous robotics to teach science and engineering. *Communications of the ACM*, vol. 42(6), páginas 85–92, 1999.

Bellin, D. y Suchman-Simone, S. *The CRC Card Book*. Addison-Wesley, Massachusetts, USA, 1997.

Ben-Ari, M., Myller, N., Sutinen, E. y Tarhio, J. Perspectives on Program Animation with Jeliot. En *Software Visualization* (editado por S. Diehl), vol. 2269 de *Lecture Notes in Computer Science*, páginas 31–45. Springer-Verlag, Dagstuhl Castle, Germany, 2002.

Berge, O., Fjuk, A., Groven, A. K., Hegna, H. y Kaasbøll, J. Comprehensive object-oriented learning - an introduction. *Journal of Computer Science Education*, vol. 13(4), páginas 331–335, 2003.

Bergin, J. Karel Universe Drag & Drop Editor. *ACM SIGCSE Bulletin*, vol. 38(3), páginas 307–307, 2006.

Bergin, J., Eckstein, J., Wallingford, E. y Manns, M. L. Patterns for Gaining Different Perspectives. En *8th Conference on Pattern Languages of Programs (PLoP 2001)*. Monticello, Illinois, USA, 2001.

Bergin, J., Stehlik, M., Roberts, J. y Pattis, R. *Karel++. A Gentle Introduction to the Art of Object-Oriented Programming*. John Wiley & Sons, Inc., New York, 1997.

Bergin, J., Stehlik, M., Roberts, J. y Pattis, R. *Karel J. Robot. A Gentle Introduction to the Art of Object-Oriented Programming in Java*. Dream Songs Press, 2005.

Biddle, R., Noble, J. y Tempero, E. Techniques for Active Learning of OO Development. En *16th ACM SIGPLAN conference on Object Oriented Programming, Systems, Languages, and Applications Educators Symposium (OOPSLA 2001)*. Tampa Bay, USA, 2001.

Bierre, K. J. y Phelps, A. M. The Use of MUPPETS in an Introductory Java Programming course. En *5th Conference on Information Technology Education (CITC 2004)*, páginas 122–127. ACM, Salt Lake City, UT, USA, 2004.

Birnbaum, B. E. y Goldman, K. J. Achieving Flexibility in Direct-Manipulation Programming Environments by Relaxing the Edit-Time Grammar. En *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2005)*, páginas 259–266. IEEE Computer Society, Dallas, Texas, USA, 2005.

Booch, G. *Object-Oriented Analysis and Design with Applicattions*. The Benjamin/Cummings Publishing Company, segunda edición, 1994.

Boogaarts, M., Daudelin, J. A., Davis, B. L., Kelly, J., Morris, L., Rhodes, F., Rhodes, R., Scholz, M. P., Smith, C. R. y Torok, R. *The LEGO MINDSTORMS NXT Idea Book: Design, Invent, and Build*. William Pollock, San Francisco, USA, 2007.

Boytchev, P. Logo tree project. Disponible en `http://www.elica.net/download/papers/LogoTreeProject.pdf`. (Último acceso: 15-02-2008).

Bærbak-Christensen, H. Frameworks: Putting Design Patterns into Perspective. En *9th Aannual Conference on Innovation and Technology*

in Computer Science Education (ITiCSE 2004), páginas 142–145. ACM Press, Leeds, United Kingdom, 2004.

Brown, C., Fell, H., Proulx, V. K. y Rasala, R. Instructional Frameworks: Toolkits and Abstractions in Introductory Computer Science. En *1993 ACM Conference on Computer Science (CSC 1993)*, páginas 195–200. ACM Press, Indianapolis, Indiana, United States, 1993.

Börstler, J. Improving CRC-card role-play with role-play diagrams. En *20th annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA 2005)*, páginas 356–364. ACM Press, San Diego, CA, USA, 2005.

Börstler, J., Nordström, M., Westin, L. K., Moström, J.-E. y Eliasson, J. Transitioning to OOP–A Never Ending Story. En *Scandinavian Pedagogy of Programming* (editado por M. Kölling, J. Bennedsen y M. Caspersen). Springer, 2007.

Börstler, J. y Schulte, C. Teaching Object Oriented Modelling with CRC Cards and Roleplaying Games. En *8th IFIP World Conference on Computers in Education*. Cape Town, South Africa, 2005.

Bruce, K. B. Controversy on How to Teach CS 1: A discussion on the SIGCSE-members mailing list. *SIGCSE Bulletin inroads*, vol. 36(4), páginas 29–35, 2004.

Bruce, K. B., Danyluk, A. P. y Murtagh, T. P. A library to support a graphics-based object-first approach to CS1. En *32nd SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2001)*, páginas 6–10. ACM Press, Charlotte, North Carolina, United States, 2001a.

Bruce, K. B., Danyluk, A. P. y Murtagh, T. P. Event-driven programming is simple enough for CS1. En *6th annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2001)*, páginas 1–4. ACM Press, Canterbury, United Kingdom, 2001b.

Bruce, K. B., Danyluk, A. P. y Murtagh, T. P. Event-driven programming facilitates learning standard programming concepts. En *19th annual ACM SIGPLAN conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA 2004)*, páginas 96 – 100. ACM Press, Vancouver, Canada, 2004.

Bruce, K. B., Danyluk, A. P. y Murtagh, T. P. *Java. An Eventful Approach*. Prentice Hall, Harlow, England, 2005.

Buck, D. y Stucki, D. J. JKarelRobot: a case study in supporting levels of cognitive development in the computer science curriculum. En *32th*

SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2001), páginas 16–20. ACM Press, Charlotte, North Carolina, United States, 2001.

Budd, T. A brief introduction to Smalltalk. En *HOPL-II: The second ACM SIGPLAN conference on History of programming languages*, páginas 367–368. ACM Press, New York, NY, USA, 1993.

Budd, T. *An Introduction to Object-Oriented Programming, 3rd Edition*. Addison-Wesley, New York, 2002.

Byous, J. Java Technology: The Early Years. Disponible en `http://java.sun.com/features/1998/05/birthday.html`. (Último acceso: 15-02-2008).

Chambers, C., Harrison, B. y Vlissides, J. A debate on language and tool support for design patterns. En *27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, páginas 277–289. ACM Press, Boston, Massachusetts, USA, 2000.

Clancy, M. J. y Linn, M. C. Patterns and pedagogy. En *30th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 1999)*, páginas 37–42. ACM Press, New Orleans, Louisiana, United States, 1999.

Conway, M., Audia, S., Burnette, T., Cosgrove, D., Christiansen, K., Deline, R., Durbin, J., Gossweiler, R., Koga, S., Long, C., Mallory, B., Miale, S., Monkaitis, K., Patten, J., Pierce, J., Shochet, J., Staack, D., Stearns, B., Stoakley, R., Sturgill, C., Viega, J., White, J., Williams, G. y Pausch, R. Alice: lessons learned from building a 3D system for novices. En *SIGCHI Conference on Human Factors in Computing Systems*, páginas 486 – 493. ACM Press, The Hague, The Netherlands, 2000.

Cooper, S., Dann, W. y Pausch, R. Alice: a 3-D tool for Introductory Programming Concepts. *The Journal of Computing in Small Colleges*, vol. 15(5), páginas 107–116, 2000.

Cooper, S., Dann, W. y Pausch, R. Teaching Objects-first In Introductory Computer Science. En *34th SIGCSE Technical Symposium on Computer Science Education*, páginas 191–195. ACM Press, Reno, Navada, USA, 2003.

Cross II, J. H., Hendrix, T. D., Jain, J. y Barowski, L. A. Dynamic object viewers for data structures. En *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2007)*, páginas 4–8. ACM, Covington, Kentucky, USA, 2007.

Dann, W., Cooper, S. y Pausch, R. Making the Connection: Programming with Animated Small World. En *5th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2000)*, páginas 41–44. ACM Press, Helsinki, Finland, 2000.

Dann, W., Cooper, S. y Pausch, R. Using Visualization To Teach Novices Recursion. En *6th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2001)*, páginas 109 – 112. ACM-Press, Canterbury, United Kingdom, 2001.

Dann, W., Cooper, S. y Pausch, R. *Learning to Program with Alice*. Prentice Hall, Harlow, England, 2005.

Dann, W., Dragon, T., Cooper, S., Dietzler, K., Ryan, K. y Pausch, R. Objects: Visualization of Behavior and State. En *8th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2003)*, páginas 84–88. ACM Press, Thessaloniki, Greece, 2003.

Dillenbourg, P. Learning in the New Millennium, Building New Education Strategies for Schools. 2000. EUN Conference.

Dorn, B. y Sanders, D. Using Jeroo to Introduce Object-Oriented Programming. En *2003 IEEE/ASEE Frontiers in Education Conference*, vol. 1, páginas T4C 22–T4C 27. IEEE Computer Society, Westminster, Colorado, USA, 2003.

Ducasse, S. y Wuyts, R. Supporting Objects as an Anthropomorphic View at Computation or Why Smalltalk for Teaching Objects. En *Sixth Workshop on Pedagogies and Tools for Learning Object Oriented Concepts on 16th European Conference Object-Oriented Programming (ECOOP 2002)*, vol. 2374 de *Lecture Notes in Computer Science*. Springer, Malaga, Spain, 2002.

Eckstein, J. Incremental Role Play. En *3rd European Conference on Pattern Languages of Programming and Computing (PLoP 1998)*. Bad Irsee, Germany, 1998.

Edwards, L. D. Embodying mathematics and science: Microworlds as representations. *Journal of Mathematical Behaviour*, vol. 17(1), páginas 53–78, 1998.

Esteves, M. y Mendes, A. A Simulation Tool to Help Learning of Object Oriented Programming Basics. En *34th ASEE/IEEE Frontiers in Education Conference*, páginas F4C–7 – F4C–12. 2004.

Fagin, B. S. y Merkle, L. Quantitative analysis of the effects of robots on introductory computer science education. *Journal on Educational Resources in Computing*, vol. 2(4), páginas 1–17, 2002.

Fell, H., Proulx, V. K. y Rasala, R. Foundations of Computer Science: What Are They and How Do We Teach Them. En *1st Conference on Integrating Technology into Computer Science Education (ITiCSE 1996)*, páginas 42–48. ACM Press, Barcelona, Spain, 1996.

Ferrari, M., Ferrari, G. y Hempel, R. *Building Robots with LEGO Mindstorms: The Ultimate Tool for Mindstorms Maniacs*. Syngress Publishing, Rockland, MA, USA, 2001.

Fischer, T., Niere, J., Torunski, L. y Zündorf, A. Story Diagrams: A new Graph Rewrite Language based on the Unified Modeling Language. En *6th International Workshop on Theory and Application of Graph Transformation* (editado por G. Engels y G. Rozenberg), vol. LNCS 1764 de *LNCS*, páginas 296–309. Springer Verlag, Paderborn, Germany, 1998.

Flowers, T. R. y Gossett, K. A. Teaching problem solving, computing, and information technology with robots. *Journal of Computing Sciences in Colleges*, vol. 17(6), páginas 45–55, 2002.

Fowler, M. *Refactoring: Improving the Design of Existing Code*. Object Technology. Addison-Wesley Professional, Massachusetts, USA, 1999.

Gamma, E., Helm, R., Johnson, R. E. y Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley Professional, Massachusetts, 1995.

Gestwicki, P. V. Computer games as motivation for design patterns. En *38th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2007)*, páginas 233–237. ACM Press, Covington, Kentucky, USA, 2007.

Gómez-Martín, M. A. *Arquitectura y metodología para el desarrollo de sistemas educativos basados en videojuegos*. Tesis Doctoral, Universidad Complutense de Madrid, 2008a.

Gómez-Martín, P. P. *Modelos de enseñanza basado en casos: de los tutores inteligentes a los videojuegos*. Tesis Doctoral, Universidad Complutense de MAdrid, 2008b.

Goldman, K. J. Capsules and Semantic Regions for Code Visualization and Direct Manipulation of Live Programs. Informe Técnico TR-2004-78, Washington University, Department of Computer Science and Engineering, 2004a.

Goldman, K. J. A concepts-first introduction to computer science. En *35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2004)*, páginas 432–436. ACM Press, Norfolk, Virginia, USA, 2004b.

Goldman, K. J. An interactive environment for beginning java programmers. *Science of Computer Programming*, vol. 53(1), páginas 3–24, 2004c.

Gosling, J., Joy, B., Steele, G. y Bracha, G. *The Java Language Specification (Third Edition)*. Java Series. Addison-Wesley Professional, 2005.

Gosling, J. y McGilton, H. The Java Language Environment. Informe técnico, Sun Microsystems, 1996.

GridWorld. AP GridWorld Case Study. Disponible en `http://apcentral.collegeboard.com/apc/public/courses/teachers_corner/151155.html`. (Último acceso: 15-02-2008).

Guzdial, M., Kolodner, J., Hmelo, C., Narayanan, H., Carlson, D., Rappin, N., Hübscher, R., Turns, J. y Newstetter, W. Computer support for learning through complex problem solving. *Communications of the ACM*, vol. 39(4), páginas 43–45, 1996. 227600 0001-0782 ACM.

Hamer, J. An approach to teaching design patterns using musical composition. En *9th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2004)*, páginas 156–160. ACM Press, Leeds, United Kingdom, 2004.

Hendrix, T. D., James H. Cross, I. y Barowski, L. A. An extensible framework for providing dynamic data structure visualizations in a lightweight IDE. En *35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2004)*, páginas 387–391. ACM, Norfolk, Virginia, USA, 2004.

Henriksen, P. *A Direct Interaction Tool for Software Engineering Education*. Master thesis, University of Southern Denmark, 2004.

Henriksen, P. y Kölling, M. greenfoot: Combining Object Visualisation with Interaction. En *19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2004)* (editado por J. Vlissides y D. Schmidt), páginas 73–82. ACM Press, Vancouver, Canada, 2004.

Holland, S., Griffiths, R. y Woodman, M. Avoiding object misconceptions. En *28th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 1997)*, páginas 131–134. ACM Press, San Jose, California, United States, 1997.

Hsiao, I. H. y Brusilovsky, P. Collaborative Example Authoring System: The Value of Re-annotation based on Community Feedback. En *World Conference on E-Learning (E-Learn 2007)* (editado por T. Bastiaens y S. Carliner), páginas 7122–7131. AACE, Quebec, Canada, 2007.

Ip, A., Linser, R. y Naidu, S. Simulated Worlds: Rapid Generation of Web-Based Role-Play. En *7th Australasian World Wide Web Conference (AusWeb 2001)*. Coffs Harbour, Australia, 2001.

Jacobson, I., Booch, G. y Rumbaugh, J. *The Unified Software Development Process*. Addison-Wesley Publishing, 1999.

Jacobson, I., Christerson, M., Jonsson, P. y Overgaard, G. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, 1992.

JHotDraw. JHotDraw Website. Disponible en `http://www.jhotdraw.org/`. (Último acceso: 15-02-2008).

Jiménez-Díaz, G. y Gómez-Albarrán, M. A Case-Based Approach for Teaching Frameworks. En *PhD-Workshop at 18th European Conference on Object-Oriented Programming (ECOOP 2004)*. Oslo, Norway, 2004.

Jiménez-Díaz, G. y Gómez-Albarrán, M. Una revisión de los aspectos clave en el diseño de los sistemas de enseñanza basada en casos. *Revista Iberoamericana de Inteligencia Artificial*, vol. 9(27), páginas 7–19, 2005.

Jiménez-Díaz, G., Gómez-Albarrán, M., Gómez-Martín, M. A. y Gónzález-Calero, P. A. Visualization and Role-play to Teach Object-Oriented Programming. En *Computers and Education - Towards Educational Change and Innovation* (editado por A. J. Nunes-Mendes), páginas 167–177. Springer, London, UK, 2007a.

Jiménez-Díaz, G., Gómez-Albarrán, M., Gómez-Martín, M. A. y González-Calero, P. A. Software behaviour understanding supported by dynamic visualization and role-play. *10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE 2005). SIGCSE Bulletin*, vol. 37(3), páginas 54–58, 2005a.

Jiménez-Díaz, G., Gómez-Albarrán, M., Gómez-Martín, M. A. y González-Calero, P. A. Understanding Object-Oriented Software through Virtual Role-Play. En *5th IEEE International Conference on Advanced Learning Technologies (ICALT 2005)* (editado por P. Goodyear, D. G. Sampson, D. Yang, Kinshuk, T. Okamoto, R. Hartley y N.-S. Chen), páginas 875–877. IEEE Computer Society, Kaohsiung, Taiwan, 2005b.

Jiménez-Díaz, G., Gómez-Albarrán, M., Gómez-Martín, M. A. y González-Calero, P. A. ViRPlay: Playing Roles to Understand Dynamic Behavior. En *9th Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts, at 19th European Conference on Object Oriented Programming (ECOOP 2005)*. Glasgow, UK, 2005c.

Jiménez-Díaz, G., Gómez-Albarrán, M. y González-Calero, P. A. Applying Case-Based Teaching to Object-Oriented Framework Training. En *2nd European Starting AI Researcher Symposium (STAIRS 2004), in 16th European Conference in Artificial Intelligence (ECAI 2004)*, páginas 235–240. IOS Press, Valencia, Spain, 2004a.

Jiménez-Díaz, G., Gómez-Albarrán, M. y González-Calero, P. A. UnderFrame: Understanding Object-Oriented Frameworks Using a Case-Based Teaching Approach. En *Poster at 18th European Conference on Object-Oriented Programming (ECOOP 2004)*. Oslo, Norway, 2004b.

Jiménez-Díaz, G., Gómez-Albarrán, M. y González-Calero, P. A. Before and After: An Active and Collaborative Approach to Teach Design Patterns. En *8th International Symposium on Computers in Education* (editado por L. Panizo Alonso, B. Fernández Manjón, L. Sánchez González y M. Llamas Nistal), vol. 1, páginas 272–279. Servicio de Imprenta de la Universidad de León, León, Spain, 2006.

Jiménez-Díaz, G., Gómez-Albarrán, M. y González-Calero, P. A. Pass the Ball: Game-Based Learning of Software Design. En *6th International Conference on Entertainment Computing (ICEC 2007)* (editado por L. Ma, M. Rauterberg y R. Nakatsu), vol. 4740, páginas 49–54. Springer, Shanghai, China, 2007b.

Jiménez-Díaz, G., Gómez-Albarrán, M. y González-Calero, P. A. Role-Play Virtual Environments: Recreational Learning of Software Design. En *Second European Conference on Technology Enhanced Learning. Addendum to Conference Proceedings (EC-TEL 2007)* (editado por E. Duval, R. Klamma y M. Wolpers), páginas 23–27. Crete, Greece, 2007c.

Jiménez-Díaz, G., Gómez-Albarrán, M. y González-Calero, P. A. Teaching GoF Design Patterns through Refactoring and Role-Play. *International Journal of Engineering Education*, (Special Issue: Trends in Software Engineering Education), 2008. Aceptado y pendiente de publicación.

Jiménez-Díaz, G., González-Calero, P. A. y Gómez-Albarrán, M. Using Role-Play Virtual Environments to Learn Software Design. En *European Conference on Games-Based Learning (ECGBL 2007)* (editado por D. Remenyi), páginas 143–151. Academic Conferences, Paisley, Scotland, UK, 2007d.

Kaiser, W. Become a programming picasso with JHotDraw. *JavaWorld*, 2001.

Kannusmäki, O., Moreno, A., Myller, N. y Sutinen, E. What a Novice Wants: Students Using Program Visualization in Distance Programming Course. En *Third Program Visualization Workshop* (editado por A. Korhonen), páginas 126–133. Research Report CS-RR-407, Department of Computer Science, University of Warwick, Coventry, UK, 2004.

Kay, A. C. The early history of Smalltalk. En *HOPL-II: The second ACM SIGPLAN Conference on History of Programming Languages*, páginas 69–95. ACM Press, New York, NY, USA, 1993.

Kerievsky, J. *Refactoring to Patterns*. Addison-Wesley Signature Series. Addison-Wesley Professional, Massachusetts, USA, 2004.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Videira Lopes, C., Loingtier, J.-M. y Irwin, J. Aspect-Oriented Programming. En *11th European Conference on Object-Oriented Programming (ECOOP 1997)* (editado por M. Aksit y S. Matsuoka), vol. 1241 de *LNCS*, páginas 220–242. Springer, Jyväskylä, Finland, 1997.

Kirk, D. Framework reuse: Process, problems and documentation. Informe Técnico EFoCS-43-2001, EFoCS research group - Strathclyde University, 2001.

Kirk, D. Evaluation of a pattern language for jhotdraw. Informe Técnico EFoCS-48-2002, EFoCS research group - Strathclyde University, 2002.

Klassner, F. A case study of LEGO MINDSTORMS suitability for artificial intelligence and robotics courses at the college level. En *33rd SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2002)* (editado por J. L. Gersting, H. M. Walker y S. Grissom), páginas 8–12. ACM Press, Cincinnati, Kentucky, USA, 2002.

Kölling, M. The problem of teaching object-oriented programming, part 1: Languages. *Journal of Object-Oriented Programming*, vol. 11(8), páginas 8–15, 1999a.

Kölling, M. The problem of teaching object-oriented programming, part 2: Environments. *Journal of Object-Oriented Programming*, vol. 11(9), páginas 6–12, 1999b.

Kölling, M. y Henriksen, P. Game programming in introductory courses with direct state manipulation. En *10th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2005)*, páginas 59–63. ACM Press, Caparica, Portugal, 2005.

Kölling, M. y Rosenberg, J. An Object-Oriented Program Development Environment for the First Programming Course. En *27th SIGCSE Techni-*

cal Symposium on Computer Science Education (SIGCSE 1996), páginas 83–87. ACM Press, Philadelphia, Pennsylvania, 1996a.

Kölling, M. y Rosenberg, J. Blue - A Language for Teaching Object-Oriented Programming. En *27th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 1996)*, páginas 190–194. ACM Press, Philadelphia, Pennsylvania, 1996b.

Kölling, M. y Rosenberg, J. Guidelines for Teaching Object Orientation with Java. En *6th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2001)*, páginas 33 – 36. ACM Press, Canterbury, United Kingdom, 2001.

Kölling, M. y Rosenberg, J. The BlueJ system and its pedagogy. *Journal of Computer Science Education, Special issue on Learning and Teaching Object Technology*, vol. 13(4), páginas 249–268, 2003.

Kolodner, J. L. Educational implications of analogy. a view from case-based reasoning. *American Psychologist*, vol. 52(1), páginas 57–66, 1997.

Lawhead, P. B., Bland, C. G., Barnes, D. J., Duncan, M. E., Goldweber, M., Hollingsworth, R. G. y Schep, M. A Road Map for Teaching Introductory Programming Using LEGO© Mindstorms Robots. *ACM SIGCSE Bulletin*, vol. 35(2), páginas 191–201, 2003.

Liskov, B. H. y Wing, J. M. A Behavioral Notion of Subtyping. *ACM Transactions on Programming Languages and Systems*, vol. 16(6), páginas 1811–1841, 1994.

Martin, R. C. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.

MBSCS. AP Marine Biology Simulation Case Study. Disponible en `http://apcentral.collegeboard.com`. (Último acceso: 09-01-2006).

McShaffry, M. *Game Coding Complete*. Paraglyph Press, Arizona, USA, 2nd edición, 2005.

Meco-Alías, A., Martínez-de Leiva-Nieto, D. y Alonso-Navarro, M. GAMA: Generador Automático de Modelos Animados. Informe técnico, Facultad de Informática, Universidad Complutense de Madrid, 2006.

Meyer, B. *Object-Oriented Software Construction, second edition*. Prentice Hall, 1997.

Mitchell, R., McKim, J. y Meyer, B. *Design by Contract, by Example*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2002.

Moreno, A., Myller, N., Sutinen, E. y Ben-Ari, M. Visualizing Programs with Jeliot 3. En *Working Conference on Advanced Visual Interfaces (AVI 2004)*, páginas 373–376. ACM Press, Gallipoli, Italy, 2004.

Myller, N. *The Fundamental Design Issues of Jeliot 3*. Master thesis, University of Joensuu, 2004.

Naidu, S., Ip, A. y Linser, R. Dynamic goal-based role-play simulation on the web: A case study. *Educational Technology & Society*, vol. 3(3), 2000.

Naps, T. L., Rössling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S. y Velázquez-Iturbide, J. A. Exploring the Role of Visualization and Engagement in Computer Science Education. En *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE 2002)* (editado por M. E. Caspersen, D. Joyce, D. Goelman y I. Utting), páginas 131–152. ACM Press, Århus, Denmark, 2002.

Nevison, C. y Wells, B. Teaching objects early and design patterns in Java using case studies. En *8th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2003)*, páginas 94–98. ACM Press, Thessaloniki, Greece, 2003.

Nevison, C. y Wells, B. Using a Maze Case Study to Teach Object-Oriented Programming Design Patterns. En *6th Australasian Computing Education Conference (ACE 2004)*, vol. 30, páginas 207–215. ACM Press, Dunedin, New Zealand, 2004.

Niere, J. y Schulte, C. Thinking in Object Structures: Teaching Modelling in Secondary Schools. En *Workshop on Pedagogies and Tools for Learning Object-Oriented Concepts (ECOOP 2002)*. Malaga, Spain, 2002.

Nygaard, C. OO is Easy to Learn but Seldom Taught. En *Invited talk in the 2001 Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA 2001)*. Tampa Bay, Florida, USA, 2001.

Nygaard, K. A Sufficiently Complex Example. Disponible en `http://www.intermedia.uio.no/cool/complex.htm`. (Último acceso: 20-04-2006).

Nygaard, K. y Dahl, O.-J. The development of the SIMULA languages. páginas 439–480. ACM Press, New York, NY, USA, 1981.

Papert, S. *Mindstorms: children, computers, and powerful ideas*. Basic Books, Inc., 1980.

Patterson-McNeill, H. y Binkerd, C. L. Resources for Using Lego® MindstormsTM. *Journal of Computing Sciences in Colleges*, vol. 16(3), páginas 48–55, 2001.

Pattis, R. E. *Karel the Robot: A Gentle Introduction to the Art of Programming*. John Wiley & Sons, 1995.

Pausch, R., Burnette, T., Capeheart, A., Conway, M., Cosgrove, D., DeLine, R., Durbin, J., Gossweiler, R., Koga, S. y White, J. Alice: Rapid prototyping system for virtual reality. *IEEE Computer Graphics and Applications*, vol. 15(3), páginas 8–11, 1995.

Phelps, A. M., Bierre, K. J. y Parks, D. M. MUPPETS: Multi-user Programming Pedagogy for Enhancing Traditional Study. En *4th Conference on Information Technology Curriculum (CITC 2003)*, páginas 100–105. ACM, Lafayette, Indiana, USA, 2003.

Phelps, A. M. y Parks, D. M. Fun and games: Multi-language development. *Queue*, vol. 1(10), páginas 46–56, 2004.

Piaget, J. *The construction of reality in the child*. Basic Books, 1955.

Price, B. A., Baecker, R. M. y Small, I. S. A principled taxonomy of sofware visualization. *Journal of Visual Languages and Computing*, vol. 4(3), páginas 211–266, 1993.

Proulx, V. K. Traffic simulation: a case study for teaching object oriented design. En *29th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 1998)*, páginas 48–52. ACM Press, Atlanta, Georgia, United States, 1998.

Proulx, V. K. Hospital emergency room simulation: object oriented design issues for CS2. En *30th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 1999)*, páginas 92–94. ACM Press, New Orleans, Louisiana, United States, 1999.

Proulx, V. K., Raab, J. y Rasala, R. Objects From the Beginning - With GUIs. En *7th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2002)*, páginas 65–69. ACM Press, Aarhus, Denmark, 2002.

Proulx, V. K., Rasala, R. y Fell, H. Foundations of Computer Science: What are they and how do we teach them? En *1st Annual Conference on Integrating Technology into Computer Science Education (ITiCSE 1996)*, páginas 42–48. ACM Press, Barcelona, Spain, 1996.

Proulx, V. K., Rasala, R. y Raab, J. Java Power Tools: A Foundation for Interactive HCI Exploration. En *9th International Conference on*

Human-Computer Interaction, vol. 2, páginas 755–759. Lawrence Eldbaum Associates, New Orleans, LA, USA, 2001.

Raab, J., Rasala, R. y Proulx, V. K. Pedagogical Power Tools for Teaching Java. En *5th annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE 2000)*, páginas 156–159. ACM Press, Helsinki, Finland, 2000.

Rasala, R. Toolkits in first year computer science: a pedagogical imperative. En *31st SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2000)*, páginas 185–191. ACM Press, Austin, Texas, United States, 2000.

Rasala, R. y Proulx, V. K. Java Power Tools. Disponible en `http://www.ccs.neu.edu/jpt/`. (Último acceso: 15-02-2008).

Rasala, R., Raab, J. y Proulx, V. K. Java Power Tools: Model Software for Teaching Object-Oriented Design. En *32nd SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2001)*, páginas 297–301. ACM Press, Charlotte, North Carolina, United States, 2001.

Rasala, R., Raab, J. y Proulx, V. K. The SIGCSE 2001 Maze Demonstration program. En *33rd SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2002)* (editado por J. L. Gersting, H. M. Walker y S. Grissom), páginas 287–291. ACM Press, Cincinnati, Kentucky, USA, 2002.

Rentsch, T. Object Oriented Programming. *ACM SIGPLAN Notices*, vol. 17(9), páginas 51–57, 1982. ISSN 0362-1340.

Ryan, C., Al-Qaimari, G. y Langan-Fox, J. Teaching Object-Oriented Analysis and Design: A Cognitive Approach. En *World Multiconference on Systemics, Cybernetics and Informatics (SCI 1997)*, páginas 380–388. Caracas, Venezuela, 1997.

Sanders, D. y Dorn, B. Classroom Experience With Jeroo. *Journal of Computing Sciences in Colleges*, vol. 18(4), páginas 308–316, 2003a.

Sanders, D. y Dorn, B. Jeroo: a tool for introducing object-oriented programming. En *34th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2003)*, páginas 201–204. ACM Press, Reno, Navada, USA, 2003b.

Sanders, D. y Dorn, B. Object-Oriented Programming with Jeroo in the Information Technology Classroom. En *21st Annual Information Systems Education Conference*, vol. 21. AITP Foundation for Information Technology Education, Rhode Island, USA, 2004.

Schank, R. C. y Cleary, C. *Engines for education*. L. Erlbaum Associates Publishers, Hillsdale, NJ, 1995.

Schmolitzky, A. A Laboratory for Teaching Object-Oriented Language and Design Concepts with Teachlets. En *20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2005)*, páginas 332–337. ACM Press, San Diego, CA, USA, 2005.

Schulte, C. y Bennedsen, J. What do Teachers Teach in Introductory Programming? En *2006 International Workshop on Computing Education Research*, páginas 17–28. ACM Press, Canterbury, United Kingdom, 2006.

Schulte, C., Magenheim, J., Niere, J. y Schäfer, W. Thinking in Objects and their Collaboration: Introducing Object-Oriented Technology. *Computer Science Education*, vol. 13(4), páginas 269–288, 2003.

Schumacher, J., Welch, D. y Raymond, D. Teaching Introductory Programming, Problem Solving and Information Technology with Robots at West Point. En *31st ASEE/IEEE Frontiers in Education Conference*, páginas F1B–2 – F1B–7. IEEE, Reno, Nevada, USA, 2001.

Sharf, J., Hsu, T. y Ryan, E. TimeScope: an Educational 3D Multi-User Role-Playing Environment. En *9th Annual Conference of the Internet Society*. San Jose, CA, USA, 1999.

Smith, S., Stoecklin, S. y Serino, C. An innovative approach to teaching refactoring. En *37th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2006)*, páginas 349–353. ACM Press, Houston, Texas, USA, 2006.

Stoecklin, S., Smith, S. y Serino, C. Teaching students to build well formed object-oriented methods through refactoring. En *38th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2007)*, páginas 145–149. ACM Press, Covington, Kentucky, USA, 2007.

Strother, J. B. An assessment of the effectiveness of e-learning in corporate training programs. *The International Review of Research in Open and Distance Learning*, vol. 3(1), 2002.

Stroustrup, B. A history of C++: 1979–1991. En *HOPL-II: The second ACM SIGPLAN conference on History of programming languages*, páginas 271–297. ACM Press, New York, NY, USA, 1993.

Stroustrup, B. Evolving a language in and for the real world: C++ 1991–2006. En *HOPL III: Proceedings of the third ACM SIGPLAN conference on History of programming languages*, páginas 4–1–4–59. ACM Press, New York, NY, USA, 2007.

Venners, B. How to Use Design Patterns. A Conversation with Erich Gamma, Part I. *Leading-Edge Java*, 2005.

Warren, I. Teaching Patterns and Software Design. En *7th Australasian Computing Education Conference (ACE 2005)* (editado por A. Young y D. Tolhurst), vol. 42, páginas 39–49. ACS, Newcastle, Australia, 2005.

West, D. *Object thinking*. Microsoft Press, 2004.

Wirfs-Brock, R. J. y McKean, A. *Object Design: Roles, Responsibilities, and Collaborations*. Addison Wesley Professional, Boston, MA, 2003.

Wolfe, D., Gosset, K., Hanlon, P. D. y Carver-Jr., A. A. Active Learning using Mechatronics in a Freshman Information Technology Course. En *33rd ASEE/IEEE Frontiers in Education Conference*, páginas S1D–24 – S1D–28. IEEE, Boulder, Colorado, USA, 2003.

Wolz, U. Teaching design and project management with LEGO RCX robots. En *32nd SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2001)* (editado por H. M. Walker, R. A. McCauley, J. L. Gersting y I. Russell), páginas 95–99. ACM Press, Charlotte, North Carolina, USA, 2001.

Y, finalmente,
me puse a bailar.